下面是一个使用 \*\*GRU\*\*（门控循环单元）的代码来预测股票价格的完整示例。GRU 的结构与 LSTM 相似，但参数更少，计算效率更高，适合一些对短期记忆要求不高的任务。代码包含数据预处理、多特征支持、GRU 模型的定义、训练和测试。

### 完整代码

```python

# 导入库

import numpy as np

import pandas as pd

import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.preprocessing import MinMaxScaler

from sklearn.model\_selection import train\_test\_split

from torch.utils.data import DataLoader, TensorDataset

import matplotlib.pyplot as plt

# 1. 加载并预处理数据

# 假设数据在 data.csv 中，有多列 'Open'、'High'、'Low'、'Close'、'Volume'

data = pd.read\_csv('data.csv')

features = data[['Open', 'High', 'Low', 'Close', 'Volume']].values

# 标准化数据

scaler = MinMaxScaler(feature\_range=(0, 1))

scaled\_features = scaler.fit\_transform(features)

# 创建时间序列数据

def create\_sequences(data, seq\_length):

sequences = []

labels = []

for i in range(len(data) - seq\_length):

sequences.append(data[i:i+seq\_length])

labels.append(data[i+seq\_length, 3]) # 使用 'Close' 列（索引 3）作为预测目标

return np.array(sequences), np.array(labels)

seq\_length = 60 # 使用前 60 天的数据预测下一天

X, y = create\_sequences(scaled\_features, seq\_length)

# 转换为 Tensor

X = torch.tensor(X, dtype=torch.float32)

y = torch.tensor(y, dtype=torch.float32)

# 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)

train\_data = TensorDataset(X\_train, y\_train)

test\_data = TensorDataset(X\_test, y\_test)

train\_loader = DataLoader(train\_data, batch\_size=32, shuffle=True)

test\_loader = DataLoader(test\_data, batch\_size=32)

# 2. 定义 GRU 模型

class GRUModel(nn.Module):

def \_\_init\_\_(self, input\_size=5, hidden\_size=50, num\_layers=2, output\_size=1):

super(GRUModel, self).\_\_init\_\_()

self.hidden\_size = hidden\_size

self.num\_layers = num\_layers

self.gru = nn.GRU(input\_size, hidden\_size, num\_layers, batch\_first=True)

self.fc = nn.Linear(hidden\_size, output\_size)

def forward(self, x):

h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_size).to(x.device)

out, \_ = self.gru(x, h0)

out = self.fc(out[:, -1, :])

return out

# 3. 设置训练参数

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

model = GRUModel().to(device)

criterion = nn.MSELoss()

optimizer = optim.Adam(model.parameters(), lr=0.001)

epochs = 50

# 4. 训练模型

model.train()

for epoch in range(epochs):

for X\_batch, y\_batch in train\_loader:

X\_batch, y\_batch = X\_batch.to(device), y\_batch.to(device)

optimizer.zero\_grad()

output = model(X\_batch)

loss = criterion(output, y\_batch)

loss.backward()

optimizer.step()

if (epoch+1) % 10 == 0:

print(f"Epoch [{epoch+1}/{epochs}], Loss: {loss.item():.4f}")

# 5. 测试模型

model.eval()

predictions = []

actuals = []

with torch.no\_grad():

for X\_batch, y\_batch in test\_loader:

X\_batch, y\_batch = X\_batch.to(device), y\_batch.to(device)

output = model(X\_batch)

predictions.append(output.cpu().numpy())

actuals.append(y\_batch.cpu().numpy())

# 转换为数组

predictions = np.concatenate(predictions).reshape(-1, 1)

actuals = np.concatenate(actuals).reshape(-1, 1)

# 反向标准化

predictions = scaler.inverse\_transform(np.hstack([np.zeros((predictions.shape[0], features.shape[1]-1)), predictions]))[:, -1]

actuals = scaler.inverse\_transform(np.hstack([np.zeros((actuals.shape[0], features.shape[1]-1)), actuals]))[:, -1]

# 6. 可视化结果

plt.figure(figsize=(10,6))

plt.plot(actuals, label='Actual Price')

plt.plot(predictions, label='Predicted Price')

plt.xlabel("Days")

plt.ylabel("Price")

plt.legend()

plt.show()

# 7. 保存模型（可选）

torch.save(model.state\_dict(), 'stock\_gru\_model.pth')

```

### 代码说明

- \*\*数据预处理\*\*：多特征数据（如开盘价、最高价、最低价、收盘价和交易量）会被标准化，并生成用于训练的时间序列。

- \*\*GRUModel\*\*：GRU 模型代替了 LSTM，通过 `nn.GRU` 层实现，其他结构保持相似。

- \*\*预测与反向标准化\*\*：为了绘制实际值和预测值的对比，反向标准化只应用于预测值和实际值的目标列（即收盘价）。

运行代码后，你将看到实际价格和预测价格的对比图。根据需求和数据特征，适当调整 `seq\_length`、`hidden\_size` 和 `num\_layers` 等超参数可以进一步优化模型效果。