### 6. 深度学习（DNN）+ 梯度下降（GD）组合模型案例题目

**题目：区域电网短期负荷预测与发电调度优化问题**

* **问题背景**：某省级电网覆盖 10 个地级市，夏季用电高峰时需平衡火电机组、风电场、光伏电站的出力，避免弃风弃光（当前弃风率约 8%）或用电缺口。传统负荷预测误差达 10%，导致调度方案频繁调整，增加煤耗成本约 500 万元 / 月。
* **问题描述**：需先预测未来 24 小时的区域总用电负荷，再优化各电源的出力计划，目标包括：① 最小化预测误差（≤5%）；② 最小化发电总成本（火电煤耗 + 新能源补贴）；③ 确保实时出力与负荷的偏差≤2%。需考虑温度、节假日、工业停产等多因素对负荷的影响。
* **数据情况**：提供近 5 年的历史数据，包括：每小时的总用电负荷（MW）、平均温度（℃）、湿度（%）、风速（m/s）、日照时长（h）、节假日标识、工业用电量占比，以及各电源的单位发电成本、最大出力限制。

### 6. 深度学习（DNN）+ 梯度下降（GD）求解区域电网短期负荷预测与发电调度优化代码

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  from sklearn.preprocessing import MinMaxScaler  from sklearn.metrics import mean\_absolute\_error, mean\_absolute\_percentage\_error  import tensorflow as tf  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import Dense, Dropout, BatchNormalization  from tensorflow.keras.optimizers import Adam  # 设置随机种子，保证结果可复现  np.random.seed(42)  tf.random.set\_seed(42)  # 数据生成与预处理  def generate\_power\_data(n\_years=5):  """生成包含多种影响因素的电网负荷数据"""  n\_days = n\_years \* 365  n\_hours = 24  n\_samples = n\_days \* n\_hours    # 创建时间特征  data = pd.DataFrame()  data['hour'] = np.tile(np.arange(n\_hours), n\_days) # 小时（0-23）  data['weekday'] = np.repeat(np.arange(7), n\_hours)[:n\_samples] # 星期（0-6）  data['month'] = np.repeat(np.tile(np.arange(1,13), [30\*n\_hours]), n\_days//12 +1)[:n\_samples] # 月份（1-12）    # 生成气象特征  data['temp'] = 15 + 12\*np.sin(np.linspace(0, 2\*np.pi\*n\_years, n\_samples)) + np.random.normal(0, 3, n\_samples) # 温度（℃）  data['humidity'] = 60 + 20\*np.sin(np.linspace(0, 2\*np.pi\*n\_years, n\_samples) + 1) + np.random.normal(0, 5, n\_samples) # 湿度（%）  data['wind\_speed'] = 3 + 2\*np.random.randn(n\_samples) # 风速（m/s）  data['sunshine'] = np.maximum(0, 8 + 4\*np.sin(np.linspace(0, 2\*np.pi\*n\_years, n\_samples) - 1) + np.random.normal(0, 1, n\_samples)) # 日照时长（h）    # 生成社会经济特征  data['holiday'] = np.zeros(n\_samples) # 节假日标识（0-1）  for d in range(0, n\_samples, n\_hours):  if np.random.rand() < 0.05: # 5%概率为节假日  data.loc[d:d+n\_hours-1, 'holiday'] = 1  data['industry\_ratio'] = 40 + 10\*np.random.randn(n\_samples) # 工业用电占比（%）  data['industry\_ratio'] = np.clip(data['industry\_ratio'], 20, 60)    # 生成目标负荷（MW）：受多种因素影响的非线性关系  base\_load = 1000 + 500\*np.sin(np.linspace(0, 2\*np.pi\*n\_years, n\_samples) + np.pi) # 基础负荷  hour\_effect = 200\*np.sin(data['hour']/24\*2\*np.pi + np.pi) # 小时效应  temp\_effect = -10 \* (data['temp'] - 15) # 温度效应  holiday\_effect = -200 \* data['holiday'] # 节假日效应  industry\_effect = 10 \* (data['industry\_ratio'] - 40) # 工业用电效应  noise = np.random.normal(0, 50, n\_samples) # 随机噪声    data['load'] = base\_load + hour\_effect + temp\_effect + holiday\_effect + industry\_effect + noise  data['load'] = np.clip(data['load'], 500, 2000) # 负荷范围限制    return data  # 生成5年数据  data = generate\_power\_data(n\_years=5)  print(f"生成数据规模: {data.shape}")  print(f"负荷数据范围: {data['load'].min():.2f} - {data['load'].max():.2f} MW")  # 特征与目标分离  features = ['hour', 'weekday', 'month', 'temp', 'humidity', 'wind\_speed', 'sunshine', 'holiday', 'industry\_ratio']  X = data[features].values  y = data['load'].values.reshape(-1, 1)  # 数据归一化  scaler\_X = MinMaxScaler()  scaler\_y = MinMaxScaler()  X\_scaled = scaler\_X.fit\_transform(X)  y\_scaled = scaler\_y.fit\_transform(y)  # 划分训练集与测试集（前4年训练，后1年测试）  train\_size = 4\*365\*24  X\_train, X\_test = X\_scaled[:train\_size], X\_scaled[train\_size:]  y\_train, y\_test = y\_scaled[:train\_size], y\_scaled[train\_size:]  # 构建DNN模型  def build\_dnn\_model(input\_dim):  """构建用于负荷预测的深度神经网络模型"""  model = Sequential([  Dense(128, activation='relu', input\_shape=(input\_dim,)),  BatchNormalization(),  Dropout(0.3),  Dense(64, activation='relu'),  BatchNormalization(),  Dropout(0.2),  Dense(32, activation='relu'),  Dense(1) # 输出层（负荷预测值）  ])    # 使用Adam优化器（基于梯度下降改进）  optimizer = Adam(learning\_rate=0.001, beta\_1=0.9, beta\_2=0.999)  model.compile(optimizer=optimizer, loss='mse')  return model  # 初始化并训练模型  model = build\_dnn\_model(input\_dim=X\_train.shape[1])  model.summary()  # 训练模型  history = model.fit(  X\_train, y\_train,  epochs=80,  batch\_size=64,  validation\_split=0.1,  verbose=1,  callbacks=[tf.keras.callbacks.EarlyStopping(patience=10, restore\_best\_weights=True)]  )  # 模型评估  def evaluate\_model(model, X\_test, y\_test, scaler\_y):  """评估模型预测性能"""  y\_pred\_scaled = model.predict(X\_test, verbose=0)  y\_pred = scaler\_y.inverse\_transform(y\_pred\_scaled)  y\_true = scaler\_y.inverse\_transform(y\_test)    mae = mean\_absolute\_error(y\_true, y\_pred)  mape = mean\_absolute\_percentage\_error(y\_true, y\_pred) \* 100    print(f"\n模型评估结果:")  print(f"平均绝对误差（MAE）: {mae:.2f} MW")  print(f"平均绝对百分比误差（MAPE）: {mape:.2f}%")    return y\_pred, y\_true  # 评估测试集性能  y\_pred, y\_true = evaluate\_model(model, X\_test, y\_test, scaler\_y)  # 发电调度优化模型  def power\_dispatch\_optimization(load\_pred, fossil\_max=1000, wind\_max=500, solar\_max=300, storage\_cap=200):  """  基于预测负荷进行发电调度优化  目标：最小化发电成本，最大化清洁能源利用率  """  n\_hours = len(load\_pred)    # 发电成本参数（元/MW）  costs = {  'fossil': 500, # 火电机组  'wind': 300, # 风电场  'solar': 200, # 光伏电站  'storage': 100 # 储能调用  }    # 初始化各电源出力  fossil = np.zeros(n\_hours)  wind = np.zeros(n\_hours)  solar = np.zeros(n\_hours)  storage = np.zeros(n\_hours)  storage\_level = np.zeros(n\_hours + 1) # 储能电量水平    # 模拟可再生能源出力（基于历史气象数据）  solar\_potential = np.maximum(0, 0.8 \* solar\_max \* np.sin(np.linspace(0, np.pi, 12)) +  0.2 \* solar\_max \* np.sin(np.linspace(0, np.pi, 12)))[np.tile(np.arange(12), 2)[:n\_hours]]  wind\_potential = wind\_max \* (0.5 + 0.5 \* np.random.rand(n\_hours))    # 逐小时调度  for t in range(n\_hours):  # 优先利用可再生能源  solar[t] = min(solar\_potential[t], load\_pred[t])  remaining = load\_pred[t] - solar[t]    wind[t] = min(wind\_potential[t], remaining)  remaining -= wind[t]    # 利用储能（优先放电）  if remaining > 0 and storage\_level[t] > 0:  use\_storage = min(remaining, storage\_level[t])  storage[t] = -use\_storage # 负号表示放电  storage\_level[t+1] = storage\_level[t] - use\_storage  remaining -= use\_storage    # 火电机组补充  fossil[t] = min(remaining, fossil\_max)  remaining -= fossil[t]    # 若仍有缺口，增加火电机组出力（允许10%过载）  if remaining > 0:  overload = min(remaining, 0.1 \* fossil\_max)  fossil[t] += overload  remaining -= overload    # 若有剩余电量，充电至储能  if remaining < 0:  charge = min(-remaining, storage\_cap - storage\_level[t])  storage[t] = charge # 正号表示充电  storage\_level[t+1] = storage\_level[t] + charge  else:  storage\_level[t+1] = storage\_level[t]    # 计算总成本  total\_cost = np.sum(fossil \* costs['fossil'] +  wind \* costs['wind'] +  solar \* costs['solar'] +  np.abs(storage) \* costs['storage'])    # 计算清洁能源占比  clean\_energy = np.sum(wind + solar + np.maximum(0, storage))  total\_energy = np.sum(fossil + wind + solar + np.abs(storage))  clean\_ratio = (clean\_energy / total\_energy) \* 100    print(f"\n调度优化结果:")  print(f"总调度成本: {total\_cost:.2f} 元")  print(f"清洁能源占比: {clean\_ratio:.2f}%")    return {  'fossil': fossil,  'wind': wind,  'solar': solar,  'storage': storage,  'total\_cost': total\_cost,  'clean\_ratio': clean\_ratio  }  # 对测试集最后30天进行调度优化  last\_month\_idx = -30\*24  load\_pred\_last\_month = y\_pred[last\_month\_idx:]  dispatch\_result = power\_dispatch\_optimization(load\_pred\_last\_month)  # 可视化结果  def plot\_results(y\_true, y\_pred, dispatch\_result, last\_month\_idx):  """可视化预测结果和调度方案"""  plt.figure(figsize=(18, 12))    # 1. 训练损失曲线  plt.subplot(3, 1, 1)  plt.plot(history.history['loss'], label='训练损失')  plt.plot(history.history['val\_loss'], label='验证损失')  plt.title('模型训练损失曲线')  plt.xlabel('Epoch')  plt.ylabel('均方误差（MSE）')  plt.legend()    # 2. 预测与实际负荷对比（最后7天）  last\_week\_idx = -7\*24  plt.subplot(3, 1, 2)  plt.plot(y\_true[last\_week\_idx:], label='实际负荷')  plt.plot(y\_pred[last\_week\_idx:], label='预测负荷', linestyle='--')  plt.title('最后7天负荷预测对比')  plt.xlabel('小时')  plt.ylabel('负荷（MW）')  plt.legend()    # 3. 发电调度方案（最后7天）  plt.subplot(3, 1, 3)  hours = np.arange(7\*24)  last\_7d\_idx = -7\*24  plt.stackplot(hours,  dispatch\_result['fossil'][last\_7d\_idx:],  dispatch\_result['wind'][last\_7d\_idx:],  dispatch\_result['solar'][last\_7d\_idx:],  np.abs(dispatch\_result['storage'][last\_7d\_idx:]),  labels=['火电', '风电', '光伏', '储能'])  plt.plot(load\_pred\_last\_month[last\_7d\_idx:], 'k--', label='预测负荷')  plt.title('最后7天发电调度方案')  plt.xlabel('小时')  plt.ylabel('出力（MW）')  plt.legend()    plt.tight\_layout()  plt.show()  # 可视化所有结果  plot\_results(y\_true, y\_pred, dispatch\_result, last\_month\_idx) |