

成绩	
----	--

金陵科技学院

# 人工智能大作业

(理工类)



课程名称: 人工智能 专业班级: 24 自动化 (W 专转本)

学生学号: 2417504036 学生姓名: 陈亚鹏

所属院部: 智能科学与控制工程学院 指导老师: 张 艳

2024——2025 学年 第二 学期

金陵科技学院教务处制

# 基于神经网络的系统辨识

## 目录

1	绪论	1
1.1	研究背景与意义	1
1.2	本文研究内容的安排	1
2	算法原理介绍	2
2.1	一阶倒立摆的动力学方程	2
2.2	神经网络的工作原理	3
2.3	MLP 结构图	3
2.4	系统辨识流程图	3
3	程序与运行结果	5
3.1	系统辨识程序	5
3.2	仿真结果	6
3.3	PID 校正	7
4	总结	11
	参考文献	11

## 1 绪论

### 1.1 研究背景与意义

随着人工智能技术的快速发展，神经网络在系统辨识中的应用得到了广泛关注。系统辨识是通过实验数据建立动态系统数学模型的过程，在控制工程、机器人学和工业自动化等领域具有重要意义。杨忠等 (2021) 然而，传统的系统辨识方法在处理复杂非线性系统时存在局限性，而神经网络凭借其强大的非线性拟合能力，为系统辨识提供了一种高效的解决方案。神经网络的系统辨识方法可以通过输入-输出数据直接拟合系统的动态特性，避免了复杂的物理建模过程。这种方法不仅提高了辨识精度，还显著降低了计算成本，为复杂系统的建模和控制提供了新的思路。

### 1.2 本文研究内容的安排

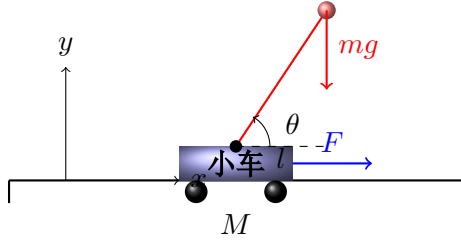
本文主要研究基于神经网络的一阶倒立摆系统辨识方法，并结合 PID 控制器对系统进行校正。全文内容安排如下：

- 第一节绪论：介绍研究背景与意义，并说明本文的研究内容。
- 第二节算法原理介绍：阐述倒立摆的动力学演化过程，以及神经网络在系统辨识中的工作原理，并绘制多层感知机（MLP）的结构图和流程图。
- 第三节程序与运行结果：基于倒立摆模型进行系统辨识，给出程序代码和仿真结果，并结合 PID 控制器对系统进行校正。
- 第四节总结：总结本文的研究内容和成果，并展望未来的研究方向。

## 2 算法原理介绍

### 2.1 一阶倒立摆的动力学方程

倒立摆是一个典型的非线性、不稳定系统。目标是通过控制输入（如力矩或推力）使摆杆保持在直立平衡位置（目标角度为 0 弧度）。



写下动力学方程:

$$\frac{d^2\theta}{dt^2} + \frac{b}{m} \frac{d\theta}{dt} - \frac{g}{l} \sin(\theta) = \frac{\tau}{ml^2} \quad (1)$$

相空间方程:

$$\begin{cases} \dot{\theta} = \omega \\ \dot{\omega} = g/l \sin(\theta) - b/m \omega + \tau/(ml^2) \end{cases} \quad (2)$$

其中:

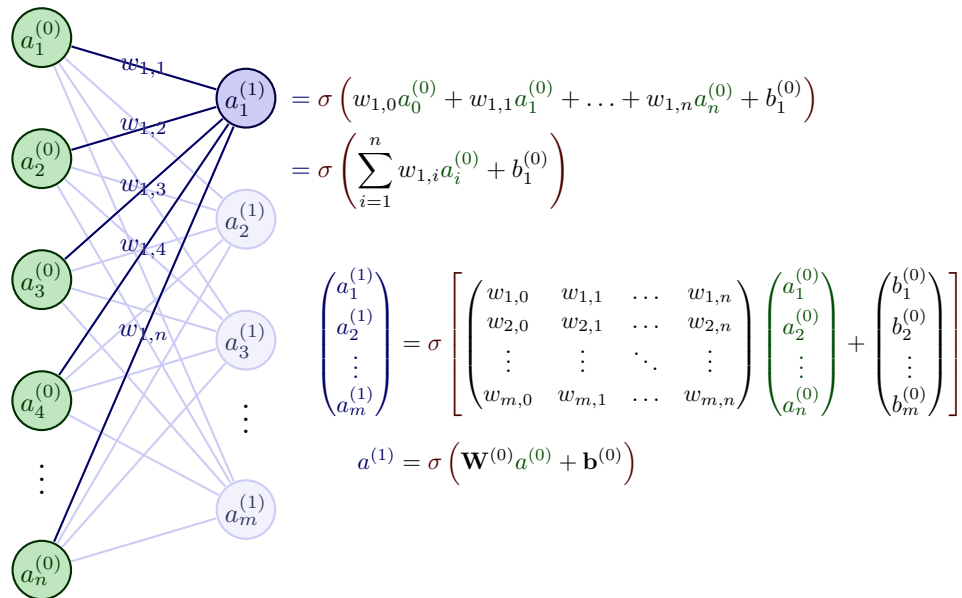
- $\theta$ : 摆角（角度）。
- $\dot{\theta}$ : 角速度。
- $\ddot{\theta}$ : 角加速度。
- $g$ : 重力加速度。
- $l$ : 摆长。
- $b$ : 阻尼系数。
- $m$ : 摆的质量。
- $\tau$ : 控制力矩。

## 2.2 神经网络的工作原理

神经网络是一种模拟生物神经系统的计算模型，具有强大的非线性拟合能力。多层感知机（MLP）是神经网络的基本结构，由输入层、隐藏层和输出层组成周志华 (2016)。通过数据训练神经网络，进行非线性系统辨识通过拟合系统的输入与输出之间的非线性映射关系，从而建立一个能够近似描述系统动态行为的模型，从而实现系统辨识。

## 2.3 MLP 结构图

以下是 MLP 结构图以及权重、阈值更新公式：



## 2.4 系统辨识流程图

以下是基于神经网络的系统辨识流程图：

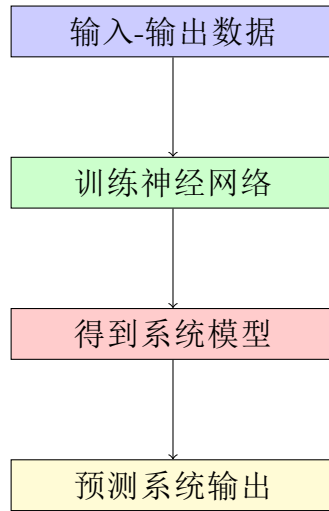


图 2.4.1: 基于神经网络的系统辨识流程图

## 详细步骤







- 数据采集** 收集系统的输入 (  $u(t)$  ) 和输出 (  $y(t)$  ) 数据, 形成数据集:  
 $\mathcal{D} = (u(t), y(t)) \mid t = 1, 2, \dots, N$
- 数据预处理** 对数据进行归一化或标准化处理:  $u'(t) = \frac{u(t) - \mu_u}{\sigma_u}$ ,  $y'(t) = \frac{y(t) - \mu_y}{\sigma_y}$
- 神经网络建模** 选择神经网络结构 (如前馈网络、递归网络) 并定义模型:  
 $\hat{y}(t) = f_{\theta}(u(t), y(t-1), y(t-2), \dots)$
- 损失函数定义** 定义损失函数 (如均方误差) 以衡量预测值与真实值的差异:  
 $\mathcal{L}(\theta) = \frac{1}{N} \sum_{t=1}^N (y(t) - \hat{y}(t))^2$
- 模型训练** 使用优化算法 (如梯度下降、Adam) 最小化损失函数:  $\theta^* = \arg \min_{\theta} \mathcal{L}(\theta)$
- 模型验证** 在验证集上评估模型性能, 计算误差指标 (如均方误差、平均绝对误差):  $\text{MSE} = \frac{1}{M} \sum_{t=1}^M (y_{\text{true}}(t) - \hat{y}(t))^2$
- 模型部署** 将训练好的模型用于实时预测或控制:  $\hat{y}(t) = f_{\theta^*}(u(t), y(t-1), y(t-2), \dots)$

### 3 程序与运行结果

### 3.1 系统辨识程序

开始训练一个神经网络, 这里仅展示核心代码 (示例程序):

- 定义动力学方程模型，生成数据集：

Python editor |      

```

# 倒立摆动力学方程
def inverted_pendulum(state, t, g, l, b, m, tau):
    theta, omega = state
    dtheta_dt = omega
    domega_dt = (g / l) * np.sin(theta) - (b / m) * omega\
        + tau / (m * l**2)
    return [dtheta_dt, domega_dt]

# 数据生成
def generate_data():
    g, l, b, m = 9.8, 1.0, 0.1, 1.0 # 参数
    t = np.linspace(0, 10, 1000) # 时间
    tau = 0 # 控制力矩
    y0 = [0.1, 0.0] # 初始条件 [初始角度, 初始角速度]
    data = odeint(inverted_pendulum, y0, t,\
        args=(g, l, b, m, tau))
    theta, omega = data[:, 0], data[:, 1]
    dtheta_dt = omega
    domega_dt = (g / l) * np.sin(theta) - (b / m) * omega\
        + tau / (m * l**2)
    X = np.column_stack((theta, omega,\
        tau * np.ones_like(theta))) # 输入 [角度, 角速度, 控制力矩]
    y = domega_dt # 输出 [角加速度]
    return X, y, t, data

```

- 训练神经网络

```
Python editor | 文件 | 编辑 | 视图 | 运行 | 窗口 | 帮助 |
# 生成数据
X, y, t, data = generate_data()

# 划分训练集和测试集
X_train, X_test, y_train, y_test = \
train_test_split(X, y, test_size=0.2, random_state=42)

# 神经网络建模
model = MLPRegressor(hidden_layer_sizes=(128, 128), \
activation='relu', max_iter=5000, random_state=42)
model.fit(X_train, y_train)

# 预测
y_pred = model.predict(X_test)

# 评估
mse = mean_squared_error(y_test, y_pred)
print(f"Mean Squared Error: {mse}")
```

Mean Squared Error: 0.009241777330554795

这里的示例程序设置了两层隐藏层，每层神经元大小为 128，使用 Relu 作为激活函数，训练轮数为 5000 轮。神经网络接受的输入参数有当前摆的角度  $\theta$ ，角加速度  $\omega$  以及控制力矩  $\tau$ ，输出为角加速度  $\dot{\omega}$ 。

### 3.2 仿真结果

通过神经网络给出的预测角加速度结合 Euler 法可以给出预测的动力学演化过程，见图3.2.1。这里设置初始角度  $\theta_0 = 0.1rad$ ，初始角速度  $\omega_0 = 0.0rad/s$ ，控制力矩  $\tau = 0N \cdot m$ 。■为数值解，■为 MLP 的预测值。



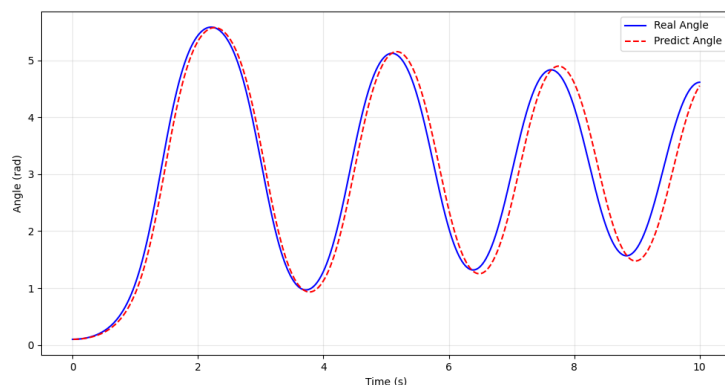


图 3.2.1: 齐次方程的动力学过程

初始角度  $\theta_0 = 0.0rad$ , 初始角速度  $\omega_0 = 0.0rad/s$ , 控制力矩  $\tau = 1N \cdot m$ 。

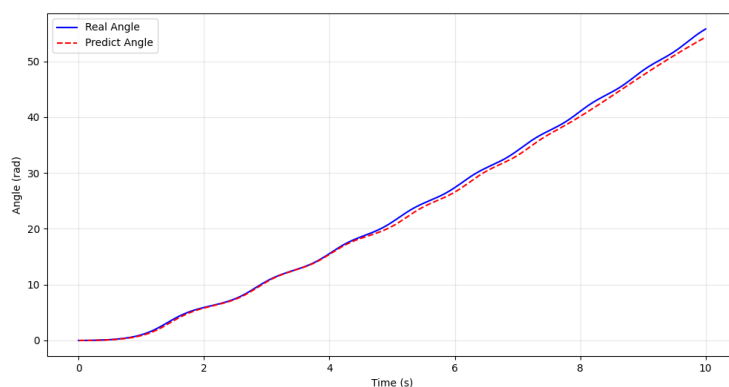


图 3.2.2: 非齐次方程的动力学过程

### 3.3 PID 校正

若使用3.1的示例程序所训练出来的神经网络并不是普遍 cover 的。换句话说, 这个模型的泛化能力不是很好。如果要改善性能需要加不同初始条件下以及不同控制力矩的数据集。下面将采用我已经训练好的一套神经网络来进行 PID 校正。注意, 图3.3.1是在神经网络上得到的结果。在进行完 PID 校正后, 我们仍需要仿真/物理实验去验证模型和参数的有效性。

---

Algorithm 1 PID 控制器伪码

---

Require:  $K_p, K_i, K_d$  (PID 增益参数)

Require: *setpoint* (目标值), *current* (当前值), *dt* (时间步长)

```

1:  $prev\_error \leftarrow 0$  {初始化前一时刻误差}
2:  $integral \leftarrow 0$  {初始化积分项}
3: while 系统中运行 do
4:    $error \leftarrow setpoint - current$  {计算当前误差}
5:    $integral \leftarrow integral + error \cdot dt$  {更新积分项}
6:    $derivative \leftarrow (error - prev\_error)/dt$  {计算误差的变化率}
7:    $output \leftarrow K_p \cdot error + K_i \cdot integral + K_d \cdot derivative$  {计算控制输出}
8:    $prev\_error \leftarrow error$  {更新前一时刻误差}
9:   应用  $output$  到系统
10: end while

```

---

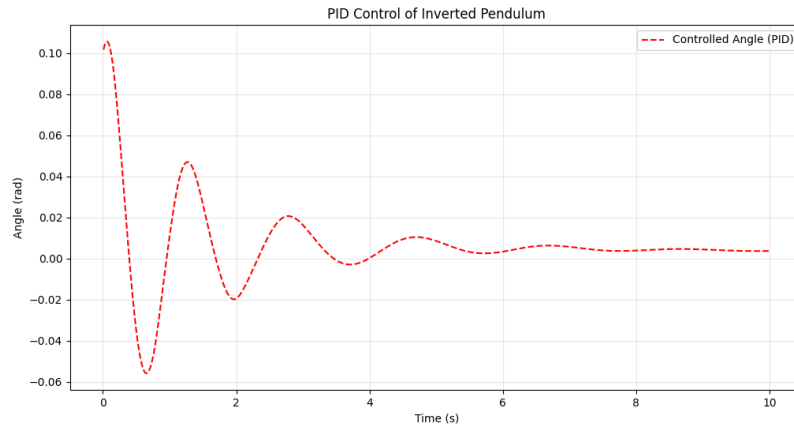


图 3.3.1: 未优化前参数给出的 PID 校正

PID 控制器通过实时调整控制力矩，校正倒立摆的偏差，使其保持直立稳定。比例控制提供快速响应，积分控制消除稳态误差，微分控制抑制振荡。对于参数  $k_P, k_I, k_D$  的确定有多种方法，图3.3.1是基于图3.2.1做的 PID 校正，设置的参数为  $k_P = 50, k_I = 2, k_D = 2$ 。当然不是最优的，甚至不是局部最优的。现使用 GA 算法进行参数的优化，以下展示了在终端上运行 GA 算法的迭代过程。

```

Terminal Win
Generation 1: Best Fitness = 44.26614012560816, Best PID = [13.38137479
9.9421895 5.99600607]
...
Generation 30: Best Fitness = 7.4091762529322365, Best PID =
[36.97944279 3.61454539 8.16905168]
...
Generation 36: Best Fitness = 6.98466953068312, Best PID = [42.44913587
3.85688028 7.83797342]
...
Optimal PID Parameters: Kp = 52.07405094371485, Ki = 6.0892017239746385,
Kd = 9.146399457554605

```

选定 PID 参数为  $k_P = 52.074, k_I = 6.0893, k_D = 9.1464$ 。

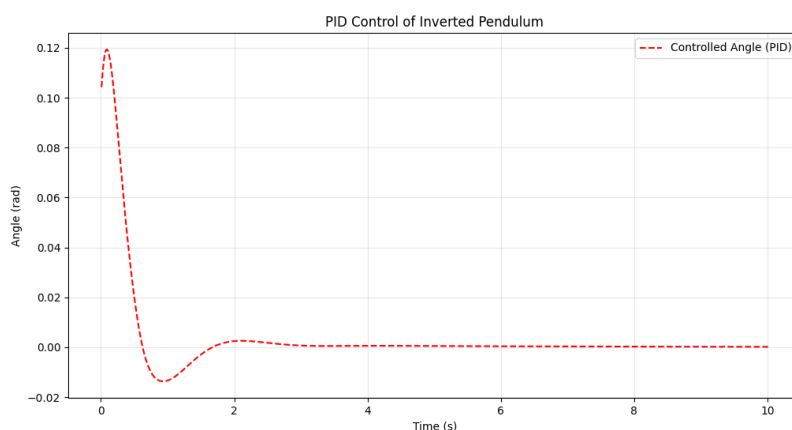


图 3.3.2: GA 算法给出的 PID 参数校正

这里的图3.3.2也是初始角度为 0.1 rad 的 PID 控制。显然控制效果优于

图3.3.1的结果。

现考虑将这套 PID 参数应用到真实倒立摆系统上，并分析控制效果。

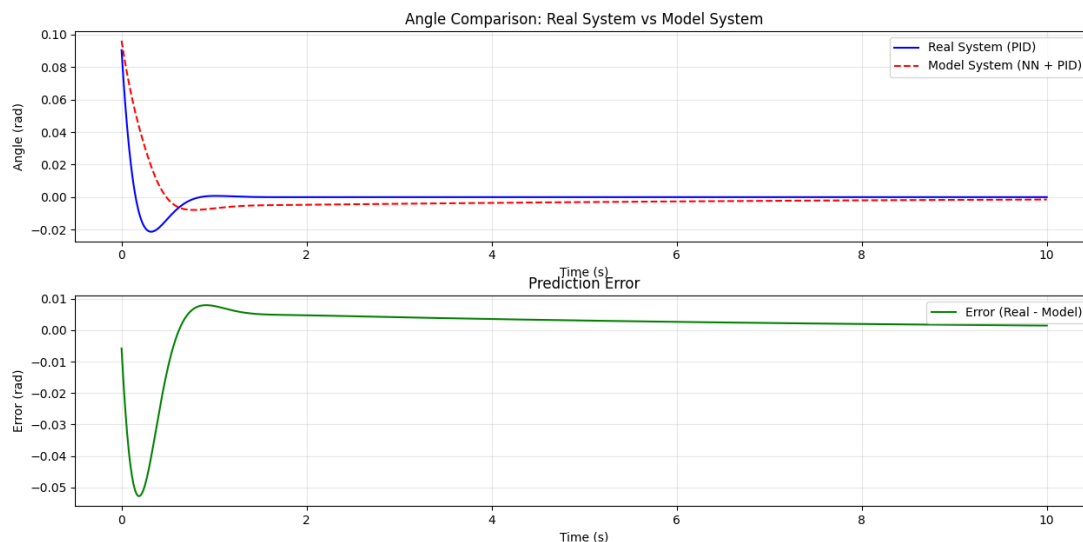


图 3.3.3: 比较及模型误差

使用平均绝对误差作为评价函数:

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_{\text{real},i} - y_{\text{model},i}|$$

Mean Absolute Error (MAE): 0.0049

从图3.3.3中会发现一个很有意思的事情，实际模型给出的预测值和真实值之间还是有着些许差异的。如果要求训练出更接近真实系统的神经网络模型，可以通过以下几点实现 (不限于):

- 增加训练数据的多样性
- 增加训练数据的分辨率
- 增加隐藏层或神经元数量

# 4 总结

本文研究了基于神经网络的一阶倒立摆系统辨识方法，并结合 PID 控制器对系统进行了校正。通过仿真实验验证了神经网络在系统辨识中的有效性，以及 PID 控制器在系统校正中的作用。未来可以进一步研究更复杂的非线性系统辨识方法，并结合强化学习优化控制策略。

整个项目已开源到个人[GitHub](#)上。核心文件的作用如下：

📁 AI-Homework	
— 36-AI.tex	LaTeX 文件
— 36-AI.pdf	我的大作业 (本文档)
— MLP-Design.py	训练神经网络的程序
— test.py	用于测试比较
— GA.py	遗传算法的程序
— Demo.py	添加 PID 控制的程序
— simulation_animation.py	倒立摆的仿真程序
— mlp_model.joblib	神经网络模型 (必要)
— README.md	

# 参考文献

周志华, 2016. 机器学习[M]. 第 1 版. 北京: 清华大学出版社. 3

杨忠, 杨荣根, 2021. 人工智能及其应用[M]. 第 2 版. 西安: 电子科技大学出版社.