**RNN**

1. **环境准备**

**Python 环境**

1. Python环境

使用Anaconda创建虚拟环境，安装所需依赖。命令：

|  |
| --- |
| Bash conda create -n ecg\_rnn python=3.8 |

|  |
| --- |
| Plain Text conda activate ecg\_rnn |

1. 安装依赖库

安装TensorFlow（或Keras）、NumPy、Pandas、Matplotlib、Scikit-learn以及信号处理相关库（如SciPy、wfdb）。

|  |
| --- |
| Bash pip install tensorflow numpy pandas matplotlib scikit-learn scipy wfdb |

需要使用中科大镜像，在下载时关闭网络代理：

|  |
| --- |
| Shell pip install tensorflow numpy pandas matplotlib scikit-learn scipy wfdb -i https://mirrors.ustc.edu.cn/pypi/web/simple |

**新建项目**



2. **数据集获取与预处理**

**数据集下载**

1. 数据集选择：使用MIT-BIH Arrhythmia Database，该数据集在PhysioNet上公开。
2. 下载方法： 访问[PhysioNet MIT-BIH Arrhythmia Database](https://physionet.org/content/mitdb/1.0.0/)下载数据文件。
3. 将下载的数据文件（通常包括 .dat、.hea 文件及注释文件）放到项目目录下的一个文件夹（ ./data/mitdb/）。

**数据预处理**

创建一个 Python 脚本（preprocess.py），编写代码：

|  |
| --- |
| Python import wfdb import numpy as np import scipy.signal as signal  *# 读取 ECG 数据* def load\_record(record\_id='100'):  *"""*  *加载ECG数据记录（从 PhysioNet 数据库）*  *"""* record = wfdb.rdrecord(f'./data/mitdb/{record\_id}')  annotation = wfdb.rdann(f'./data/mitdb/{record\_id}', 'atr') *# 获取标注数据* return record, annotation  *# 带通滤波函数* def bandpass\_filter(signal\_data, lowcut=0.5, highcut=40.0, fs=360, order=4):  *"""*  *对 ECG 信号进行带通滤波，去除噪声*  *"""* nyquist = 0.5 \* fs  low = lowcut / nyquist  high = highcut / nyquist  b, a = signal.butter(order, [low, high], btype='band')  return signal.filtfilt(b, a, signal\_data)  *# 数据分割函数* def preprocess\_data(record\_id='100', window\_size=200):  *"""*  *读取并处理 ECG 信号，将其分割成小片段*  *"""* record, annotation = load\_record(record\_id)   *# 提取 ECG 信号* ecg\_signal = record.p\_signal[:, 0] *# 使用第一个通道的数据*   *# 对 ECG 信号进行滤波* filtered\_ecg = bandpass\_filter(ecg\_signal)   *# 分割信号* samples = []  for i in range(0, len(filtered\_ecg) - window\_size, window\_size):  samples.append(filtered\_ecg[i:i + window\_size])   return np.array(samples)  if \_\_name\_\_ == "\_\_main\_\_":  preprocess\_data('100') *# 处理 record 100* |

**提示**：

* 分段处理时可根据固定窗口（如 2-5 秒）划分数据，每个段作为一个样本；
* 同时需结合注释信息生成对应的标签（例如正常/异常）。

首先运行 preprocess.py，将 MIT-BIH 数据集（例如 100 记录）进行滤波处理并保存为 .npy 文件：

|  |
| --- |
| Shell python data\_preprocess.py |



3. **模型构建（Vanilla RNN）**

在项目中创建一个新的 Python 文件（例如 model.py），编写如下代码：

|  |
| --- |
| Python import tensorflow as tf from tensorflow.keras.models import Sequential from tensorflow.keras.layers import SimpleRNN, Dense, Dropout  def build\_rnn\_model(timesteps=200, num\_features=1, num\_classes=2):  *"""*  *构建 Vanilla RNN 模型。*  *"""* model = Sequential([  SimpleRNN(64, activation='tanh', input\_shape=(timesteps, num\_features)),  Dropout(0.2),  Dense(num\_classes, activation='softmax')  ])   model.compile(optimizer='adam',  loss='sparse\_categorical\_crossentropy',  metrics=['accuracy'])  return model  if \_\_name\_\_ == "\_\_main\_\_":  model = build\_rnn\_model()  model.summary() |

说明：

* timesteps 根据你的分段长度（例如 200 个时间步）进行设置；
* num\_classes 为分类任务的类别数，如正常与异常两类。

4. **模型训练与验证**

创建训练脚本（train.py）：

**数据加载与处理**

在脚本中加载预处理好的数据，并对数据进行分割：

|  |
| --- |
| Python import numpy as np import tensorflow as tf from sklearn.model\_selection import train\_test\_split import matplotlib.pyplot as plt from model import build\_rnn\_model  *# 设置随机种子以保证结果复现性* import random seed = 42 np.random.seed(seed) tf.random.set\_seed(seed) random.seed(seed)  *# 加载预处理后的数据（假设已经处理并保存在 npy 文件中）* def load\_data():  *# 加载处理后的 ECG 数据，假设每个样本为 200 个时间步* data = np.load("ecg\_filtered\_100.npy")  data\_segments = data.reshape(-1, 200, 1) *# 假设每个样本为200个时间步* labels = np.zeros(len(data\_segments)) *# 仅示例，真实任务中需要根据注释生成标签* return data\_segments, labels  *# 加载数据* X, y = load\_data()  *# 划分数据集（70% 训练，15% 验证，15% 测试）* X\_train, X\_temp, y\_train, y\_temp = train\_test\_split(X, y, test\_size=0.3, random\_state=42) X\_val, X\_test, y\_val, y\_test = train\_test\_split(X\_temp, y\_temp, test\_size=0.5, random\_state=42) |

**训练模型**

继续在 train.py 中加入模型训练部分：

|  |
| --- |
| Python *# 构建模型* model = build\_rnn\_model(timesteps=200, num\_features=1, num\_classes=2)  *# 配置回调函数（早停和模型检查点）* from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True) checkpoint = ModelCheckpoint('best\_model.h5', monitor='val\_accuracy', save\_best\_only=True)  *# 训练模型* history = model.fit(X\_train, y\_train,  validation\_data=(X\_val, y\_val),  epochs=50,  batch\_size=32,  callbacks=[early\_stopping, checkpoint]) |

**模型评估与可视化**

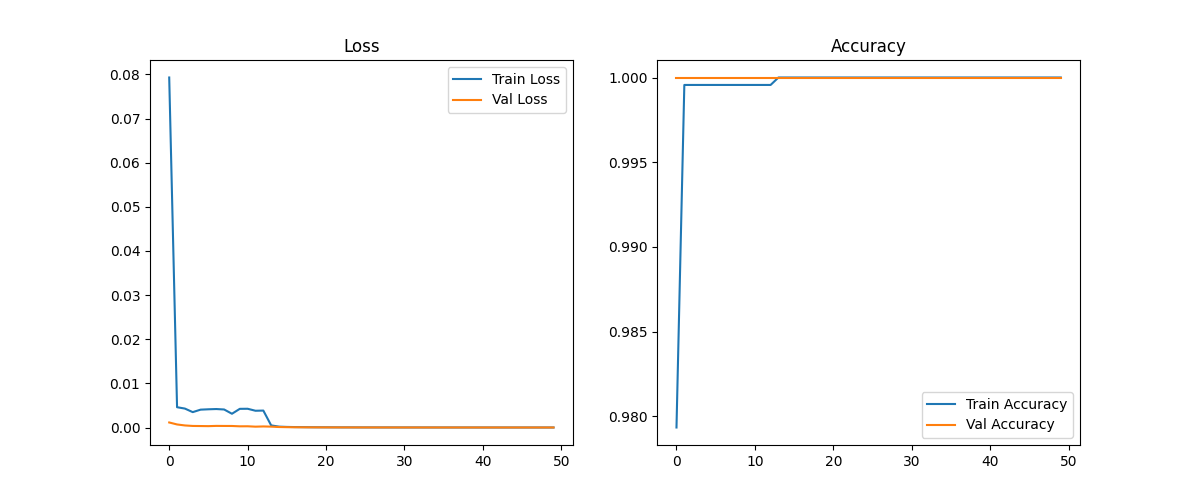
在同一脚本中添加测试集评估和结果可视化代码：

|  |
| --- |
| Python *# 评估模型* test\_loss, test\_accuracy = model.evaluate(X\_test, y\_test) print(f"Test Loss: {test\_loss}, Test Accuracy: {test\_accuracy}")  *# 可视化训练过程* plt.figure(figsize=(12, 5))  *# 绘制损失曲线* plt.subplot(1, 2, 1) plt.plot(history.history['loss'], label='Train Loss') plt.plot(history.history['val\_loss'], label='Val Loss') plt.legend() plt.title('Loss')  *# 绘制准确率曲线* plt.subplot(1, 2, 2) plt.plot(history.history['accuracy'], label='Train Accuracy') plt.plot(history.history['val\_accuracy'], label='Val Accuracy') plt.legend() plt.title('Accuracy')  plt.show() |

运行 train.py 来训练模型并评估其性能：

|  |
| --- |
| Shell python train.py |





可以看到，损失曲线在训练过程中急剧下降，准确率曲线急剧上升，这说明我们的模型正在训练的越来越好，到最后准确率达到100%

5. **模型预测**

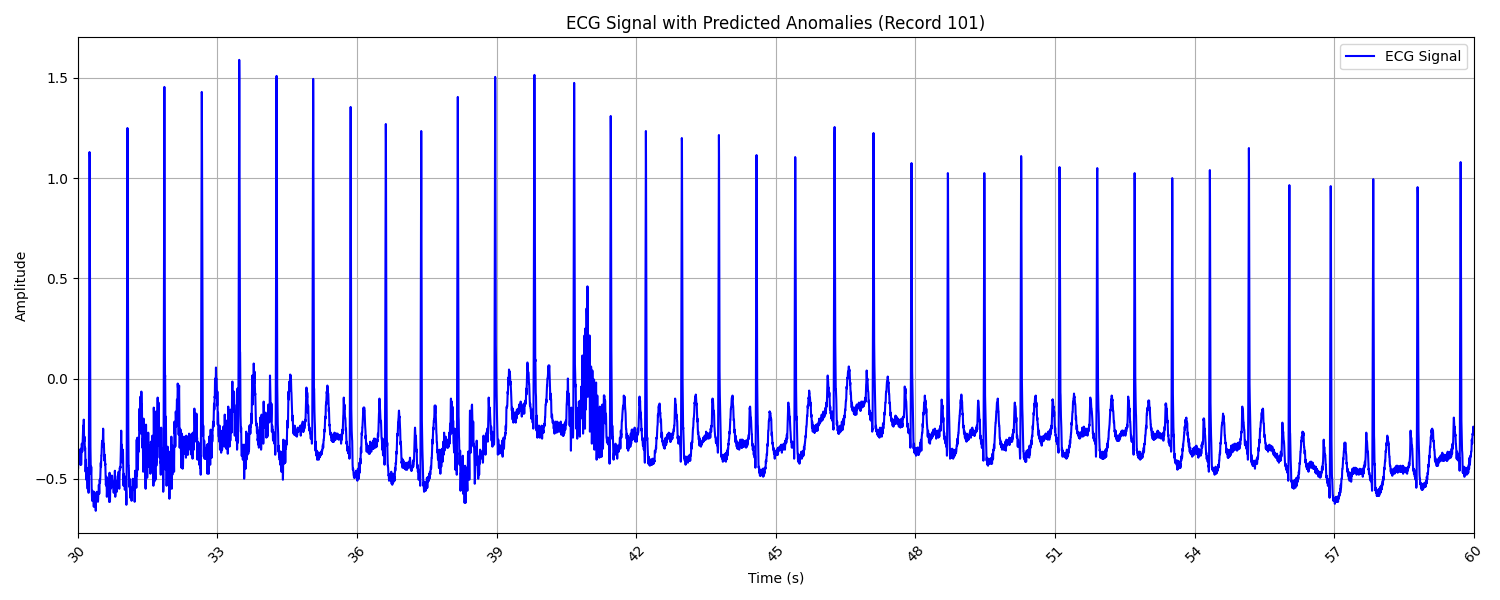
新建predict文件进行预测：

|  |
| --- |
| Python import numpy as np import wfdb import matplotlib.pyplot as plt from tensorflow.keras.models import load\_model from preprocess import preprocess\_data *# 引用预处理函数*  *# 加载训练好的模型* model = load\_model('best\_model.h5')  *# 使用模型进行预测* def predict\_ecg(record\_id='101', window\_size=200, start\_time=0, end\_time=60):  *"""*  *预测给定 ECG 数据集的结果，限制显示时间范围在 start\_time 和 end\_time 之间*  *"""*  *# 预处理数据* ecg\_data = preprocess\_data(record\_id, window\_size)   *# 获取采样率* fs = 360 *# 采样率，单位：Hz*  *# 限制数据为指定时间段内的样本* min\_samples = int(start\_time \* fs)  max\_samples = int(end\_time \* fs)   *# 截取对应时间段的数据* ecg\_data = ecg\_data[min\_samples // window\_size:max\_samples // window\_size]   *# 使用模型进行预测* predictions = model.predict(ecg\_data)   *# 获取预测结果（假设为二分类：0 正常，1 异常）* predicted\_classes = np.argmax(predictions, axis=1)   return ecg\_data, predicted\_classes  *# 可视化 ECG 信号与预测结果* def plot\_ecg\_with\_time(record\_id='101', window\_size=200, start\_time=0, end\_time=60):  *"""*  *使用时间轴绘制 ECG 信号，允许用户查看任意时间段的 ECG 信号*  *"""*  *# 读取整个 ECG 信号* record = wfdb.rdrecord(f'./data/mitdb/{record\_id}')  ecg\_signal = record.p\_signal[:, 0] *# 使用第一个通道的 ECG 信号*   *# 获取采样率* fs = record.fs *# 采样频率，单位为 Hz*   *# 创建时间轴（单位：秒）* time\_axis = np.arange(len(ecg\_signal)) / fs *# 每个数据点对应的时间*   *# 根据给定的时间范围限制数据* min\_samples = int(start\_time \* fs)  max\_samples = int(end\_time \* fs)  ecg\_signal = ecg\_signal[min\_samples:max\_samples]  time\_axis = time\_axis[min\_samples:max\_samples]   *# 获取预测结果* ecg\_data, predicted\_classes = predict\_ecg(record\_id, window\_size, start\_time, end\_time)   *# 设置图像大小（宽度增加，以便更好地查看信号）* plt.figure(figsize=(15, 6)) *# 增加图像的宽度*   *# 绘制信号* plt.plot(time\_axis, ecg\_signal, label="ECG Signal", color='blue')   *# 标注预测为异常的部分（假设预测结果为 1 的部分是异常）* for i in range(len(predicted\_classes)):  if predicted\_classes[i] == 1: *# 异常*  *# 标记异常部分，红色标记* plt.axvspan(i \* window\_size / fs + start\_time, (i + 1) \* window\_size / fs + start\_time, color='red', alpha=0.5)   *# 优化横坐标显示密度*  *# 使用 np.linspace 生成合理的刻度* xticks = np.linspace(start\_time, end\_time, num=11) *# 生成从 start\_time 到 end\_time 的 11 个点* plt.xticks(xticks, rotation=45) *# 设置 x 轴刻度并旋转标签*   *# 标题和标签* plt.title(f"ECG Signal with Predicted Anomalies (Record {record\_id})")  plt.xlabel("Time (s)") *# 横坐标单位为秒* plt.ylabel("Amplitude")  plt.legend()  plt.grid(True)   *# 设置横坐标的范围* plt.xlim([start\_time, end\_time])   *# 调整布局，避免图像显示过于拥挤* plt.tight\_layout()   *# 显示图像* plt.show()  if \_\_name\_\_ == "\_\_main\_\_":  *# 可视化 ECG 信号，并标注预测为异常的部分* plot\_ecg\_with\_time('101', window\_size=200, start\_time=30, end\_time=60) *# 查看从 30 秒到 60 秒的信号* |

预测结果为蓝色则说明心电图正常，如果为红色则说明心电图异常：



预测101号数据第30秒到第60秒，显示心电图正常：



预测101号样本第160秒到第190秒，显示心电图异常：



