### 12. 模糊神经网络（FNN）+ 粒子群优化（PSO）组合模型案例题目

**题目：城市污水处理厂曝气参数优化问题**

* **问题背景**：某污水处理厂采用活性污泥法处理城市污水，曝气环节的溶解氧浓度（DO）、曝气时长直接影响 COD 去除率和能耗（曝气能耗占总能耗的 60%）。当前采用固定参数（DO=2mg/L，曝气 4 小时），COD 去除率仅 80%，且冬季因水温低效率下降更明显。
* **问题描述**：需根据进水水质（含模糊特征如 “COD 较高”“污泥浓度偏稀”）动态调整曝气参数，目标包括：① 最大化 COD 去除率（≥90%）；② 最小化曝气能耗；③ 避免 DO 浓度剧烈波动（≤±0.5mg/L）。
* **数据情况**：提供连续 6 个月的运行数据，包括：进水 COD 浓度（mg/L）、污泥浓度（g/L）、水温（℃）、当前曝气参数、COD 去除率、小时能耗（kWh）。数据中含传感器噪声（如 DO 测量误差 ±0.2mg/L）。

### 12. 模糊神经网络（FNN）+ 粒子群优化（PSO）求解城市污水处理厂曝气参数优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  import random  import copy  from sklearn.preprocessing import MinMaxScaler  from sklearn.model\_selection import train\_test\_split  from sklearn.metrics import mean\_squared\_error, r2\_score  import matplotlib.dates as mdates  from matplotlib.animation import FuncAnimation  # 设置随机种子，保证结果可复现  np.random.seed(42)  random.seed(42)  # 1. 数据生成与预处理  def generate\_sewage\_data(n\_samples=180, noise\_level=0.05):  """  生成污水处理厂运行数据（模拟6个月数据）  n\_samples: 样本数量（天）  noise\_level: 数据噪声水平  """  # 时间序列  dates = pd.date\_range(start='2023-01-01', periods=n\_samples, freq='D')    # 环境与进水参数  data = pd.DataFrame()  data['date'] = dates    # 水温（随季节变化，1-12月）  month = dates.month  data['water\_temp'] = 15 + 10 \* np.sin(month / 12 \* 2 \* np.pi) + np.random.normal(0, 1, n\_samples)  data['water\_temp'] = np.clip(data['water\_temp'], 5, 28) # 限制在5-28℃    # 进水COD浓度（100-400 mg/L）  base\_cod = 250 + 100 \* np.sin(np.linspace(0, 4\*np.pi, n\_samples))  data['influent\_cod'] = base\_cod + np.random.normal(0, 30, n\_samples)  data['influent\_cod'] = np.clip(data['influent\_cod'], 100, 400)    # 污泥浓度（2-6 g/L）  data['sludge\_conc'] = 4 + 1.5 \* np.sin(np.linspace(0, 2\*np.pi, n\_samples) + np.pi) + np.random.normal(0, 0.3, n\_samples)  data['sludge\_conc'] = np.clip(data['sludge\_conc'], 2, 6)    # 传统曝气参数（固定值）  data['do\_setpoint'] = 2.0 # 溶解氧设定值（mg/L）  data['aeration\_time'] = 4.0 # 曝气时长（小时）    # 计算COD去除率（受多种因素影响）  # 温度影响：温度升高，反应加快  temp\_factor = 0.8 + 0.2 \* (data['water\_temp'] - 5) / 23 # 归一化到0.8-1.0    # 污泥浓度影响：适中浓度最佳  sludge\_factor = 1 - 0.3 \* abs(data['sludge\_conc'] - 4) / 2 # 4g/L时最佳    # 进水COD影响：浓度过高抑制去除率  cod\_factor = 1 - 0.2 \* (data['influent\_cod'] - 250) / 150 # 250mg/L时最佳  cod\_factor = np.clip(cod\_factor, 0.6, 1.0)    # 基础去除率（传统参数下）  base\_removal = 0.8 + 0.05 \* (temp\_factor + sludge\_factor + cod\_factor - 2.4)  data['cod\_removal'] = base\_removal + np.random.normal(0, noise\_level, n\_samples)  data['cod\_removal'] = np.clip(data['cod\_removal'], 0.6, 0.9) # 限制在60%-90%    # 计算能耗（kWh）  # 曝气能耗与曝气时间、DO设定值正相关  energy\_base = 150 + data['aeration\_time'] \* 20 + data['do\_setpoint'] \* 30  # 低温时需更高能耗  temp\_energy\_factor = 1 + (15 - data['water\_temp']) / 30  data['energy\_consumption'] = energy\_base \* temp\_energy\_factor + np.random.normal(0, 10, n\_samples)  data['energy\_consumption'] = np.clip(data['energy\_consumption'], 100, 400)    # DO波动（mg/L）  data['do\_fluctuation'] = 0.3 + 0.2 \* np.random.random(n\_samples) # 传统控制下的波动    return data  # 2. 模糊逻辑系统设计  class FuzzySystem:  def \_\_init\_\_(self):  """初始化模糊逻辑系统，定义模糊集和规则"""  # 输入变量：进水COD、污泥浓度、水温  # 输出变量：DO设定值调整量、曝气时间调整量    # 2.1 定义模糊集  # 进水COD模糊集：低(L)、中(M)、高(H)  self.cod\_mf = {  'L': {'center': 150, 'width': 80}, # 梯形隶属度函数参数  'M': {'center': 250, 'width': 100},  'H': {'center': 350, 'width': 80}  }    # 污泥浓度模糊集：稀(L)、适中(M)、浓(H)  self.sludge\_mf = {  'L': {'center': 2.5, 'width': 1.0},  'M': {'center': 4.0, 'width': 1.0},  'H': {'center': 5.5, 'width': 1.0}  }    # 水温模糊集：低(L)、中(M)、高(H)  self.temp\_mf = {  'L': {'center': 10, 'width': 5},  'M': {'center': 18, 'width': 6},  'H': {'center': 25, 'width': 5}  }    # 输出：DO调整量模糊集：负大(NB)、负小(NS)、零(Z)、正小(PS)、正大(PB)  self.do\_delta\_mf = {  'NB': {'center': -0.8, 'width': 0.5},  'NS': {'center': -0.3, 'width': 0.4},  'Z': {'center': 0, 'width': 0.3},  'PS': {'center': 0.3, 'width': 0.4},  'PB': {'center': 0.8, 'width': 0.5}  }    # 输出：曝气时间调整量模糊集：负大(NB)、负小(NS)、零(Z)、正小(PS)、正大(PB)  self.time\_delta\_mf = {  'NB': {'center': -1.5, 'width': 0.8},  'NS': {'center': -0.5, 'width': 0.6},  'Z': {'center': 0, 'width': 0.5},  'PS': {'center': 0.5, 'width': 0.6},  'PB': {'center': 1.5, 'width': 0.8}  }    # 2.2 定义模糊规则（共3×3×3=27条规则）  self.rules = []    # 规则生成：根据专家知识  # 规则格式：(cod, sludge, temp) → (do\_delta, time\_delta)  cod\_levels = ['L', 'M', 'H']  sludge\_levels = ['L', 'M', 'H']  temp\_levels = ['L', 'M', 'H']    for c in cod\_levels:  for s in sludge\_levels:  for t in temp\_levels:  # 确定DO调整量  if c == 'H': # COD高需要更高DO  do = 'PB' if t == 'L' else 'PS'  elif c == 'L': # COD低可以降低DO  do = 'NB' if t == 'H' else 'NS'  else: # COD中等  if s == 'H': # 污泥浓，需更高DO  do = 'PS'  elif s == 'L': # 污泥稀，需较低DO  do = 'NS'  else: # 污泥适中  do = 'Z'    # 确定曝气时间调整量  if c == 'H': # COD高需要更长曝气时间  time = 'PB' if t == 'L' else 'PS'  elif c == 'L': # COD低可以缩短曝气时间  time = 'NB' if t == 'H' else 'NS'  else: # COD中等  if t == 'L': # 温度低需要更长时间  time = 'PS'  elif t == 'H': # 温度高可以缩短时间  time = 'NS'  else: # 温度适中  time = 'Z'    self.rules.append( ((c, s, t), (do, time)) )    def membership(self, x, center, width):  """梯形隶属度函数"""  if x < center - width:  return 0.0  elif center - width <= x < center:  return (x - (center - width)) / width  elif center <= x < center + width:  return (center + width - x) / width  else:  return 0.0    def fuzzify(self, cod, sludge, temp):  """模糊化输入变量"""  # 计算各输入的隶属度  cod\_degree = {k: self.membership(cod, v['center'], v['width'])  for k, v in self.cod\_mf.items()}  sludge\_degree = {k: self.membership(sludge, v['center'], v['width'])  for k, v in self.sludge\_mf.items()}  temp\_degree = {k: self.membership(temp, v['center'], v['width'])  for k, v in self.temp\_mf.items()}    return cod\_degree, sludge\_degree, temp\_degree    def infer(self, cod\_degree, sludge\_degree, temp\_degree):  """模糊推理"""  do\_weights = {k: 0.0 for k in self.do\_delta\_mf.keys()}  time\_weights = {k: 0.0 for k in self.time\_delta\_mf.keys()}    total\_weight = 0.0    for (cond, res), rule in enumerate(self.rules):  (c, s, t), (do, time) = rule    # 规则触发强度（取最小值）  strength = min(cod\_degree[c], sludge\_degree[s], temp\_degree[t])    if strength > 0:  # 累积权重  do\_weights[do] += strength  time\_weights[time] += strength  total\_weight += strength    return do\_weights, time\_weights, total\_weight    def defuzzify(self, weights, mf\_params, total\_weight):  """去模糊化（重心法）"""  if total\_weight == 0:  return 0.0    weighted\_sum = 0.0  for k, v in weights.items():  if v > 0:  weighted\_sum += v \* mf\_params[k]['center']    return weighted\_sum / total\_weight    def predict(self, cod, sludge, temp):  """完整模糊推理过程"""  cod\_deg, sludge\_deg, temp\_deg = self.fuzzify(cod, sludge, temp)  do\_weights, time\_weights, total\_w = self.infer(cod\_deg, sludge\_deg, temp\_deg)    do\_delta = self.defuzzify(do\_weights, self.do\_delta\_mf, total\_w)  time\_delta = self.defuzzify(time\_weights, self.time\_delta\_mf, total\_w)    # 限制调整范围  do\_delta = np.clip(do\_delta, -1.0, 1.0)  time\_delta = np.clip(time\_delta, -2.0, 2.0)    return do\_delta, time\_delta  # 3. 模糊神经网络（FNN）模型  class FuzzyNeuralNetwork:  def \_\_init\_\_(self, fuzzy\_system, learning\_rate=0.01):  """  模糊神经网络  将模糊系统的隶属度函数参数作为可训练的权重  """  self.fuzzy = fuzzy\_system  self.lr = learning\_rate    # 初始化可训练参数（隶属度函数的中心和宽度）  # 使用模糊系统的初始参数  self.params = {  'cod': copy.deepcopy(fuzzy\_system.cod\_mf),  'sludge': copy.deepcopy(fuzzy\_system.sludge\_mf),  'temp': copy.deepcopy(fuzzy\_system.temp\_mf),  'do\_delta': copy.deepcopy(fuzzy\_system.do\_delta\_mf),  'time\_delta': copy.deepcopy(fuzzy\_system.time\_delta\_mf)  }    def membership(self, x, center, width):  """梯形隶属度函数（与模糊系统相同）"""  if x < center - width:  return 0.0  elif center - width <= x < center:  return (x - (center - width)) / width  elif center <= x < center + width:  return (center + width - x) / width  else:  return 0.0    def forward(self, cod, sludge, temp):  """前向传播：计算输出和中间变量"""  # 模糊化（使用当前参数）  cod\_degree = {k: self.membership(cod, v['center'], v['width'])  for k, v in self.params['cod'].items()}  sludge\_degree = {k: self.membership(sludge, v['center'], v['width'])  for k, v in self.params['sludge'].items()}  temp\_degree = {k: self.membership(temp, v['center'], v['width'])  for k, v in self.params['temp'].items()}    # 推理  do\_weights = {k: 0.0 for k in self.params['do\_delta'].keys()}  time\_weights = {k: 0.0 for k in self.params['time\_delta'].keys()}  total\_weight = 0.0    for (c, s, t), (do, time) in self.fuzzy.rules:  strength = min(cod\_degree[c], sludge\_degree[s], temp\_degree[t])    if strength > 0:  do\_weights[do] += strength  time\_weights[time] += strength  total\_weight += strength    # 去模糊化  do\_delta = 0.0  time\_delta = 0.0    if total\_weight > 0:  do\_delta = sum(w \* self.params['do\_delta'][k]['center']  for k, w in do\_weights.items()) / total\_weight  time\_delta = sum(w \* self.params['time\_delta'][k]['center']  for k, w in time\_weights.items()) / total\_weight    # 限制范围  do\_delta = np.clip(do\_delta, -1.0, 1.0)  time\_delta = np.clip(time\_delta, -2.0, 2.0)    # 保存中间结果用于反向传播  self.cache = {  'cod\_degree': cod\_degree,  'sludge\_degree': sludge\_degree,  'temp\_degree': temp\_degree,  'do\_weights': do\_weights,  'time\_weights': time\_weights,  'total\_weight': total\_weight  }    return do\_delta, time\_delta    def backward(self, cod, sludge, temp, do\_delta\_pred, time\_delta\_pred, do\_delta\_true, time\_delta\_true):  """反向传播：计算梯度并更新参数"""  # 计算误差  do\_error = do\_delta\_pred - do\_delta\_true  time\_error = time\_delta\_pred - time\_delta\_true    # 获取缓存的中间结果  cache = self.cache  if cache['total\_weight'] == 0:  return # 无有效推理，不更新    # 1. 更新输出隶属度函数的中心  # DO调整量  for k in self.params['do\_delta'].keys():  grad = (cache['do\_weights'][k] / cache['total\_weight']) \* do\_error  self.params['do\_delta'][k]['center'] -= self.lr \* grad    # 曝气时间调整量  for k in self.params['time\_delta'].keys():  grad = (cache['time\_weights'][k] / cache['total\_weight']) \* time\_error  self.params['time\_delta'][k]['center'] -= self.lr \* grad</doubaocanvas> |