哈尔滨工业大学

模式识别与深度学习

实验四：循环神经网络

姓　　名：刘天瑞

院　（系）：未来技术学院

专　　业：视听觉信息处理

学　　号：7203610121

指导教师：左旺孟

提交日期：2023.5.23

摘 要

本次深度学习实验为模式识别与深度学习课程的实验四：即循环神经网络的实现，在本次实验中，我还是利用了 Python 中的 Pytorch 库从而自己实现了 RNN、GRU、LSTM 以及 Bi-LSTM 总共四种网络的结构，并且利用自己实现的上述网络 结构，完成了文本多分类和温度预测的任务。最终，我在文本多分类任务上可以达到的准确率，而采用 GRU 结构可以在温度预测任务上也取得了较好的效果。

关键词： 模式识别与深度学习，循环神经网络，RNN，GRU，LSTM，Bi-LSTM，文本多分类，温度预测

目 录

一、 深度学习框架与实验环境

本次实验采用的深度学习框架是 Python 中强大的深度学习库 Pytorch，整个实验是在 Visual Studio + Pip 环境下完成的。Pip 是一个开源的 Python 发行版本，可以很方便地安装许多深度学习所需要的模块包，而 Visual Studio 则是一个功能强大的 IDE，可以在其中完成 python 代码编写深度学习过程、进行训练测试等深度学习环节。由于我在大二时参加过美赛有接触到一些深度学习相关的工具知识，因此在本次实验中的配置环境过程相对得心应手。配置环境的具体流程比较繁琐，查看系统 Pytorch 库以及 cuda 版本如下图 1 所示（英伟达表示 cuda 可升级到的最高版本）：

```
C:\Users\刘天瑞>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> print(torch.__version__)
2.0.0+cu118
>>> ^Z
```

```
>>> import torch
>>> print(torch.backends.cudnn.version())
8700
```

```
C:\Users\刘天瑞>nvidia-smi
Sat Apr 29 18:24:48 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 531.41                 Driver Version: 531.41       CUDA Version: 12.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name                     TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                         |                      |               MIG M. |
|=========================================+======================+======================|
|   0  NVIDIA GeForce RTX 3060 L...  WDDM | 00000000:01:00.0  On |                  N/A |
| N/A   39C    P8              15W /  N/A|   1491MiB /  6144MiB |     21%      Default |
|                                         |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2884    C+G   ...on\112.0.1722.58\msedgewebview2.exe    N/A      |
|    0   N/A  N/A      4944    C+G   ...crosoft\Edge\Application\msedge.exe    N/A      |
|    0   N/A  N/A      5324    C+G   ...\cef\cef.win7x64\steamwebhelper.exe    N/A      |
|    0   N/A  N/A      9704    C+G   ...7.0_x64__w2gh52qy24etm\Nahimic3.exe    N/A      |
|    0   N/A  N/A      9900    C+G   C:\Windows\explorer.exe                   N/A      |
|    0   N/A  N/A     10908    C+G   ...nt.CBS_cw5n1h2txyewy\SearchHost.exe    N/A      |
|    0   N/A  N/A     11620    C+G   ...5n1h2txyewy\ShellExperienceHost.exe    N/A      |
|    0   N/A  N/A     13316    C+G   ...2txyewy\StartMenuExperienceHost.exe    N/A      |
|    0   N/A  N/A     16712    C+G   ...CBS_cw5n1h2txyewy\TextInputHost.exe    N/A      |
|    0   N/A  N/A     17748    C+G   ...GeForce Experience\NVIDIA Share.exe    N/A      |
|    0   N/A  N/A     19364    C+G   ...GeForce Experience\NVIDIA Share.exe    N/A      |
|    0   N/A  N/A     20712    C+G   ...1.0_x64__8wekyb3d8bbwe\Video.UI.exe    N/A      |
|    0   N/A  N/A     22552    C+G   ...t.LockApp_cw5n1h2txyewy\LockApp.exe    N/A      |
|    0   N/A  N/A     25060    C+G   ...4036\office6\promecefpluginhost.exe    N/A      |
|    0   N/A  N/A     25076    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe    N/A      |
|    0   N/A  N/A     25856    C+G   ...siveControlPanel\SystemSettings.exe    N/A      |
|    0   N/A  N/A     26908      C   ...Programs\Python\Python39\python.exe    N/A      |
|    0   N/A  N/A     26948    C+G   ...__8wekyb3d8bbwe\WindowsTerminal.exe    N/A      |
|    0   N/A  N/A     30532    C+G   ...22\Community\Common7\IDE\devenv.exe    N/A      |
|    0   N/A  N/A     31092    C+G   ...ft Office\root\Office16\WINWORD.EXE    N/A      |
|    0   N/A  N/A     31864    C+G   ...4036\office6\promecefpluginhost.exe    N/A      |
+-----------------------------------------------------------------------------+
```

图 1

二、 主要研究内容以及实验背景知识

2.1 四种网络结构模型的实现与介绍

在本次深度学习实验中，我利用 Python 中的 Pytorch 库来实现 RNN、GRU、LSTM 以及 Bi-LSTM 总共四种网络结构模型。具体而言，网络结构模型的实现被划分为两部分，其中一部分是实现循环神经网络单个模块 cell 的结构（具体运行代码保存在了程序 RNNcell_series.py 中），而另一部分为将各模块 cell 组合为完整的网络结构（具体运行代码保存在了程序 RNN_series.py 中）。

下面我就以 RNNcell 与 RNN 网络结构为例子，来详尽而简洁地说明我所实现的网络模型结构。

```
class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers = 1, bias = True, act_fn = 'tanh'):
        super(RNN, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.num_layers = num_layers
        self.bias = bias

        self.rnn_cell_list = nn.ModuleList()  # 与nn.Sequential类似，区别是nn.Sequential无法自定义forward函数（它会自动构建）

        self.rnn_cell_list.append(RNN_cell(self.input_size, self.hidden_size, self.bias, act_fn))

        for i in range(1, self.num_layers):
            self.rnn_cell_list.append(RNN_cell(self.hidden_size, self.hidden_size, self.bias, act_fn))
```

图 1

如上图 1 所示，在我所实现的 RNNcell 网络结构中，设置可以选择 tanh 或者 Relu 作为激活函数（此处我设置默认为 tanh），同时并且按照 RNN 网络的结构重写了 Forward 方法。而整体的 RNN 网络结构则被设定成了多个 RNNcell 模块的组合，这样一来便于可以根据目标指定的模型层数来搭建多层 RNN 网络结构模型。最后在 RNN 网络结构的 Forward()函数中，设置可以读入前一个 RNNcell 的输出，从而实现了“记忆存储”的功能。

我所实现的四种网络结构的接口与 torch.nn 中对应的网络结构模型的接口完全一致，所以在实际搭建的各类网络结构中可以直接替换为 torch.nn.RNN 等具体已有网络结构模型。

三、实验详细过程

3.1 文本多分类任务

3.1.1 数据预处理

相较于计算机视觉方面的任务，文本分类的任务则需要进行复杂的数据预处理。而数据预处理所相关的具体运行代码我将其直接保存在了与其他模块相同的文件夹下，以下我就按照文件程序执行的顺序来简述一下各阶段的数据预处理过程：

对于 build_words.py 这一部分所完成的预处理任务包括：

1. 读入 online_shopping_10_cats.csv 数据集；

2. 将 Label 映射成为 number 数据；

3. 调用 Python 的 jieba 库来对文本进行分词处理；

4. 完成训练集、测试集以及相关验证集的具体划分；

5. 将已经划分好的词的训练集、测试集以及验证集保存到本地文件夹下（此处我按照“标签，文本”的格式存放）。

而在运行 buildwords.py 程序完成之后，就可以在./Data/Online_Shopping 路径下得到(train/test/ver).words.txt 文件，其内容的具体样式如下图 2 所示：

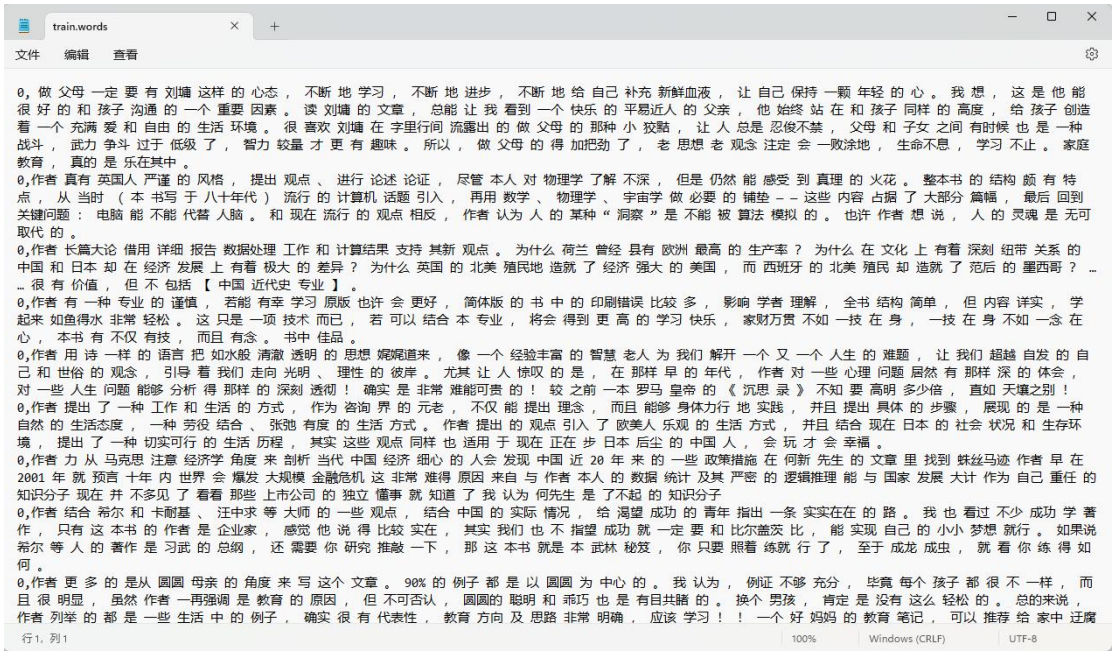


图 2

对于 build_vocab.py 这一部分所完成的预处理任务是按照词的顺序来构建词典，在运行 build_vocab.py 程序完成之后，就可以在./Data/Online_Shopping 路径下得到 vocab.words.txt 文件，其内容的具体样式如下图 3 所示：

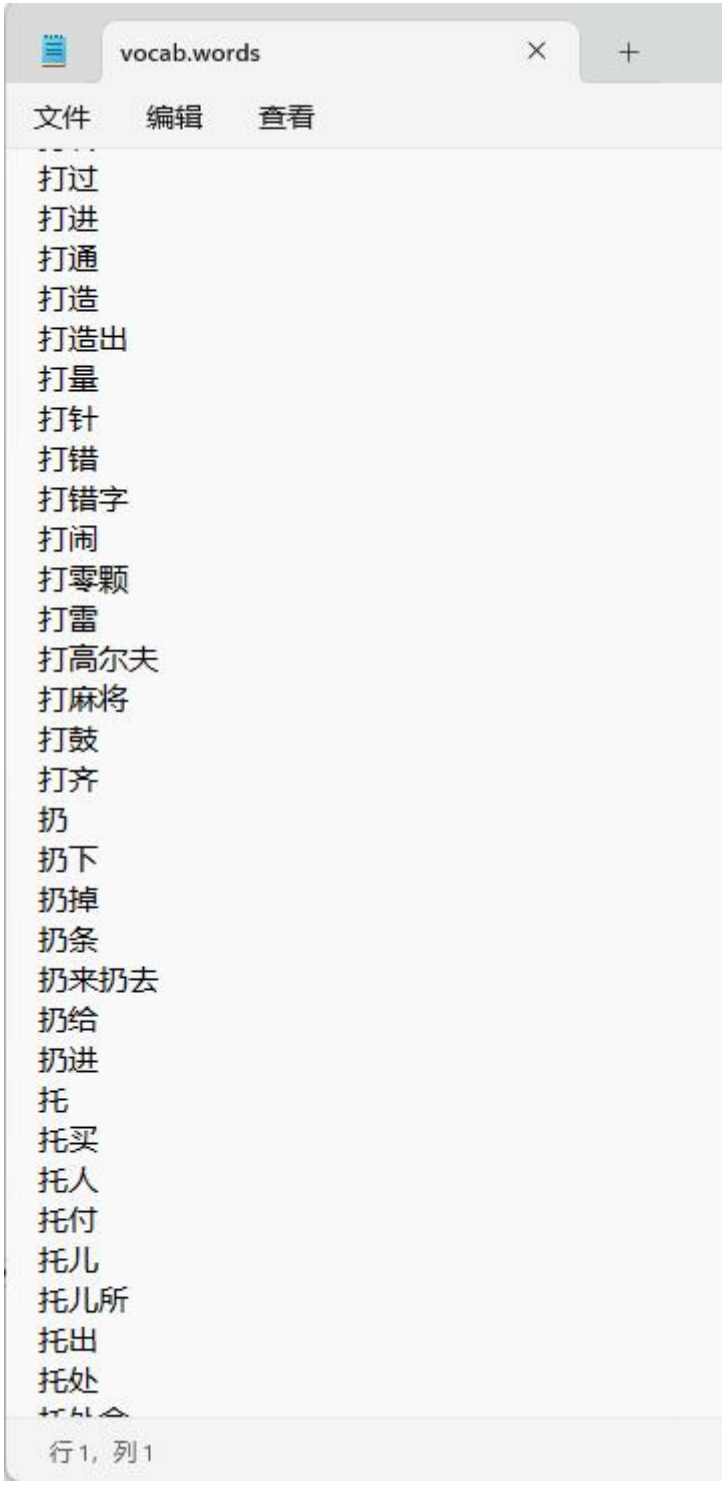


图 3

值得注意的是，这一部分只能根据训练集来构建词表。如果使用所有数据来构建词表，则会出现“标签泄露”的情况（即：在进行测试的时候所有词在 embedding 矩阵中都有对应的向量，故并不会被归为<UNK>）。

对于 build_embeddings.py 这一部分所完成的预处理任务是对 embedding 矩阵进行初始化处理，从而为 RNN 网络结构模型的 embedding 层提供初始化权重。在这一步骤中，我还使用了开源词向量 Chinese-Word-Vector 所提供的 sgns.merge.words。同时我还设置了 embedding 矩阵的大小为(vocab size +1, 300)，此处我对其的具体代码解释如下所示：

1. vocab size +1：代表 vocab 表中出现的所有词都有其对应的词向量，而多余剩下的 1 则表示<UNK>和<PAD>（此处为了简便表示，我将这两类标签统一归为一条向量，后续来看这对结果影响并不大）。

2. 300：代表词向量的维度，即一个词被表示成为了 300*1 的向量。而被设置为 300 则是因为开源词向量 Chinese-Word-Vectors 的维度即是 300。

对于词表中的词，将其词向量设置为在开源词向量 Chinese-Word-Vectors 中对应的 300 维向量，若其未出现在开源词向量 Chinese-Word-Vectors 中，则将其对应的 300 维词向量全部设置为 0。而标签<UNK>和<PAD>对应着 embedding 矩阵的第一行，也都全部设置为 0。

在运行 build_embeddings.py 完成之后，其初始化之后的词向量的权重也被保存在了./Data/Online_Shopping/w2v.npz 路径文件中。

最后对于 load_online_shopping.py 这一部分所完成的预处理任务是按照 max_len 长度来切分每一段文本（只保留从头开始的最长 max_len 长度的文本），并同时将文本转化为文本在 embedding 矩阵中所对应的行号（也即是在 vocab 中所对应的行号）。最终完成之后再将数据装入 dataloader 中。

3.1.2 搭建四种网络结构模型

用于文本多分类的网络结构模型具体运行代码部分如下图 4 所示：

```
class RNN(nn.Module):
    def __init__(self, embeddings, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.embedding = nn.Embedding.from_pretrained(embeddings = embeddings, freeze = False, padding_idx = 0)
        self.rnn = nn.RNN(embedding_dim, hidden_dim, batch_first = True)
        self.fc = nn.Linear(hidden_dim, output_dim)
```

图 4

其中总共包含了三层：embedding 层，RNN 层以及最后的全连接层。embedding 层使用了在上一个步骤中所导入进来的预训练处理初始化权重，RNN 层是我自行实现的循环神经网络结构，而全连接层则用于归一化结果。

最终，网络结构模型的输入为[batch size(64), max_len(50)]。而经过 embedding 层之后，实际输入到 RNN 层的向量为 [batch size(64), max_len(50), embedding dim(300)]。

3.1.3 网络结构模型训练过程

文本多分类任务的训练过程的具体运行代码保存在了 train.py 程序中，其主要模块（如模型训练，测试以及反向传播）主要来自于前几次的深度学习实验。而在本次深度学习实验中，我经过尝试额外使用了 Python 中的 sklearn 库的 classification_report 方法，这样可以直接计算模型在测试集上的准确率，召回率以及 F1 值。

而对于各类网络结构模型，我设置进行 100 轮 epoch 的训练，学习率设置为 0.0001，优化器采用 Adam 优化器，损失函数采用交叉熵函数。

3.2 温度预测任务

3.2.1 数据预处理

温度预测任务的数据预处理程序保存在了 load_jena.py 文件中，其主要的处理步骤如下所示：

1. 处理日期数据。可以通过 datetime.strptime()方法将原本的字符串转化成为 datetime 类型的数据，从而便于后续的进一步处理；

2. 将日期数据转化为正余弦函数的数值形式，从而使得模型在训练时能够感知到时间周期；

3. 对数据集进行归一化，剔除无用数据（即只保留了 month, day 所对应的 sin, cos 值以及温度，总共 5 个特征）；

4. 按照前 5 年为一段，后 2 年为另一段来划分数据集；

5. 根据原始数据的数据集来得到模型的输入与输出。并且同时需要将数据装入 dataloader 中。按照实验指导书的要求，需要以 5 天的温度值来预测 2 天的温度值。因此设置模型的输入为 5 天的温度值以及时间数据（大小 720*5 的 tensor 向量），输出为 2 天的温度值（大小 288*1 的 tensor 向量）。

3.2.2 搭建 GRU 网络结构模型

用于温度预测的网络结构模型具体运行代码部分如下图 5 所示：

```python
class Jena_Net(nn.Module):
    def __init__(self, embedding_dim, hidden_dim, output_dim):
        super().__init__()
        self.rnn = nn.GRU(embedding_dim, hidden_dim, batch_first = True)
        self.fc = nn.Linear(hidden_dim, output_dim)

    def forward(self, data):
        output, hidden = self.rnn(data)

        a = output[:, -1, :]
        b = hidden.squeeze(0)

        assert torch.equal(output[:, -1, :], hidden.squeeze(0))

        return self.fc(hidden.squeeze(0))
```

图 5

由于并不涉及到词与词向量之间的映射，因此我这里故意将 embedding 层去除掉，并且直接输入了[batch size(16), data of 5 days(720), feature num(5)]的 tensor 向量。

3.2.3 网络结构模型训练过程

温度预测任务的训练过程的具体运行代码保存在了 train_jena.py 程序中。这里进行 100 轮 epoch 的训练，学习率设置为 0.0001，优化器采用了 Adam 优化器，损失函数则采用 MSE 函数。

由于温度预测是预测任务，而不是前几次深度学习实验所做的分类任务，因此此处需要对损失函数以及 test()函数进行一系列的修改，此处就并不再多赘述了。

四、实验结果与分析

4.1 文本多分类任务实验结果

GRU 网络结构模型训练与评测结果如下图 6 和图 7 所示：

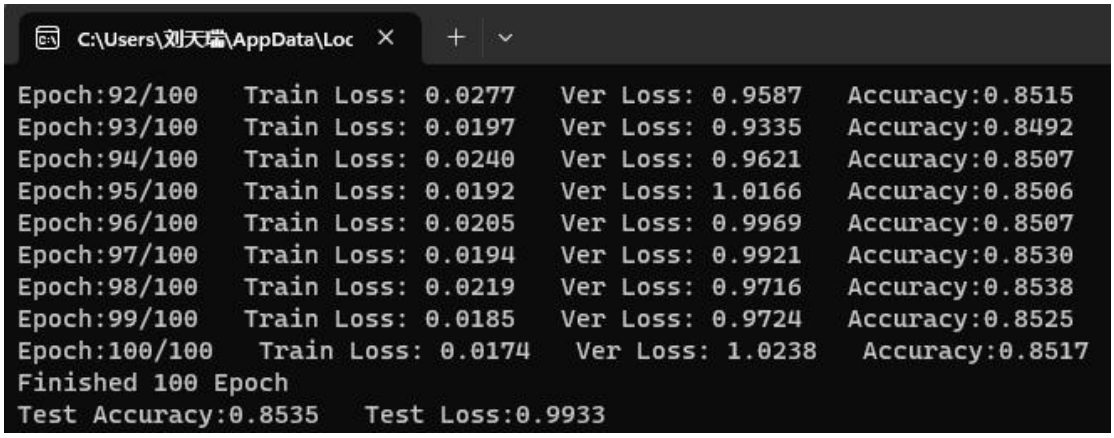
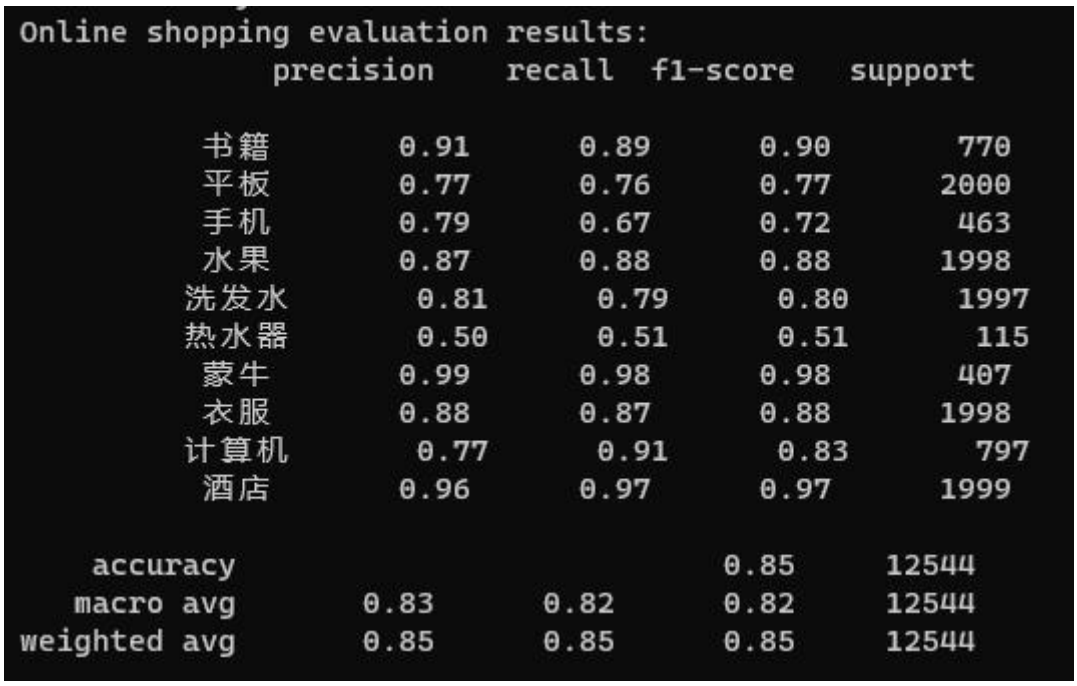

C:\Users\刘天瑞\AppData\Loc
Epoch:92/100 Train Loss: 0.0277 Ver Loss: 0.9587 Accuracy:0.8515
Epoch:93/100 Train Loss: 0.0197 Ver Loss: 0.9335 Accuracy:0.8492
Epoch:94/100 Train Loss: 0.0240 Ver Loss: 0.9621 Accuracy:0.8507
Epoch:95/100 Train Loss: 0.0192 Ver Loss: 1.0166 Accuracy:0.8506
Epoch:96/100 Train Loss: 0.0205 Ver Loss: 0.9969 Accuracy:0.8507
Epoch:97/100 Train Loss: 0.0194 Ver Loss: 0.9921 Accuracy:0.8530
Epoch:98/100 Train Loss: 0.0219 Ver Loss: 0.9716 Accuracy:0.8538
Epoch:99/100 Train Loss: 0.0185 Ver Loss: 0.9724 Accuracy:0.8525
Epoch:100/100 Train Loss: 0.0174 Ver Loss: 1.0238 Accuracy:0.8517
Finished 100 Epoch
Test Accuracy:0.8535 Test Loss:0.9933

图 6


Online shopping evaluation results:
precision recall f1-score support
书籍 0.91 0.89 0.90 770
平板 0.77 0.76 0.77 2000
手机 0.79 0.67 0.72 463
水果 0.87 0.88 0.88 1998
洗发水 0.81 0.79 0.80 1997
热水器 0.50 0.51 0.51 115
蒙牛 0.99 0.98 0.98 407
衣服 0.88 0.87 0.88 1998
计算机 0.77 0.91 0.83 797
酒店 0.96 0.97 0.97 1999
accuracy 0.85 12544
macro avg 0.83 0.82 0.82 12544
weighted avg 0.85 0.85 0.85 12544

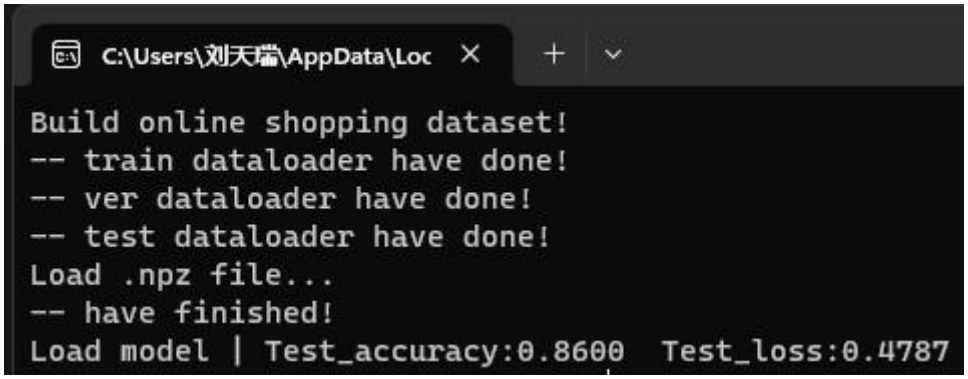

C:\Users\刘天瑞\AppData\Loc
Build online shopping dataset!
-- train dataloader have done!
-- ver dataloader have done!
-- test dataloader have done!
Load .npz file...
-- have finished!
Load model | Test_accuracy:0.8600 Test_loss:0.4787

图 7

RNN 网络结构模型训练与评测结果如下图 8 和图 9 所示：

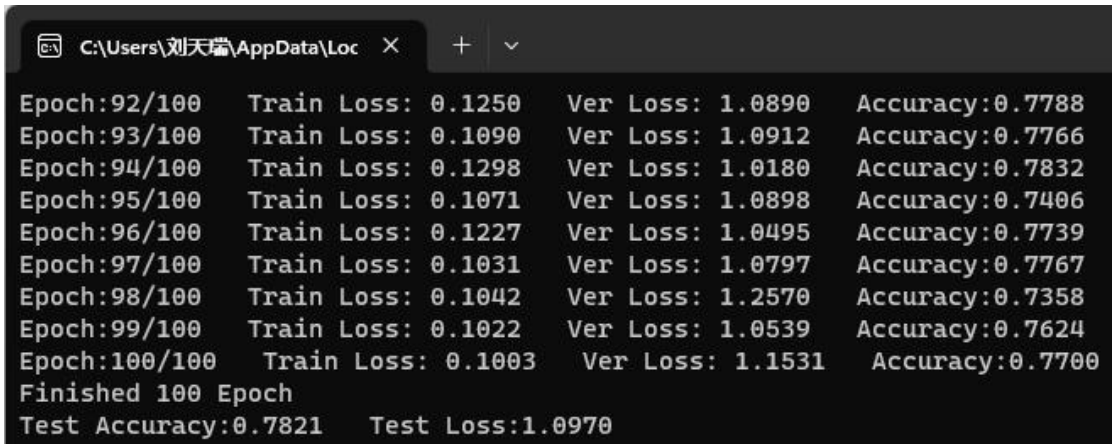
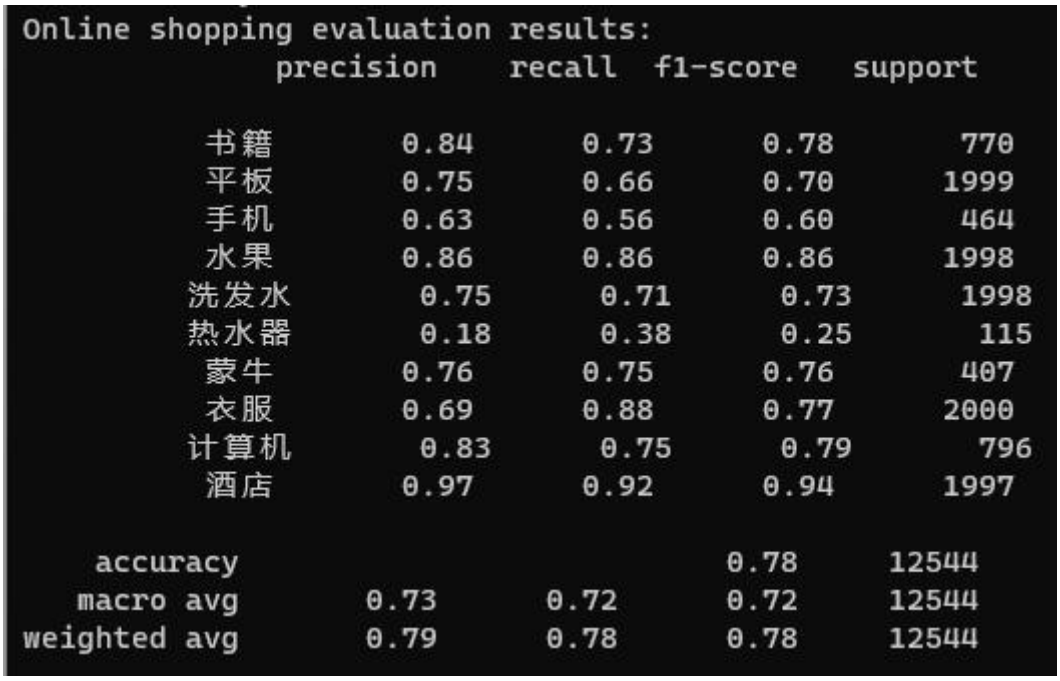


图 8



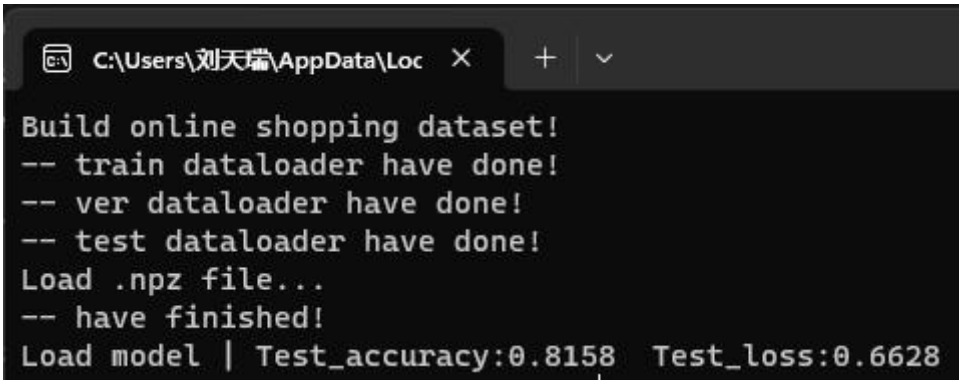


图 9

LSTM 网络结构模型训练与评测结果如下图 10 和图 11 所示：

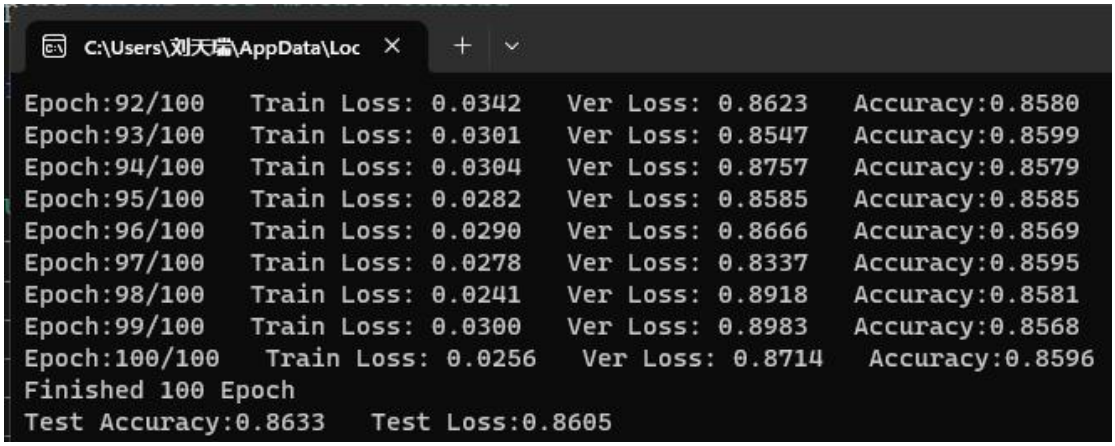
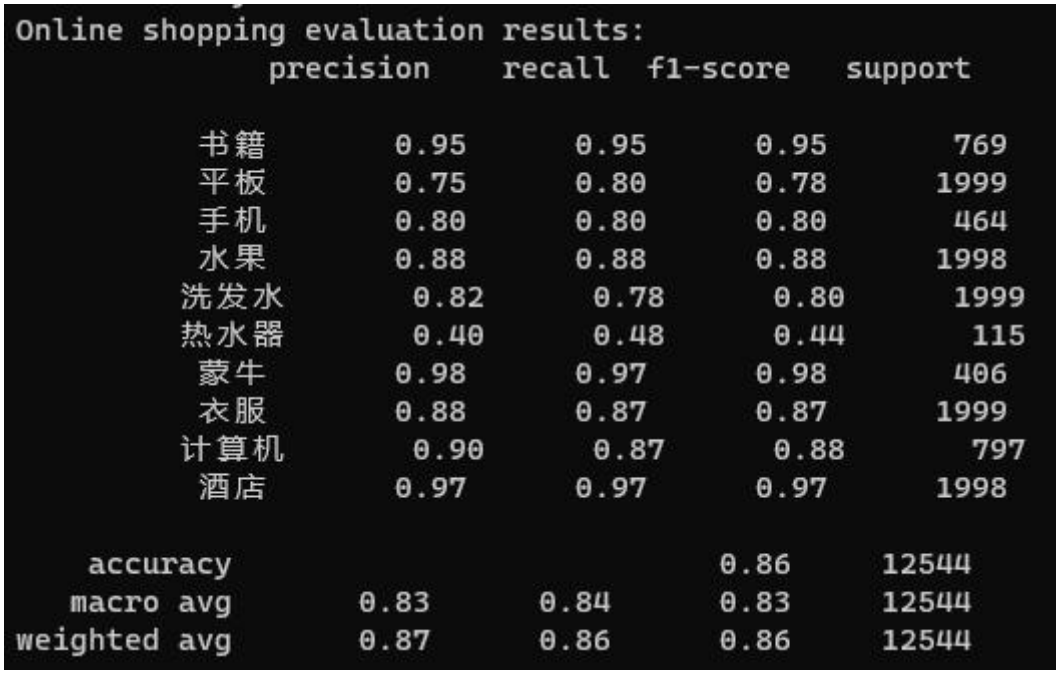


图 10



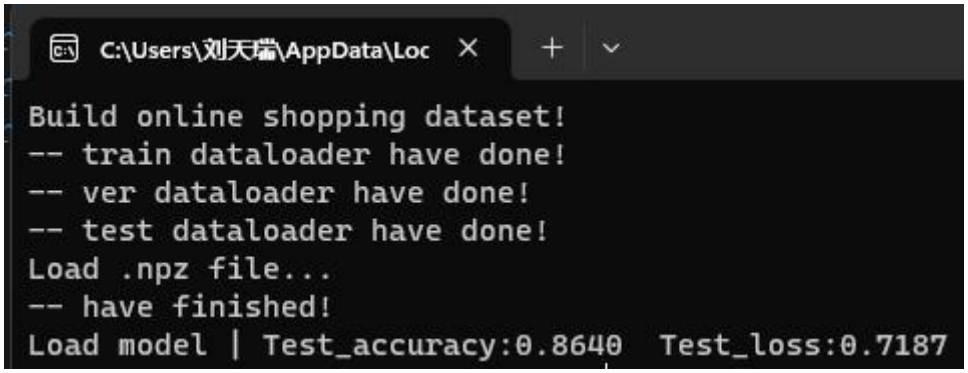


图 11

Bi-LSTM 网络结构模型训练与评测结果如下图 12 和图 13 所示：

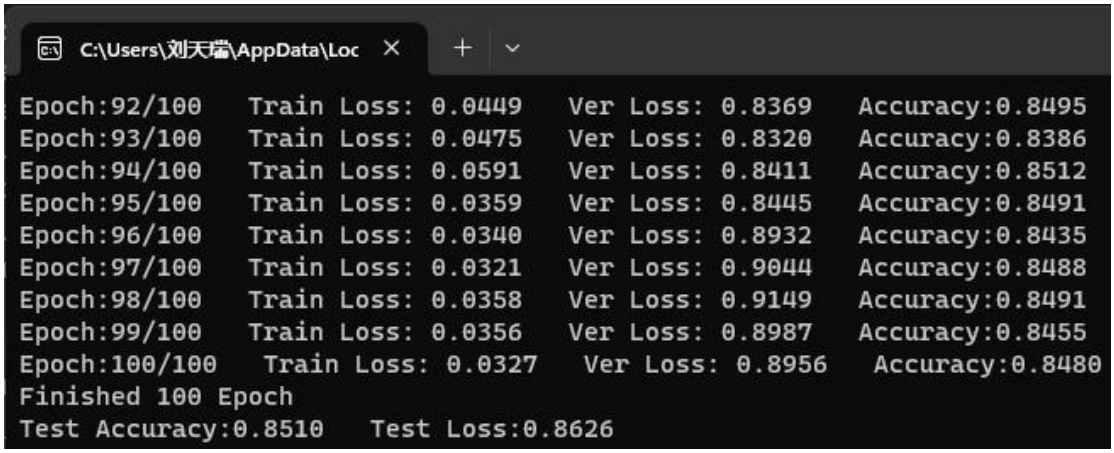
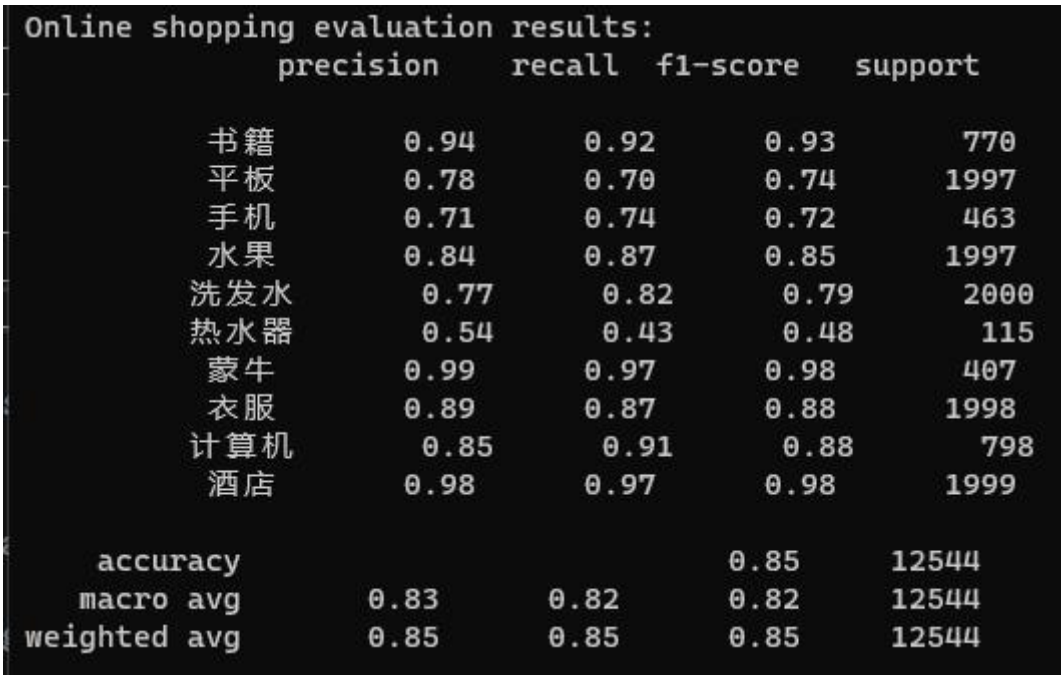


图 12



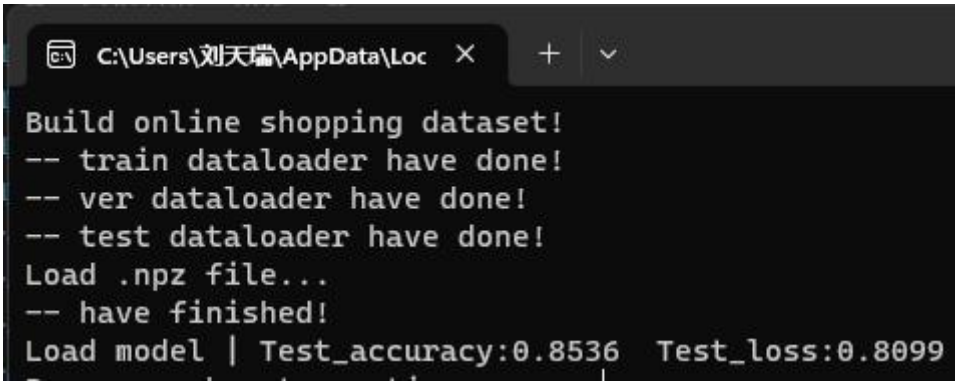


图 13

上述四个网络结构模型的实验结果如下表 1 所示：

	precision	**recall**	**f1-score**	**accuracy**
RNN	0.73	0.72	0.72	0.82
LSTM	0.83	0.84	0.83	0.86
GRU	0.83	0.82	0.82	0.86
Bi-LSTM	0.83	0.82	0.82	0.85

从实验结果便可以看出，LSTM，GRU，Bi-LSTM 网络结构模型的实验结果差距并不大，而 RNN 网络结构模型的训练效果相对较为明显地差于其他三者。除此此外，GRU 网络结构模型由于其结构较 LSTM 更为简单，故而简化了计算量，因此在运行时其运算速度最快。

4.2 温度预测任务实验结果

在开始训练之后，train_loss 和 test_loss 便迅速下降，而且训练速度也非常快，最终得到的实验结果如下图 14 所示：

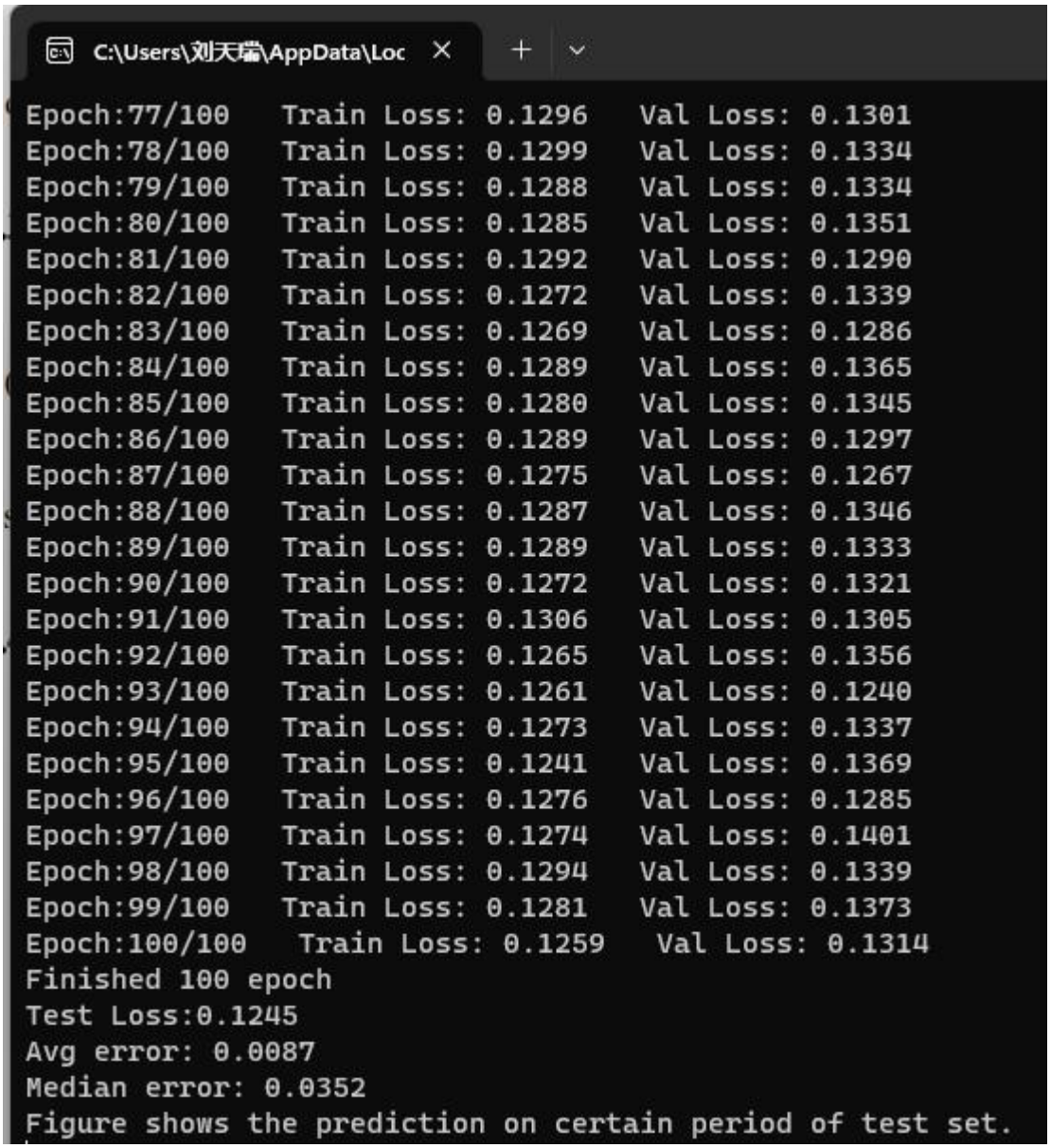


图 14

随机选取某一段的预测值与真实值进行 tensorboard 绘制曲线图，其显示结果

如下图 15 所示：

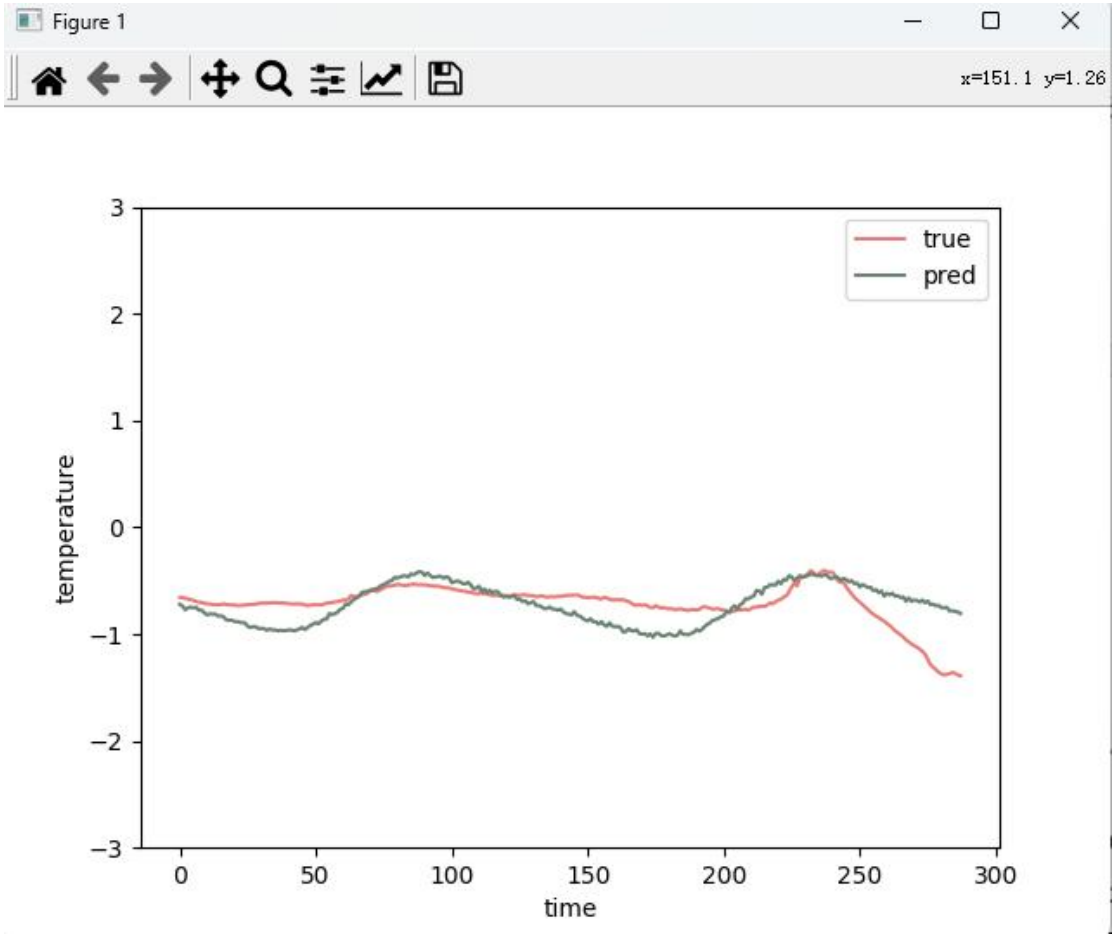


图 15

最后我说明一下模型和实验数据文档结构归类。在实验的当前文件夹下，其中 custom_dataset.py 程序是自定义的 online_shopping 和 jena_climate 数据集（需要自己手动下载）；Jena_Net.py 程序用于温度预测任务的网络结构模型；Online_Shopping_Net.py 程序则用于文本多分类任务的网络结构模型；RNN_series.py 程序是自己实现的循环神经网络结构模型；RNNcell_series.py 程序是另一个自己实现的循环神经网络单个单元的结构（各模块整合成上述的 RNN_series）；train_online_shopping.py 程序是文本多分类任务的训练文件；train_jena.py 则是温度预测任务的训练文件；除此之外，文件夹下还保存了训练程序中所用到的功能函数。

另外，./Data 目录下保存有 Online Shopping 以及 Jena Climate 数据集；./Model 目录下保存有已经训练好的网络结构模型（本次深度学习实验中我仅保留了最优模型）；./Runs_Online_Shopping 目录下保存有文本多分类任务的 tensorboard 日志文件；./Runs_Jena 目录下则保存有温度预测任务的 tensorboard 日志文件。

源代码以及注释见附件。