### 15. 模拟退火（SA）+ 神经网络（NN）组合模型案例题目

**题目：智慧农业温室环境调控优化问题**

**・问题背景：**某现代化温室基地种植高端果蔬，当前依赖传统 PID 控制或人工经验设定环境参数，存在能耗过高（占生产成本 45%）、调控滞后（响应延迟超 30 分钟）、品质波动（差异率超 15%）及多目标冲突等问题。基地已部署传感器网络和智能执行设备，具备精准调控基础。

**・问题描述：**需优化温室环境调控参数，目标包括：① 最大化作物周生长量（≥50g / 株）；② 最大化果实品质评分（≥85 分）；③ 最小化单位产量能耗（≤8kWh/kg）。需遵循温度、湿度、CO₂浓度等约束，每 30 分钟调整一次参数。

**・数据情况：**提供近 2 年历史数据，包括每 5 分钟采集的温室内外环境数据、每周的作物生长指标、收获期的品质数据，以及分设备记录的能耗数据，数据含一定噪声和缺失值。

### 15. 模拟退火（SA）+ 神经网络（NN）求解智慧农业温室环境调控优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  import random  import copy  from datetime import datetime, timedelta  from sklearn.model\_selection import train\_test\_split  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_squared\_error, r2\_score  import torch  import torch.nn as nn  import torch.optim as optim  from torch.utils.data import Dataset, DataLoader  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  torch.manual\_seed(42)  # 1. 数据生成与预处理  def generate\_greenhouse\_data(n\_days=730, n\_samples\_day=288):  """  生成温室环境调控问题数据（模拟2年数据）  n\_days: 天数  n\_samples\_day: 每天的采样次数（5分钟一次，288次/天）  """  # 时间序列  time\_stamps = []  current\_time = datetime(2022, 1, 1, 0, 0)  for \_ in range(n\_days \* n\_samples\_day):  time\_stamps.append(current\_time)  current\_time += timedelta(minutes=5)    # 基础数据框架  data = pd.DataFrame({  '时间': time\_stamps,  '季节': [t.month % 12 for t in time\_stamps], # 1-12月  '小时': [t.hour for t in time\_stamps]  })    # 1.1 外部环境数据  # 室外温度（受季节和时间影响）  base\_temp = 15 + 10 \* np.sin(data['季节'] / 12 \* 2 \* np.pi)  hourly\_temp = 5 \* np.sin((data['小时'] - 12) / 24 \* 2 \* np.pi)  data['室外温度(℃)'] = base\_temp + hourly\_temp + np.random.normal(0, 1.5, len(data))  data['室外温度(℃)'] = np.clip(data['室外温度(℃)'], -5, 35)    # 室外光照（白天有光照）  daylight = np.where((data['小时'] >= 6) & (data['小时'] <= 18), 1, 0)  data['室外光照(lux)'] = daylight \* (50000 + 30000 \* np.sin((data['小时'] - 12) / 12 \* np.pi))  data['室外光照(lux)'] += np.random.normal(0, 5000, len(data))  data['室外光照(lux)'] = np.clip(data['室外光照(lux)'], 0, 100000)    # 1.2 温室调控参数（历史设定值）  # 日间温度设定（20-28℃）  day\_temp = 24 + 2 \* np.sin((data['小时'] - 12) / 12 \* np.pi)  # 夜间温度设定（15-18℃）  night\_temp = 16 + np.sin((data['小时'] - 22) / 12 \* np.pi)  # 昼夜区分  is\_day = np.where((data['小时'] >= 6) & (data['小时'] <= 18), 1, 0)  data['温度设定(℃)'] = is\_day \* day\_temp + (1 - is\_day) \* night\_temp  data['温度设定(℃)'] += np.random.normal(0, 0.5, len(data))  data['温度设定(℃)'] = np.clip(data['温度设定(℃)'], 15, 28)    # 湿度设定（60%-80%）  data['湿度设定(%)'] = 70 + 5 \* np.sin((data['小时'] - 8) / 16 \* np.pi)  data['湿度设定(%)'] += np.random.normal(0, 2, len(data))  data['湿度设定(%)'] = np.clip(data['湿度设定(%)'], 60, 80)    # CO₂浓度设定（400-1200ppm）  data['CO2设定(ppm)'] = 800 + 300 \* is\_day  data['CO2设定(ppm)'] += np.random.normal(0, 50, len(data))  data['CO2设定(ppm)'] = np.clip(data['CO2设定(ppm)'], 400, 1200)    # 补光时长（小时/天）  light\_needed = np.where(data['室外光照(lux)'] < 30000, 1, 0)  data['补光设定(小时)'] = light\_needed \* (3 + 2 \* np.sin((data['小时'] - 10) / 8 \* np.pi))  data['补光设定(小时)'] = np.clip(data['补光设定(小时)'], 0, 8)    # 1.3 实际环境数据（受设定值和外部环境影响）  data['室内温度(℃)'] = data['温度设定(℃)'] + 0.3 \* (data['室外温度(℃)'] - data['温度设定(℃)'])  data['室内温度(℃)'] += np.random.normal(0, 0.3, len(data))    data['室内湿度(%)'] = data['湿度设定(%)'] + 0.2 \* np.random.normal(0, 5, len(data))  data['室内湿度(%)'] = np.clip(data['室内湿度(%)'], 50, 90)    data['室内CO2(ppm)'] = data['CO2设定(ppm)'] + np.random.normal(0, 30, len(data))  data['室内CO2(ppm)'] = np.clip(data['室内CO2(ppm)'], 350, 1300)    # 1.4 作物生长与品质数据（每周记录，这里按比例扩展）  # 生长阶段（0-2：苗期、花期、结果期）  growth\_phase = np.floor((np.arange(len(data)) / (n\_samples\_day \* 30)) % 3).astype(int)  data['生长阶段'] = growth\_phase    # 周生长量（每30分钟的累积值）  temp\_factor = np.exp(-0.1 \* np.abs(data['室内温度(℃)'] - 25))  co2\_factor = data['室内CO2(ppm)'] / 1000  light\_factor = np.minimum(data['室外光照(lux)'] / 50000 + data['补光设定(小时)'] / 8, 1)  data['生长量(g/株)'] = 0.001 \* temp\_factor \* co2\_factor \* light\_factor  data['生长量(g/株)'] += np.random.normal(0, 0.0002, len(data))    # 品质评分（综合值）  data['品质评分'] = 70 + 15 \* temp\_factor + 5 \* co2\_factor + np.random.normal(0, 2, len(data))  data['品质评分'] = np.clip(data['品质评分'], 50, 100)    # 1.5 能耗数据（kWh）  # 加温能耗（温度低于设定值时）  heating = np.where(data['室外温度(℃)'] < data['温度设定(℃)'],  (data['温度设定(℃)'] - data['室外温度(℃)']) \* 0.5, 0)  # 降温能耗（温度高于设定值时）  cooling = np.where(data['室外温度(℃)'] > data['温度设定(℃)'],  (data['室外温度(℃)'] - data['温度设定(℃)']) \* 0.3, 0)  # 补光能耗  lighting = data['补光设定(小时)'] / 24 \* 1.2  # CO2生成能耗  co2\_energy = (data['CO2设定(ppm)'] - 400) / 800 \* 0.8    data['能耗(kWh)'] = heating + cooling + lighting + co2\_energy + np.random.normal(0, 0.1, len(data))  data['能耗(kWh)'] = np.clip(data['能耗(kWh)'], 0, 5)    # 按30分钟聚合数据（调控周期）  data['调控周期'] = (data.index // 6).astype(int) # 每6个5分钟为30分钟  agg\_data = data.groupby('调控周期').agg({  '时间': 'first',  '季节': 'first',  '室外温度(℃)': 'mean',  '室外光照(lux)': 'mean',  '温度设定(℃)': 'mean',  '湿度设定(%)': 'mean',  'CO2设定(ppm)': 'mean',  '补光设定(小时)': 'sum', # 累计补光时间  '生长阶段': 'first',  '生长量(g/株)': 'sum', # 累计生长量  '品质评分': 'mean',  '能耗(kWh)': 'sum' # 累计能耗  }).reset\_index(drop=True)    # 计算单位产量能耗（每克生长量的能耗）  agg\_data['单位产量能耗(kWh/g)'] = agg\_data['能耗(kWh)'] / (agg\_data['生长量(g/株)'] + 0.001)    return agg\_data  # 2. 神经网络模型（预测生长、品质和能耗）  class GreenhouseNN(nn.Module):  def \_\_init\_\_(self, input\_size=8, hidden\_sizes=[64, 32, 16], output\_size=3):  """  温室环境预测神经网络  input\_size: 输入特征数量  hidden\_sizes: 隐藏层大小列表  output\_size: 输出特征数量（生长量、品质、能耗）  """  super(GreenhouseNN, self).\_\_init\_\_()    # 构建网络层  layers = []  prev\_size = input\_size  for size in hidden\_sizes:  layers.append(nn.Linear(prev\_size, size))  layers.append(nn.ReLU())  prev\_size = size  layers.append(nn.Linear(prev\_size, output\_size))    self.model = nn.Sequential(\*layers)    # 初始化权重  self.\_initialize\_weights()    def \_initialize\_weights(self):  """初始化网络权重"""  for m in self.model.modules():  if isinstance(m, nn.Linear):  nn.init.kaiming\_normal\_(m.weight, mode='fan\_in', nonlinearity='relu')  if m.bias is not None:  nn.init.constant\_(m.bias, 0)    def forward(self, x):  """前向传播"""  return self.model(x)  # 数据加载器  class GreenhouseDataset(Dataset):  def \_\_init\_\_(self, data, scaler\_X=None, scaler\_y=None, is\_train=True):  """  温室数据数据集  data: 原始数据  scaler\_X: 输入特征标准化器  scaler\_y: 输出特征标准化器  is\_train: 是否为训练集（决定是否拟合标准化器）  """  # 选择特征  self.features = [  '季节', '室外温度(℃)', '室外光照(lux)',  '温度设定(℃)', '湿度设定(%)', 'CO2设定(ppm)',  '补光设定(小时)', '生长阶段'  ]  self.targets = [  '生长量(g/株)', '品质评分', '能耗(kWh)'  ]    # 提取特征和目标  X = data[self.features].values  y = data[self.targets].values    # 标准化  if is\_train:  self.scaler\_X = MinMaxScaler()  self.scaler\_y = MinMaxScaler()  self.X = self.scaler\_X.fit\_transform(X)  self.y = self.scaler\_y.fit\_transform(y)  else:  self.scaler\_X = scaler\_X  self.scaler\_y = scaler\_y  self.X = self.scaler\_X.transform(X)  self.y = self.scaler\_y.transform(y)    # 转换为张量  self.X = torch.FloatTensor(self.X)  self.y = torch.FloatTensor(self.y)    def \_\_len\_\_(self):  return len(self.X)    def \_\_getitem\_\_(self, idx):  return self.X[idx], self.y[idx]  # 训练神经网络  def train\_nn\_model(data, epochs=100, batch\_size=64, lr=0.001):  """训练神经网络模型"""  # 划分训练集和验证集  train\_data, val\_data = train\_test\_split(data, test\_size=0.2, random\_state=42)    # 创建数据集  train\_dataset = GreenhouseDataset(train\_data, is\_train=True)  val\_dataset = GreenhouseDataset(val\_data,  scaler\_X=train\_dataset.scaler\_X,  scaler\_y=train\_dataset.scaler\_y,  is\_train=False)    # 创建数据加载器  train\_loader = DataLoader(train\_dataset, batch\_size=batch\_size, shuffle=True)  val\_loader = DataLoader(val\_dataset, batch\_size=batch\_size, shuffle=False)    # 初始化模型、损失函数和优化器  model = GreenhouseNN()  criterion = nn.MSELoss()  optimizer = optim.Adam(model.parameters(), lr=lr)    # 记录训练过程  train\_losses = []  val\_losses = []    # 训练模型  for epoch in range(epochs):  model.train()  train\_loss = 0.0    for X\_batch, y\_batch in train\_loader:  # 前向传播  outputs = model(X\_batch)  loss = criterion(outputs, y\_batch)    # 反向传播和优化  optimizer.zero\_grad()  loss.backward()  optimizer.step()    train\_loss += loss.item() \* X\_batch.size(0)    # 计算平均训练损失  train\_loss /= len(train\_loader.dataset)  train\_losses.append(train\_loss)    # 在验证集上评估  model.eval()  val\_loss = 0.0  with torch.no\_grad():  for X\_batch, y\_batch in val\_loader:  outputs = model(X\_batch)  loss = criterion(outputs, y\_batch)  val\_loss += loss.item() \* X\_batch.size(0)    val\_loss /= len(val\_loader.dataset)  val\_losses.append(val\_loss)    # 每10个epoch打印一次  if (epoch + 1) % 10 == 0:  print(f'Epoch [{epoch+1}/{epochs}], 训练损失: {train\_loss:.6f}, 验证损失: {val\_loss:.6f}')    # 绘制损失曲线  plt.figure(figsize=(10, 6))  plt.plot(train\_losses, label='训练损失')  plt.plot(val\_losses, label='验证损失')  plt.title('模型训练损失曲线')  plt.xlabel('Epoch')  plt.ylabel('MSE损失')  plt.legend()  plt.grid(True)  plt.show()    return model, train\_dataset.scaler\_X, train\_dataset.scaler\_y, train\_dataset.features  # 3. 模拟退火算法（优化调控参数）  class SimulatedAnnealing:  def \_\_init\_\_(self, model, scaler\_X, scaler\_y, features, initial\_temp=100,  cooling\_rate=0.95, max\_iter=1000, final\_temp=1e-3):  """  模拟退火算法优化温室调控参数  model: 训练好的神经网络模型  scaler\_X: 输入标准化器  scaler\_y: 输出标准化器  features: 特征名称列表  initial\_temp: 初始温度  cooling\_rate: 降温速率  max\_iter: 最大迭代次数  final\_temp: 终止温度  """  self.model = model  self.scaler\_X = scaler\_X  self.scaler\_y = scaler\_y  self.features = features    # SA参数  self.T = initial\_temp  self.cooling\_rate = cooling\_rate  self.max\_iter = max\_iter  self.final\_temp = final\_temp    # 调控参数的约束范围  self.param\_ranges = {  '温度设定(℃)': (15, 28),  '湿度设定(%)': (60, 80),  'CO2设定(ppm)': (400, 1200),  '补光设定(小时)': (0</doubaocanvas> |