代码讲解

前言

根据 ItNC assignment.ipynb 这个文件的内容大致可以得知我们的任务。

1. 需要增加3个文件， model.py train.py requirements.txt 。
2. 需要 python train.py 直接开始训练模型。
3. 使用 RNN ,目的是给定时间序列预测接下来 t+5 的结果，根据训练数据来看，就是预测销售量。
4. 得到的模型要通过 test.py 评估，内容按照该文件注释，这里我们需要更改 test.py 文件一部分内容。

接下来根据以上任务我会给出相应代码。 由于理解可能会有偏差，我尽可能留有更改余地方便之后更改。

requirements.txt

```
torch==2.1.2
pandas==2.1.4
numpy==1.26.4
tqdm==4.65.0
```

这部分是我本地用到的包环境，我觉得不需要解释。

model.py

```
import torch
import torch.nn as nn

class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.RNN(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.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        out = out.unsqueeze(-1)
        return out
```

这一部分我不想设计太复杂，一是太复杂可能会有bug，二是性能。

因此我就只要了一个 RNN层 和一个 全连接层。

网络设计比较简单，这里我不想接着讲解了，都是很板的。

唯一需要注重的点是数据的维度或者说是 shape 。

我这里只解释输入数据，和输出数据的 shape。

输入数据 (1, input_seq_length + 2)

输出数据 一维，长度为5 。

train.py

```
import argparse
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from model import RNNModel
from tqdm import tqdm

def load_data(file_path, input_seq_length, pred_seq_length):
    df = pd.read_csv(file_path)
    gb = df.groupby(["store", "product"])
    groups = {x: gb.get_group(x) for x in gb.groups}

    inputs, targets = [], []

    for key, data in groups.items():
        X = data['number_sold'].values
        X = X.reshape(-1, 1)
        N = X.shape[0]
        start = input_seq_length
        while start + pred_seq_length <= N:
            input = X[(start - input_seq_length) : start, :]
            input = np.concatenate((input, np.array([[key[0]], [key[1]]])),
axis=0)
            target = X[start : (start + pred_seq_length), :]
            inputs.append(input)
            targets.append(target)
            start += pred_seq_length

    inputs = np.array(inputs)
    targets = np.array(targets)

    return torch.tensor(inputs, dtype=torch.float32), torch.tensor(targets,
dtype=torch.float32)
```

```
def train_model(train_loader, model, criterion, optimizer, num_epochs, device,
model_path):
    model.train()
    for epoch in range(num_epochs):
        epoch_loss = 0.0
        for input, target in tqdm(train_loader, desc=f'Epoch
{epoch+1}/{num_epochs}', unit='batch'):
            input, target = input.to(device), target.to(device)
            output = model(input)
            loss = criterion(output, target)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            epoch_loss += loss.item()
        avg_loss = epoch_loss / len(train_loader)
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}')
    torch.save(model.state_dict(), model_path)

def main(args):
    input_seq_length = args.input_seq_length
    pred_seq_length = args.pred_seq_length
    batch_size = args.batch_size
    num_epochs = args.num_epochs
    learning_rate = args.learning_rate
    hidden_size = args.hidden_size
    num_layers = args.num_layers

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

    inputs, outputs = load_data(args.data_file, input_seq_length, pred_seq_length)
    dataset = TensorDataset(inputs, outputs)
    train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

    model = RNNModel(input_size=1, hidden_size=hidden_size,
output_size=pred_seq_length, num_layers=num_layers).to(device)
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    train_model(train_loader, model, criterion, optimizer, num_epochs, device,
args.model_path)

if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_file', type=str, default='train.csv', help='Path
to the CSV data file')
    parser.add_argument('--model_path', type=str, default='model.pth', help='Path
to the model')
    parser.add_argument('--input_seq_length', type=int, default=50, help='Length
of input sequences')
    parser.add_argument('--pred_seq_length', type=int, default=5, help='Length of
prediction sequences')
    parser.add_argument('--batch_size', type=int, default=32, help='Batch size for
training')
```

```
    parser.add_argument('--num_epochs', type=int, default=15, help='Number of
epochs for training')
    parser.add_argument('--learning_rate', type=float, default=0.001,
help='Learning rate')
    parser.add_argument('--hidden_size', type=int, default=128, help='Number of
hidden units in RNN')
    parser.add_argument('--num_layers', type=int, default=2, help='Number of
layers in RNN')
    args = parser.parse_args()
    main(args)
```

这部分代码就几个部分：

1. 加载训练数据集
2. 训练模型
3. main函数
4. 参数处理

写道最后发现参数处理不需要，但我依然留下了。 如果你们需要自己改参数，可以改里面相应的 default 值。

接下来我就仅解释加载数据集部分和参数部分。

因为其它两个函数，逻辑很板，没什么讲的，深度学习都是这么搞得，想了解就浏览器即可。

加载数据集这个函数我们会返回训练所需的输入数据，和给定的需要预测的数据。

对于一个数据集我们会把它按照 store,product 的组合进行分类，很好理解，对某个商店某个商品进行预测，比把所有数据整合起来更合理。

所有我们每次训练的数据其实就是对于 store,product 中某一段 number_sold 按时间顺序截取下来，根据我们的参数设定来看具体截取多长。

之后对这段数据再加上 store,product 两个数便于标识这段序列的归属，由于预测固定为 t+5 ，我们截取长度固定为5。

之后把每组放到列表里返回列表就行。

加上 store,product 两个数是非常重要的，这个也在给的 ItNC assignment.ipynb 文件中有提到要使用。

好了，前面的部分可以看成黑盒没什么特别需要改的，接下来参数介绍才是最重要的，根据不同的参数分配，模型的精度也会有不同。 你们可以通过了解参数含义后自己更改，训练出更好的模型，不想训练也可以用我给你们的模型。

```
'--data_file' 训练集地址
'--model_path' 保存模型地址
'--input_seq_length' 就是测试代码里的 window_size 这一段长度会用于预测。一般越长越好，
但是训练会需要更对时长。可改。
'--pred_seq_length' 预测时间长度，这里固定为5。
'--batch_size' 每一批次数据个数，一般为 32， 64， 128。可改，不过建议32就行。
'--num_epochs' 训练轮数，这个可自行更改。
'--learning_rate' 学习率。可改。
```

```
'--hidden_size' 隐藏层大小，太小不太好太大也不好，可以随便试试看，可改。
'--num_layers' RNN层数，建议1-2
```

test.py

```
import numpy as np
import pandas as pd
from sklearn.metrics import mean_absolute_percentage_error
import argparse
import torch
from model import Model


parser = argparse.ArgumentParser()
parser.add_argument('--data_file', type=str, default='train.csv', help='Path to
the CSV data file')
parser.add_argument('--model_path', type=str, default='best_model.pth', help='Path
to the model')
parser.add_argument('--input_seq_length', type=int, default=50, help='Length of
input sequences')
parser.add_argument('--pred_seq_length', type=int, default=5, help='Length of
prediction sequences')
parser.add_argument('--batch_size', type=int, default=32, help='Batch size for
training')
parser.add_argument('--num_epochs', type=int, default=25, help='Number of epochs
for training')
parser.add_argument('--learning_rate', type=float, default=0.001, help='Learning
rate')
parser.add_argument('--hidden_size', type=int, default=128, help='Number of hidden
units in RNN')
parser.add_argument('--num_layers', type=int, default=2, help='Number of layers in
RNN')
args = parser.parse_args()

df = pd.read_csv("test_example.csv")
model = Model(args)  # load your model here
window_size = 25  # please fill in your own choice: this is the length of history
you have to decide


model.init()

# split the data set by the combination of `store` and `product``
gb = df.groupby(["store", "product"])
groups = {x: gb.get_group(x) for x in gb.groups}
scores = {}

for key, data in groups.items():
    # By default, we only take the column `number_sold`.
    # Please modify this line if your model takes other columns as input
```

```
    # X = data.drop(["Date", "store", "product"], axis=1).values  # convert to
numpy array
    # N = X.shape[0]  # total number of testing time steps

    X = data['number_sold'].values
    X = X.reshape(-1, 1)
    N = X.shape[0]

    mape_score = []
    start = window_size
    while start + 5 <= N:
        inputs = X[(start - window_size) : start, :]
        inputs = np.concatenate((inputs, np.array([[key[0]], [key[1]]])), axis=0)
        targets = X[start : (start + 5), :]

        # you might need to modify `inputs` before feeding it to your model, e.g.,
convert it to PyTorch Tensors
        # you might have a different name of the prediction function. Please
modify accordingly
        predictions = list(model.predict(torch.tensor(inputs,
dtype=torch.float32))[0])
        start += 5
        # calculate the performance metric
        mape_score.append(mean_absolute_percentage_error(targets, predictions))
    scores[key] = mape_score

# save the performance metrics to file
np.savez("score.npz", scores=scores)
```

为了尽量按照 test.py 的逻辑执行，我被迫把之前代码封装成了类。 以下是更改后的代码。

model.py

```
import argparse
import pandas as pd
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from tqdm import tqdm

class RNNModel(nn.Module):
    def __init__(self, input_size, hidden_size, output_size, num_layers):
        super(RNNModel, self).__init__()
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.rnn = nn.RNN(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.rnn(x, h0)
        out = self.fc(out[:, -1, :])
        out = out.unsqueeze(-1)
        return out

class Model:

    args = None

    def __init__(self, args):
        self.args = args

        self.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
        self.input_seq_length = self.args.input_seq_length
        self.pred_seq_length = self.args.pred_seq_length
        self.batch_size = self.args.batch_size
        self.num_epochs = self.args.num_epochs
        self.learning_rate = self.args.learning_rate
        self.hidden_size = self.args.hidden_size
        self.num_layers = self.args.num_layers

    def load_data(self, file_path, input_seq_length, pred_seq_length):
        df = pd.read_csv(file_path)
        gb = df.groupby(["store", "product"])
        groups = {x: gb.get_group(x) for x in gb.groups}

        inputs, targets = [], []

        for key, data in groups.items():
            X = data['number_sold'].values
            X = X.reshape(-1, 1)
            N = X.shape[0]
            start = input_seq_length
            while start + pred_seq_length <= N:
                input = X[(start - input_seq_length) : start, :]
                input = np.concatenate((input, np.array([[key[0]], [key[1]]])),
axis=0)
                target = X[start : (start + pred_seq_length), :]
                inputs.append(input)
                targets.append(target)
                start += pred_seq_length

        inputs = np.array(inputs)
        targets = np.array(targets)

        return torch.tensor(inputs, dtype=torch.float32), torch.tensor(targets,
dtype=torch.float32)

    def train_model(self, train_loader, model, criterion, optimizer, num_epochs,
device, model_path):
        model.train()
        for epoch in range(num_epochs):
```

```
            epoch_loss = 0.0
            for input, target in tqdm(train_loader, desc=f'Epoch
{epoch+1}/{num_epochs}', unit='batch'):
                input, target = input.to(device), target.to(device)
                output = model(input)
                loss = criterion(output, target)
                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
                epoch_loss += loss.item()
            avg_loss = epoch_loss / len(train_loader)
            print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}')
        torch.save(model.state_dict(), model_path)

    def train(self):
        args = self.args
        input_seq_length = args.input_seq_length
        pred_seq_length = args.pred_seq_length
        batch_size = args.batch_size
        num_epochs = args.num_epochs
        learning_rate = args.learning_rate
        hidden_size = args.hidden_size
        num_layers = args.num_layers

        device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

        inputs, outputs = self.load_data(args.data_file, input_seq_length,
pred_seq_length)
        dataset = TensorDataset(inputs, outputs)
        train_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True)

        model = RNNModel(input_size=1, hidden_size=hidden_size,
output_size=pred_seq_length, num_layers=num_layers).to(device)
        criterion = nn.MSELoss()
        optimizer = optim.Adam(model.parameters(), lr=learning_rate)

        self.train_model(train_loader, model, criterion, optimizer, num_epochs,
device, args.model_path)

    def init(self):
        self.model = RNNModel(input_size=1, hidden_size=self.hidden_size,
output_size=self.pred_seq_length, num_layers=self.num_layers).to(self.device)
        self.model.load_state_dict(torch.load(self.args.model_path))
        self.model.eval()

    def predict(self, input):
        with torch.no_grad():
            input = input.unsqueeze(0)
            return self.model(input)
```

train.py

```
import argparse
from model import Model


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_file', type=str, default='train.csv', help='Path
to the CSV data file')
    parser.add_argument('--model_path', type=str, default='model.pth', help='Path
to the model')
    parser.add_argument('--input_seq_length', type=int, default=50, help='Length
of input sequences')
    parser.add_argument('--pred_seq_length', type=int, default=5, help='Length of
prediction sequences')
    parser.add_argument('--batch_size', type=int, default=32, help='Batch size for
training')
    parser.add_argument('--num_epochs', type=int, default=25, help='Number of
epochs for training')
    parser.add_argument('--learning_rate', type=float, default=0.001,
help='Learning rate')
    parser.add_argument('--hidden_size', type=int, default=128, help='Number of
hidden units in RNN')
    parser.add_argument('--num_layers', type=int, default=2, help='Number of
layers in RNN')
    args = parser.parse_args()

    model = Model(args)
    model.train()
```

总结

我给你训练的模型目前是对于所有的训练数据集以及给出的测试数据集，MAPE基本上都在0.05以下是更改后的代码。

我认为效果算不错的了。

如果你想更进一步，可以试着拉高 input_seq_length 。

debug好累QAQ。