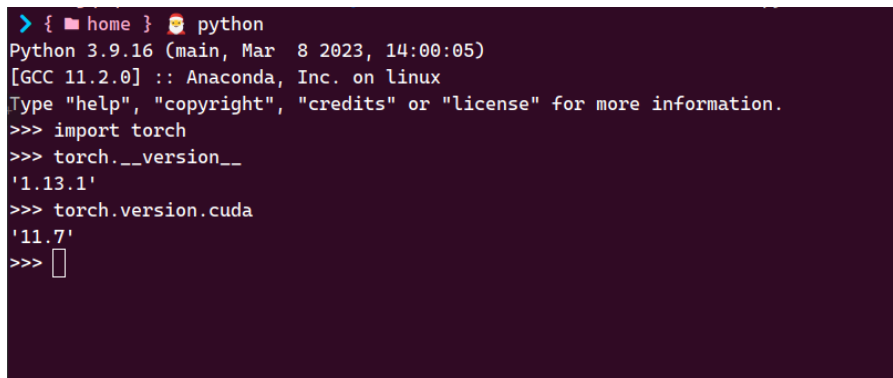


实验 4-循环神经网络实现

符兴 7203610316

1 实验环境

Ubuntu 20.04, Python 3.9, PyTorch1.13, Cuda11.7, TensorBoard;

A terminal window with a dark purple background. The prompt is '> { home } python'. The output shows 'Python 3.9.16 (main, Mar 8 2023, 14:00:05)', '[GCC 11.2.0] :: Anaconda, Inc. on linux', and a help message. The user enters '>>> import torch', '>>> torch.__version__', and '>>> torch.version.cuda'. The outputs are '1.13.1' and '11.7' respectively. The prompt '>>>' is followed by a cursor.

```
> { home } python
Python 3.9.16 (main, Mar 8 2023, 14:00:05)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.__version__
'1.13.1'
>>> torch.version.cuda
'11.7'
>>> 
```

图 1 Python 环境

2 实验过程

2.1 模型构建

在本次实验中，需要分别构建 RNN、GRU、LSTM 和 BiLSTM 网络的模型。

2.1.1 RNN 模型

在本次实验中，基于 `nn.Linear()` 搭建了 RNN 的网络结构；同时，基于 PyTorch 中自带的 `SequencePacked` 结构，实现了多个语句并行计算；具体的计算步骤为：

- (1) 使用 `nn.utils.rnn` 中 `pack_padded_sequence()` 工具，将多个语句按列进行合并，该函数提供按列合并后每个 `batch_size` 的长度以及内部语句的顺序；
- (2) 模型根据 `batch_size` 的长度从输入中读取数据，然后根据 `sorted_indices()` 提供的排列顺序迭代更新每一个时间步的 `hidden` 和 `output`；
- (3) 在得到结果后，经过 `classifier` 层得到最终的输出；

```

# 定义内部参数
# self.all_weight = []
# for _ in range(self.layer_num):
self.u = nn.Linear(input_size, hidden_size)
self.w = nn.Linear(hidden_size, hidden_size)
self.v = nn.Linear(hidden_size, hidden_size//2)
self.classifier = nn.Linear(hidden_size//2, output_size)
self.tanh = nn.Tanh()
self.sigmoid = nn.Sigmoid()

def forward(self, x):
    hidden = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
    output = torch.zeros((x.sorted_indices.shape[0], self.hidden_size//2)).to(self.device)
    index = 0
    i = 0

    while i < x.data.shape[0]:
        input = x.data[i:i+x.batch_sizes[index]]
        id = x.sorted_indices[:x.batch_sizes[index]]
        hidden[id] = self.u(input) + self.w(hidden[id])
        hidden[id] = self.tanh(hidden[id])
        output[id] = self.sigmoid(self.v(hidden[id]))
        i += x.batch_sizes[index]
        index += 1
    y = self.classifier(output)

    return y, hidden

```

图 2 RNN 模型

2.1.2 LSTM 模型

RNN 中会出现梯度消失的问题，即近处梯度作为主导而远处的梯度逐渐消失，因此简单的 RNN 很难建模长距离的依赖关系。在此基础上，人们通过引入门机制来控制信息的传播。

在 LSTM 中，第一步由遗忘门通过读取当前输入 x 和前神经元信息 h ，由 f_t 来决定从 Cell State 中丢弃什么信息，输出结果为 0~1 的值，表示保留该信息的程度；

第二步是确定 Cell State 所存放的新信息，sigmoid 层决定将要更新的值 i ；tanh 层创建一个新的候选值向量 \tilde{c}_t 加入到状态中。

第三步就是更新 Cell State，将 C_{t-1} 更新为 C_t 。把旧状态与 f_t 相乘，接着加上 $i_t * \tilde{c}_t$ 。

最后一步先通过 sigmoid 来确定 Cell State 的哪个部分将输出出去。然后将 Cell State 进行 tanh 处理，得到一个在 -1 到 1 之间的值，并将它和 sigmoid 的输出相乘，得到最终的输出。

除此之外，LSTM 还有一个双向的版本 BiLSTM，BiLSTM 由两个 LSTM 组合而成，一个是正向去处理输入序列，另一个反向处理序列，处理完成后将两个 LSTM 的输出拼接起来。

```

def forward(self, x):
    hidden = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
    c = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
    output = torch.zeros((x.sorted_indices.shape[0], self.hidden_size//2)).to(self.device)
    index = 0
    i = 0

    while i < x.data.shape[0]:
        input = x.data[i:i+x.batch_sizes[index]]
        id = x.sorted_indices[:x.batch_sizes[index]]
        combined = torch.cat((input, hidden[id]), dim=1)
        f_t = self.sigmoid(self.forget_gate(combined))
        i_t = self.sigmoid(self.input_gate(combined))
        c_hat = self.tanh(self.c_gate(combined))
        c[id] = f_t*c[id]+i_t*c_hat
        output[id] = self.sigmoid(self.output_gate(combined))
        hidden[id] = output[id]*self.tanh(c[id])
        i += x.batch_sizes[index]
        index += 1
    y = self.classifier(hidden)

```

图 3 LSTM 模型

```

# 正向
hidden = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
c = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
output = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
index = 0
i = 0

while i < x.data.shape[0]:
    input = x.data[i:i+x.batch_sizes[index]]
    id = x.sorted_indices[:x.batch_sizes[index]]
    combined = torch.cat((input, hidden[id]), dim=1)
    f_t = self.sigmoid(self.forget_gate(combined))
    i_t = self.sigmoid(self.input_gate(combined))
    c_hat = self.tanh(self.c_gate(combined))
    c[id] = f_t*c[id]+i_t*c_hat
    output[id] = self.sigmoid(self.output_gate(combined))
    hidden[id] = output[id]*self.tanh(c[id])
    i += x.batch_sizes[index]
    index += 1

# 反向
hidden_ = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
c_ = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
output_ = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
index = len(x.batch_sizes)-1
i = len(x.data)

while i > 0:
    input = x.data[i-x.batch_sizes[index]:i]
    id = x.sorted_indices[x.batch_sizes[index]:]
    combined = torch.cat((input, hidden_[id]), dim=1)
    f_t = self.sigmoid(self.forget_gate(combined))
    i_t = self.sigmoid(self.input_gate(combined))
    c_hat = self.tanh(self.c_gate(combined))
    c_[id] = f_t*c_[id]+i_t*c_hat
    output_[id] = self.sigmoid(self.output_gate(combined))
    hidden_[id] = output_[id]*self.tanh(c_[id])
    i -= x.batch_sizes[index]
    index -= 1
o = torch.cat((hidden, hidden_), dim=1)
y = self.classifier(o)

return y, o

```

图 4 BiLSTM 模型

2.1.3 GRU 模型

```
# 定义内部参数
self.reset_gate = nn.Linear(input_size + hidden_size, hidden_size)
self.update_gate = nn.Linear(input_size + hidden_size, hidden_size)
self.h_gate = nn.Linear(input_size + hidden_size, hidden_size)
self.output_gate = nn.Linear(hidden_size, hidden_size//2)
self.classifier = nn.Linear(hidden_size//2, output_size)
self.tanh = nn.Tanh()
self.sigmoid = nn.Sigmoid()

def forward(self, x):
    hidden = torch.zeros((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
    output = torch.zeros((x.sorted_indices.shape[0], self.hidden_size//2)).to(self.device)
    ones = torch.ones((x.sorted_indices.shape[0], self.hidden_size)).to(self.device)
    index = 0
    i = 0

    while i < x.data.shape[0]:
        input = x.data[i:x.batch_sizes[index]]
        id = x.sorted_indices[:x.batch_sizes[index]]
        compined = torch.cat((input, hidden[id]), dim=1)
        z = self.sigmoid(self.update_gate(compined))
        r = self.sigmoid(self.reset_gate(compined))
        compined_ = torch.cat((input, r*hidden[id]), dim=1)
        h_hat = self.tanh(self.h_gate(compined_))
        hidden[id] = (ones[id]-z)*hidden[id] + z*h_hat
        output[id] = self.sigmoid(self.output_gate(hidden[id]))
        i += x.batch_sizes[index]
        index += 1
    y = self.classifier(output)
```

图 5 GRU 模型

GRU 作为 LSTM 的一种变体，将遗忘门和输入门合成为一个更新门，同时还混合了 Cell State 和 Hidden State，最终的模型比标准的 LSTM 模型要简单，参数量更小，收敛速度更快。

GRU 首先通过更新门控制当前状态 h_t 需要从上一时刻状态 h_{t-1} 中保留多少信息，以及从候选状态 \tilde{h}_t 中接受多少信息；GRU 通过重置门控制候选状态 \tilde{h}_t 的计算。

2.2 任务

2.2.1 文本分类

在本次实验中，数据集是十种物品的网购评论数据；其中共 62774 条数据，按给定顺序编号 $i=1,2,\dots,62774$ ， $i\%5=4$ 做验证集， $i\%5=0$ 做测试集，其余作为训练集。同时，需要将文本信息转化为计算机能够理解的信息，在这里，使用 Word2Vec 模型获取词向量，具体步骤为：

- (1) 使用 jieba 对文本分词
- (2) 使用 gensim 的 Word2Vec 模型对步骤(1)中所构建的语料库进行训练；在

训练之前，每个句子首尾分别加上”<s>”和”</s>” 标签；训练参数：窗口长度 5，最少出现次数为 1，负采样数为 5，模型为 CBOW，线程数为 10；

(3) 在获取文本向量时按照分词结构依次读取 Word2Vec 中所对应的词向量即可。

(4) 由于每个句子长度不一致，还需要在句尾进行截断或补齐操作；

2.2.2 温度预测

在本次实验中，使用 jena_climate_2009_2016 数据集，其记录了 2009 至 2016 年每 10 分钟一次的气压、气温、风速等天气数据，总共 420551 条记录；

将其分为两个部分，2009 年-2014 年的数据作为训练集，2015-2016 年的数据作为测试集；除此之外，在对数据进行归一化操作后，每五天的天气数据作为输入，其中包括['hour', 'T (degC)', 'H2OC (mmol/mol)', 'rho (g/m**3)', 'sh (g/kg)', 'Tpot (K)', 'VPmax (mbar)']；五天后紧接两天的气温数据作为输出，即输入的维度为(5*144,7)，输出的维度为(1,288)；

除此之外，在构建训练集时使用滑动窗口的方式，比如 1、2、3、4、5 天的数据作为输入，6、7 天的气温作为输出；2、3、4、5、6 天的数据作为输入，7、8 天的气温作为输出。

在训练的过程中，使用 nn.MSELoss() 计算真实温度和预测温度的误差，并使用 adamW() 优化器进行优化。

3 实验结果

3.1 文本分类结果

(1) RNN 网络

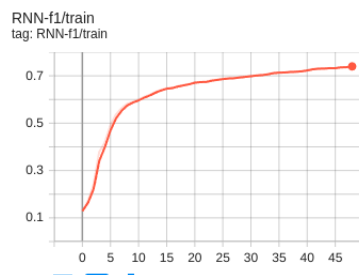


图 6-a 训练集 F1

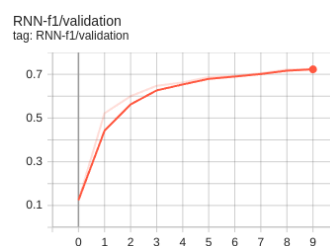


图 6-b 验证集 F1

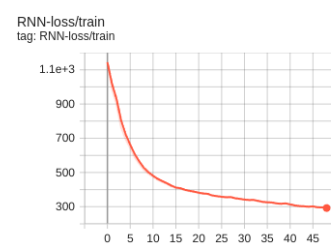


图 6-c 训练集 loss

```

root@autodl-container-d8bc11a952-9ed3a721:~/autodl-tmp/lab4# python main.py --task DModel --mode test --model RNN --pt ./pt/DModel_1685536886_RNN_testACC_0.725.pt
{'书籍': 0, '平板': 1, '手机': 2, '水果': 3, '洗发水': 4, '热水器': 5, '蒙牛': 6, '衣服': 7, '计算机': 8, '酒店': 9} {0: '书籍', 1: '平板', 2: '手机', 3: '水果', 4: '洗发水', 5: '热水器', 6: '蒙牛', 7: '衣服', 8: '计算机', 9: '酒店'}
100% | 393/393 [00:09<00:00, 40.73it/s]
类别: 书籍      精确率:0.9082 召回率:0.8718 F1:0.8896
类别: 平板      精确率:0.7261 召回率:0.7221 F1:0.7241
类别: 手机      精确率:0.4722 召回率:0.3656 F1:0.4121
类别: 水果      精确率:0.8905 召回率:0.8451 F1:0.8672
类别: 洗发水    精确率:0.8104 召回率:0.7646 F1:0.7868
类别: 热水器    精确率:0.0000 召回率:0.0000 F1:0.0
类别: 蒙牛      精确率:0.9875 召回率:0.9706 F1:0.979
类别: 衣服      精确率:0.7678 召回率:0.9055 F1:0.831
类别: 计算机    精确率:0.6881 召回率:0.8173 F1:0.7471
类别: 酒店      精确率:0.9676 召回率:0.9550 F1:0.9613
测试集: F1=0.7198

```

图 7 测试集结果

(2) GRU 网络

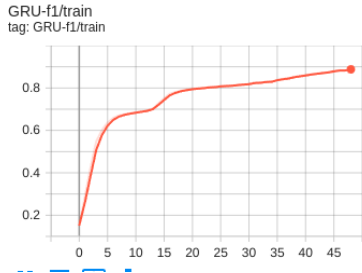


图 8-a 训练集 F1

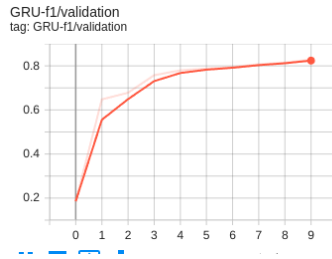


图 8-b 验证集 F1

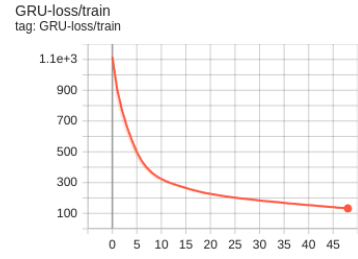


图 8-c 训练集 loss

```

root@autodl-container-d8bc11a952-9ed3a721:~/autodl-tmp/lab4# python main.py --task DModel --mode test --model GRU --pt ./pt/DModel_1685536826_GRU_testACC_0.828.pt
{'书籍': 0, '平板': 1, '手机': 2, '水果': 3, '洗发水': 4, '热水器': 5, '蒙牛': 6, '衣服': 7, '计算机': 8, '酒店': 9} {0: '书籍', 1: '平板', 2: '手机', 3: '水果', 4: '洗发水', 5: '热水器', 6: '蒙牛', 7: '衣服', 8: '计算机', 9: '酒店'}
100% | 393/393 [00:11<00:00, 32.76it/s]
类别: 书籍      精确率:0.9091 召回率:0.9197 F1:0.9144
类别: 平板      精确率:0.7793 召回率:0.8101 F1:0.7944
类别: 手机      精确率:0.8146 召回率:0.7656 F1:0.7894
类别: 水果      精确率:0.9131 召回率:0.8561 F1:0.8837
类别: 洗发水    精确率:0.7710 召回率:0.8536 F1:0.8106
类别: 热水器    精确率:0.5667 召回率:0.1478 F1:0.2345
类别: 蒙牛      精确率:0.9975 召回率:0.9853 F1:0.9914
类别: 衣服      精确率:0.8699 召回率:0.8791 F1:0.8745
类别: 计算机    精确率:0.8856 召回率:0.8335 F1:0.8588
类别: 酒店      精确率:0.9787 召回率:0.9635 F1:0.971
测试集: F1=0.8123
root@autodl-container-d8bc11a952-9ed3a721:~/autodl-tmp/lab4#

```

图 9 测试集结果

(3) LSTM 网络

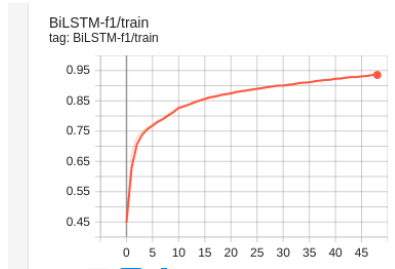


图 10-a 训练集 F1

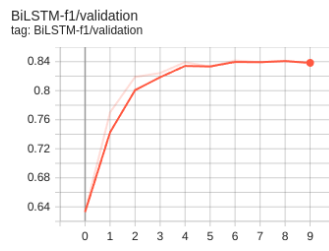


图 10-b 验证集 F1

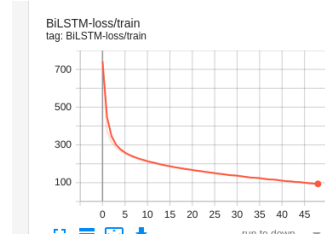


图 10-c 训练集 loss

```

root@autodl-container-d8bc11a952-9ed3a721:~/autodl-tmp/lab4# python main.py --task DModel --mode test --model LSTM --pt ./pt/DModel_1685538269_LSTM_testACC_0.831.pt
{'书籍': 0, '平板': 1, '手机': 2, '水果': 3, '洗发水': 4, '热水器': 5, '蒙牛': 6, '衣服': 7, '计算机': 8, '酒店': 9} {0: '书籍', 1: '平板', 2: '手机', 3: '水果', 4: '洗发水', 5: '热水器', 6: '蒙牛', 7: '衣服', 8: '计算机', 9: '酒店'}
100% | 393/393 [00:11<00:00, 34.65it/s]
类别: 书籍      精确率:0.9012 召回率:0.9339 F1:0.9173
类别: 平板      精确率:0.7604 召回率:0.7836 F1:0.7718
类别: 手机      精确率:0.8518 召回率:0.7290 F1:0.7856
类别: 水果      精确率:0.9072 召回率:0.8501 F1:0.8777
类别: 洗发水    精确率:0.7520 召回率:0.8366 F1:0.7921
类别: 热水器    精确率:0.5521 召回率:0.4609 F1:0.5024
类别: 蒙牛      精确率:0.9877 召回率:0.9853 F1:0.9865
类别: 衣服      精确率:0.8932 召回率:0.8361 F1:0.8637
类别: 计算机    精确率:0.8236 召回率:0.8608 F1:0.8437
类别: 酒店      精确率:0.9708 召回率:0.9645 F1:0.9677
测试集: F1=0.8308
root@autodl-container-d8bc11a952-9ed3a721:~/autodl-tmp/lab4#

```

图 11 测试集结果

(4) BiLSTM 网络

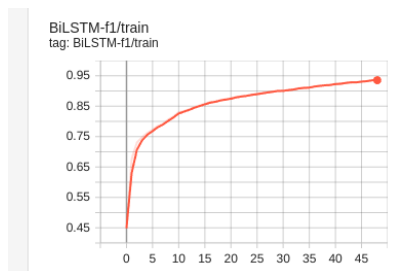


图 12-a 训练集 F1

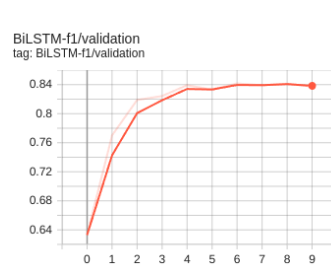


图 12-b 验证集 F1

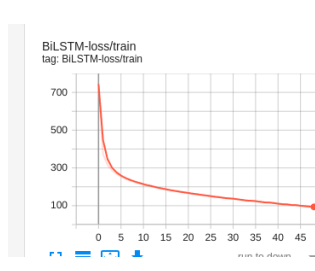


图 12-c 训练集 loss

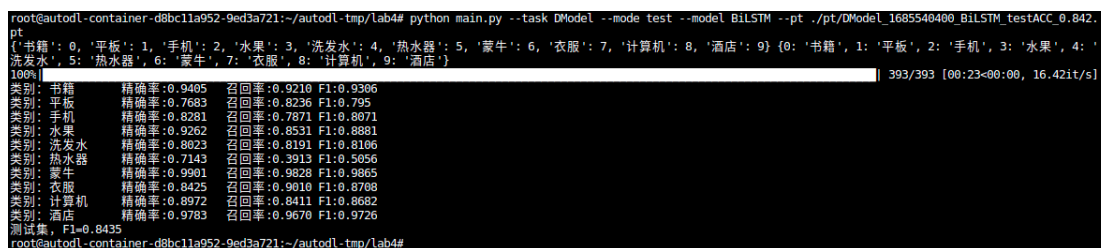


图 13 测试集结果

3.2 温度预测结果

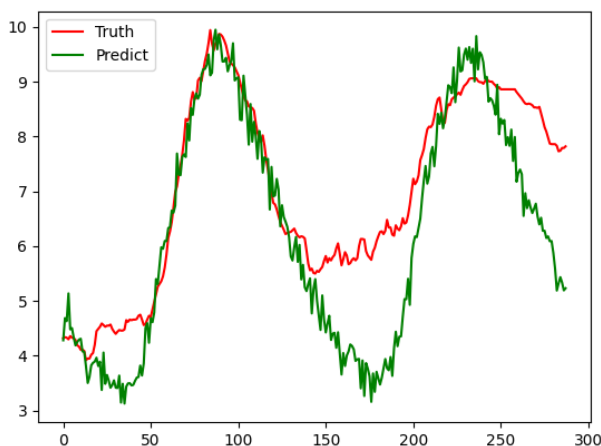
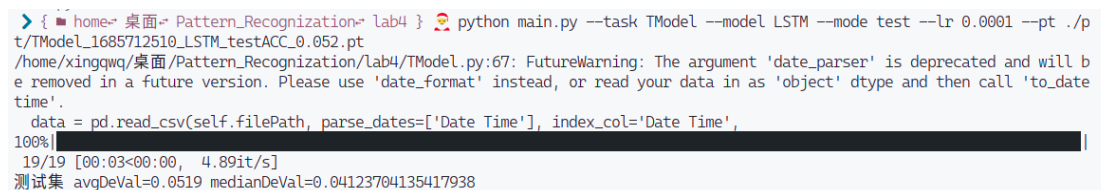


图 14 测试集结果

4 心得与体会

在本次实验中，具体掌握了 RNN、GRU、LSTM 和 BiLSTM 模型的具体结构，以及他们的优势与劣势；同时，通过使用 PyTorch 工具包可以使得模型在

`batch_size>1` 的情况下多个 `batch` 并行计算，提高了训练效率；除此之外，发现如果在模型中存储了每一个步长的 `hidden`，会导致模型无法收敛，但仍没有查出具体的原因，希望随着进一步的学习可以解释其原由。