



研究生《深度学习》课程 实验报告

实验名称:	循环神经网络
姓 名:	唐麒
学 号:	21120299
上课类型:	专业课
日 期:	2022 年 12 月 10 日

一、实验内容

本实验报告包括的实验有分别手写和 `torch.nn` 实现循环神经网络 RNN，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果，并在此基础上对不同的超参数进行对比分析（例如 `hidden_size`、`batchsize`、`lr` 等）。除此之外，还要使用 PyTorch 实现 LSTM 和 GRU 并在至少一个数据集进行试验分析，并设计实验，对比分析 LSTM 和 GRU 在相同数据集上的结果。通过本次实验，学生将对循环神经网络等的原理及实现有较为清晰的认识。

1.1 手动实现循环神经网络

手动实现循环神经网络 RNN，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验结果(最好 使用图表展示)。

1.2 使用 `torch.nn.rnn` 实现循环神经网络

使用 `torch.nn.rnn` 实现循环神经网络，并在至少一个数据集上进行实验，从训练时间、预测精度、Loss 变化等角度分析实验 结果（最好使用图表展示）。

1.3 超参数的对比分析

不同超参数的对比分析（包括 `hidden_size`、`batchsize`、`lr` 等）选其中至少 1-2 个进行分析。

1.4 实现 LSTM 和 GRU

（1）使用 PyTorch 实现 LSTM 和 GRU 并在至少一个数据集进行试验分析；

（2）设计实验，对比分析 LSTM 和 GRU 在相同数据集上的结果。

二、实验设计

本次实验所包含的内容皆为指定内容，涉及部分数据加载、模型及其损失函数和优化器的实现，故此部分省略，在报告的后续内容中进行介绍。

三、实验环境及实验数据集

实验所用的开发环境如表 1 所示。

表 1 实验环境

开发环境	版本
操作系统	macOS Big Sur 11.5.1
开发工具	Jupyter Notebook
Python	3.8.2
PyTorch	1.8.0 (CPU)

本次实验所涉及的数据集为高速公路车流量数据，用历史流量数据预测未来流量，即回归任务。

PeMS 是美国加利福尼亚州高速公路的实时车流量数据，由铺设在道路上的检测线圈采集。本实验中包含 04 地区的数据，储存在 PEMS04.npz 文件中。数据中的三个特征维度：车流量、拥挤程度和车速。

四、实验过程

本小节将对实验的具体内容进行详细说明。

首先对数据集划分训练集和测试集的代码进行介绍和展示。

```
1  def train_test_split(window_size):
2      data = np.load("../dataset/pems-traffic-flow/PEMS04.npz")
3      pem04 = data['data']
4      scaler_data = np.zeros((pem04.shape[0], pem04.shape[1], pem04.shape[2]))
5      v_scaler = MinMaxScaler() # 速度归一化
6      o_scaler = MinMaxScaler() # 拥挤程度归一化
7      f_scaler = MinMaxScaler() # 车流量归一化
8      scaler_data[:, :, 0] = f_scaler.fit_transform(pem04[:, :, 0]) # 车流量
9      scaler_data[:, :, 1] = o_scaler.fit_transform(pem04[:, :, 1]) # 拥挤程度
10     scaler_data[:, :, 2] = v_scaler.fit_transform(pem04[:, :, 2]) # 速度
11     ratio = int(pem04.shape[0] * 0.75)
12     train_data = scaler_data[:ratio, :, :]
13     test_data = scaler_data[ratio:, :, :]
14     # 训练集数据时间序列采样
15     result = []
16     for i in range(len(train_data) - window_size - 1):
```

```

17         tmp = train_data[i: i + window_size, :, :]
18         tmp = tmp.reshape(-1, 307 * 3)
19         # 后 1min 的数据作为 label
20         label = train_data[i + window_size + 1, :, :].reshape(1, -1)
21         tmp = np.concatenate((tmp, label), axis=0)
22         result.append(tmp)
23     train_loader = DataLoader(result, batch_size=30, shuffle=False)
24     test_sets = []
25     for i in range(len(test_data) - window_size - 1):
26         tmp = test_data[i: i + window_size, :, :]
27         tmp = tmp.reshape(-1, 307 * 3)
28         # 后 1min 的数据作为 label
29         label = test_data[i + window_size + 1, :, :].reshape(1, -1)
30         tmp = np.concatenate((tmp, label), axis=0)
31         test_sets.append(tmp)
32     test_loader = DataLoader(test_sets, batch_size=36, shuffle=False)
33     # 返回 MinMaxScaler 以便反归一化
34     return train_loader, test_loader, v_scaler, f_scaler, o_scaler

```

为了利用图表对训练损失曲线和测试损失曲线进行展示, 通过将损失值和预测准确率存储来进行曲线图像的绘制。

```

1     #绘制 loss 曲线变化图
2     epoches = np.arange(1, epoch + 1)
3     train_loss_list = np.array(train_loss_list)
4     test_loss_list = np.array(test_loss_list)
5     plt.plot(epoches, train_loss_list, label = "train")
6     plt.plot(epoches, test_loss_list, label = "test")
7     plt.legend()
8     plt.plot()
9
10    def show_preds(batch, test_preds, test_label, scaler, type, no):
11        # 绘制第 no 个探头的速度、车流量、拥挤程度的对比
12        # 获取前 batch 的预测数据
13        p = torch.stack(test_preds[:batch], dim=0).reshape(-1, 307, 3)
14        t = torch.stack(test_label[:batch], dim=0).reshape(-1, 307, 3)
15        predict = scaler.inverse_transform(p[:, :, type].detach().numpy())
16        labels = scaler.inverse_transform(t[:, :, type].detach().numpy())
17        no_predict = predict[:, no]

```

```

18     no_labels = labels[:, no]
19     x = np.arange(1, p.shape[0] + 1)
20     plt.plot(x, no_predict, label='predict')
21     plt.plot(x, no_labels, label="original")
22     plt.legend()
23     plt.show()

```

在下文的介绍中，将重点介绍与实验探究目的相关的代码，数据加载、模型训练等基本骨架代码将不再进行详细展示。

4.1 手动实现循环神经网络

```

1  class RNN(nn.Module):
2      def __init__(self, input_size, hidden_size, output_size):
3          super().__init__()
4          self.hidden_size = hidden_size
5          self.i_h = nn.Parameter(torch.normal(0, 0.01, (input_size, hidden_size)),
6                                   requires_grad=True)
7          self.h_h = nn.Parameter(torch.normal(0, 0.01, (hidden_size, hidden_size)),
8                                   requires_grad=True)
9          self.h_o = nn.Parameter(torch.normal(0, 0.01, (hidden_size, output_size)),
10                                  requires_grad=True)
11         self.b_i = nn.Parameter(torch.zeros(hidden_size), requires_grad=True)
12         self.b_o = nn.Parameter(torch.zeros(output_size), requires_grad=True)
13         self.tanh = nn.Tanh()
14         self.leaky_relu = nn.LeakyReLU()
15     def forward(self, x):
16         batch_size = x.shape[1]
17         seq_len = x.shape[0]
18         h = torch.normal(0, 0.01, (batch_size, self.hidden_size))
19         outputs = []
20         for i in range(seq_len):
21             h = torch.matmul(x[i, :, :], self.i_h) + torch.matmul(h, self.h_h) + self.b_i
22             h = self.tanh(h)
23             o = self.leaky_relu(torch.matmul(h, self.h_o) + self.b_o)
24             outputs.append(o)
25         return torch.stack(outputs, dim=0), h

```

4.2 使用 torch.nn 实现循环神经网络

```
1 class RNNModel(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size):
3         super().__init__()
4         self.rnn = nn.RNN(input_size=input_size, hidden_size=hidden_size)
5         self.output = nn.Linear(hidden_size, output_size)
6     def forward(self, x):
7         _, hidden = self.rnn(x)
8         out = self.output(hidden)
9         return out
```

4.3 实现 LSTM 和 GRU

```
1 class LSTMModel(nn.Module):
2     def __init__(self, input_size, hidden_size, output_size):
3         super().__init__()
4         self.hidden_size = hidden_size
5         self.lstm = nn.LSTM(input_size=input_size, hidden_size=hidden_size)
6         self.out = nn.Linear(hidden_size, output_size)
7     def forward(self, x):
8         _, (hid, cell) = self.lstm(x)
9         output = self.out(hid)
10        return output
11
12 class GRUModel(nn.Module):
13     def __init__(self, input_size, hidden_size, output_size):
14         super().__init__()
15         self.hidden_size = hidden_size
16         self.gru = nn.GRU(input_size=input_size, hidden_size=hidden_size)
17         self.out = nn.Linear(hidden_size, output_size)
18     def forward(self, x):
19         _, hid = self.gru(x)
20         output = self.out(hid)
21         return output
```

五、实验结果

5.1 手动实现循环神经网络

在该实验中，模型训练了 20 个 epoch，模型在训练集和测试集上的 loss 曲线如图 1 所示，均呈下降趋势。

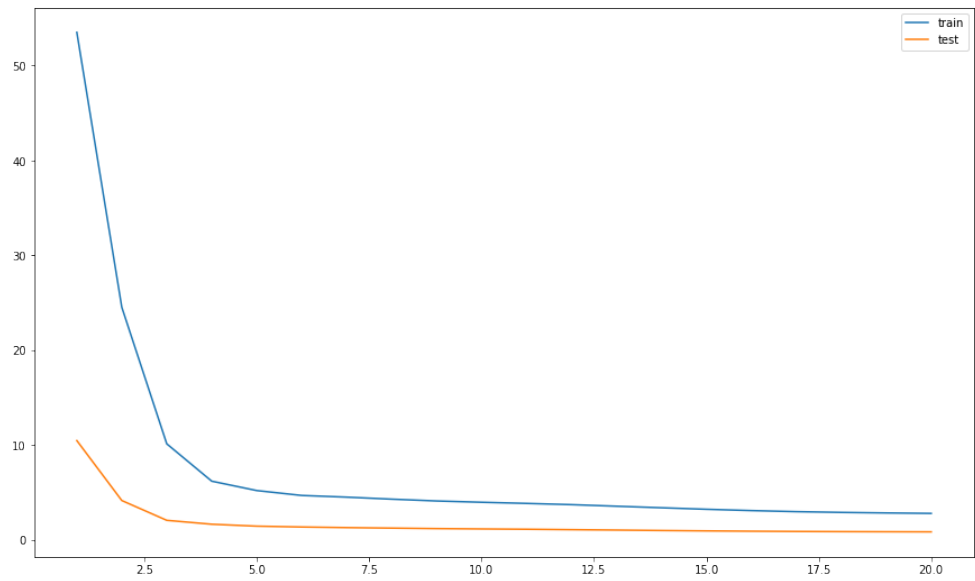


图 1 手动实现循环神经网络损失函数曲线图

模型在在测试集上测试模型效果如图 2 所示。



图 2 手动实现卷积神经网络预测准确率曲线图

训练 20 个 epoch 共用时约 100.20 秒。

5.2 使用 torch.nn.rnn 实现循环神经网络

在该实验中，模型训练了 20 个 epoch，模型在训练集和测试集上的 loss 曲线如图 3 所示，均呈下降趋势。

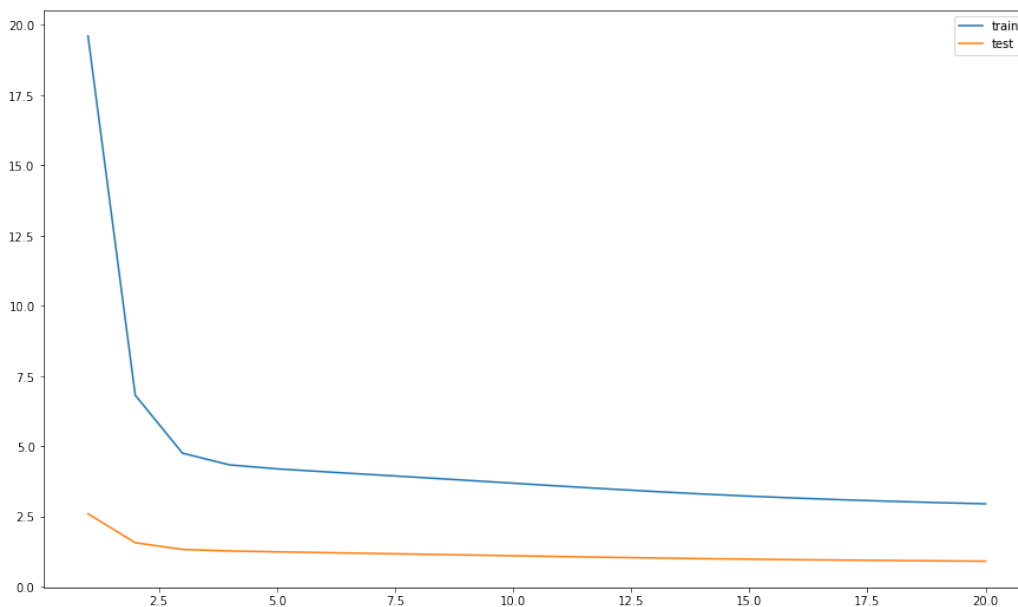


图 3 torch.nn.rnn 实现循环神经网络损失函数曲线图

模型在在测试集上测试模型效果如图 4 所示。

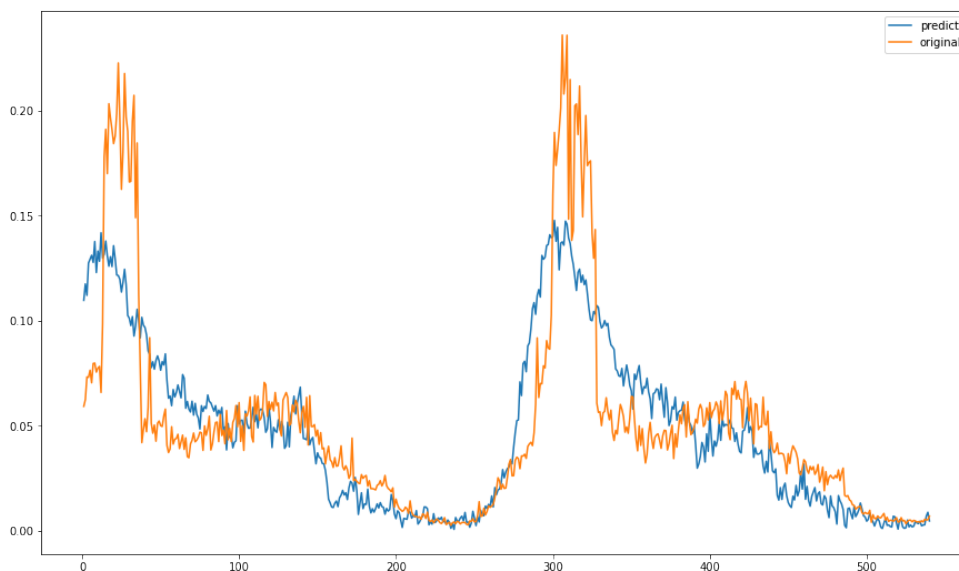


图 4 torch.nn.rnn 实现循环神经网络预测准确率曲线图

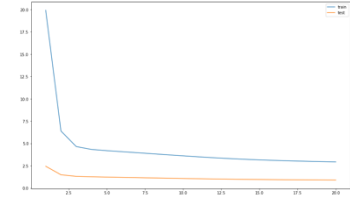
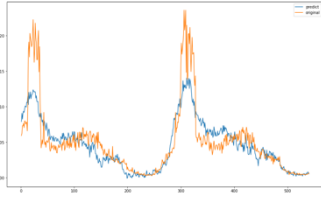
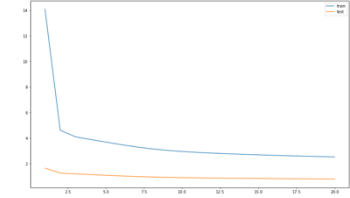
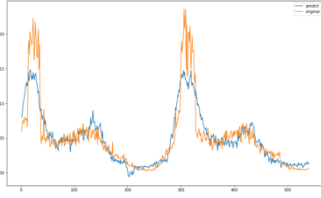
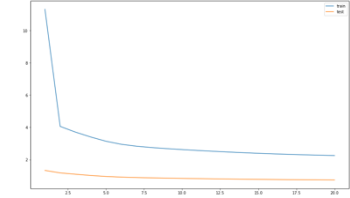
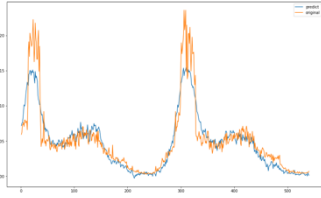
torch.nn.rnn 实现的循环神经网络训练 20 个 epoch 所需时间约为 51.81 秒，比手动实现的方式要快。

5.3 不同超参数对比分析

1) hidden_size

可以看到，hidden_size 的增加，可以提升预测的准确率，但同时也会造成计算开销的增加，从而导致训练时间的增长。

表 2 循环神经网络不同 hidden_size 结果对比

hidden_size	损失函数曲线图	预测准确率曲线图	Times(s)
256			50.77
512			114.23
1024			339.90

2) 学习率

可以看到，过小的学习率会收敛较慢，在相同的 epoch 下，最终损失较大，对训练时间影响不大。

表 3 循环神经网络不同学习率结果对比

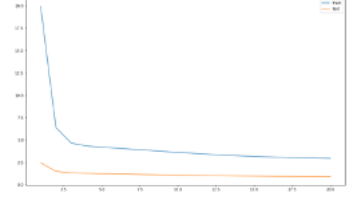
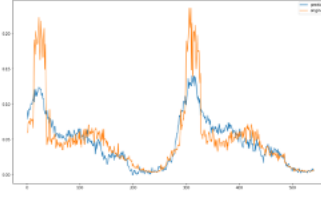
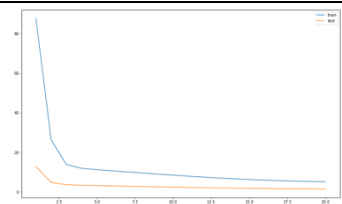
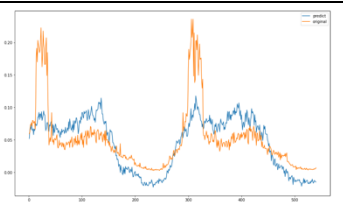
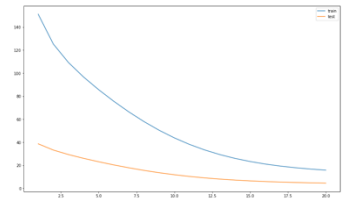
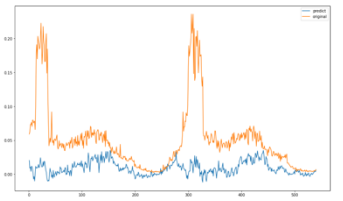
学习率	损失函数曲线图	预测准确率曲线图	Times(s)
1e-1			50.77

表 4 循环神经网络不同学习率结果对比。

(续)

学习率	损失函数曲线图	预测准确率曲线图	Times(s)
1e-2			51.67
1e-3			51.87

5.4 实现 LSTM 和 GRU

(1) 使用 PyTorch 实现 LSTM 和 GRU 并在至少一个数据集进行试验分析;

在该实验中, LSTM 模型训练了 20 个 epoch, 模型在训练集和测试集上的 loss 曲线如图 5 所示, 均呈下降趋势。

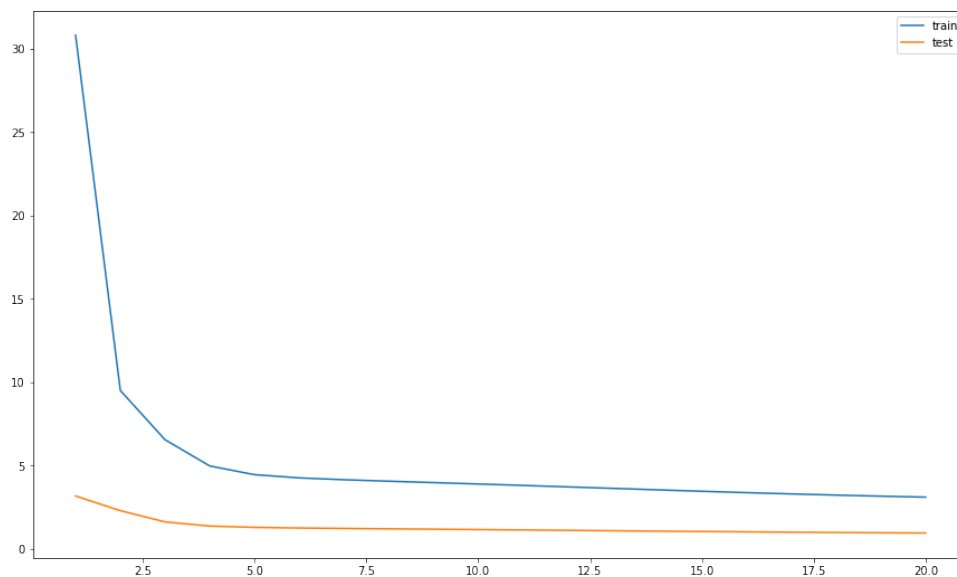


图 5 PyTorch 实现 LSTM 损失函数曲线图

模型在在测试集上测试模型效果如图 6 所示。

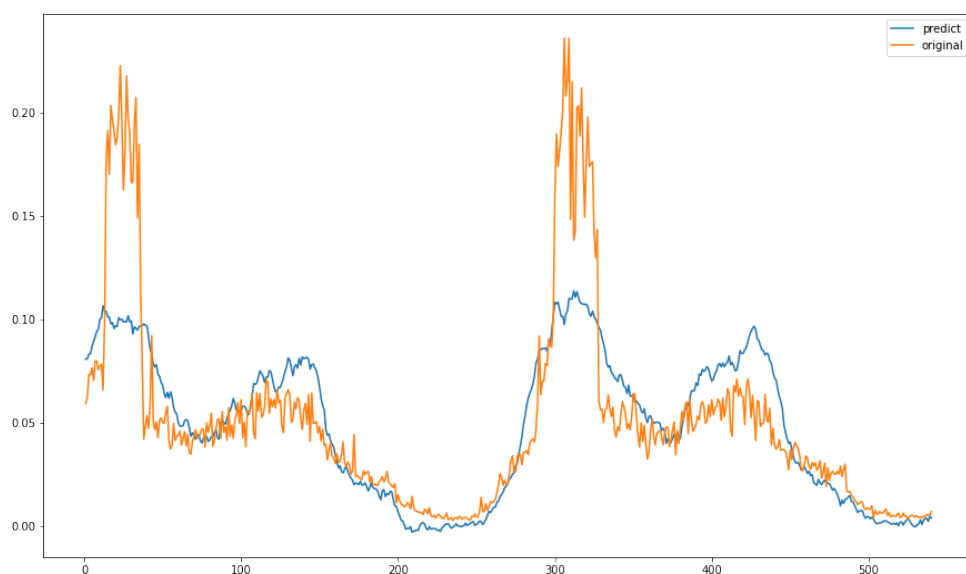


图 6 PyTorch 实现 LSTM 预测准确率曲线图

训练 20 个 epoch 所需时间约为 150.47 秒。

在该实验中，GRU 模型训练了 20 个 epoch，模型在训练集和测试集上的 loss 曲线如图 7 所示，均呈下降趋势。

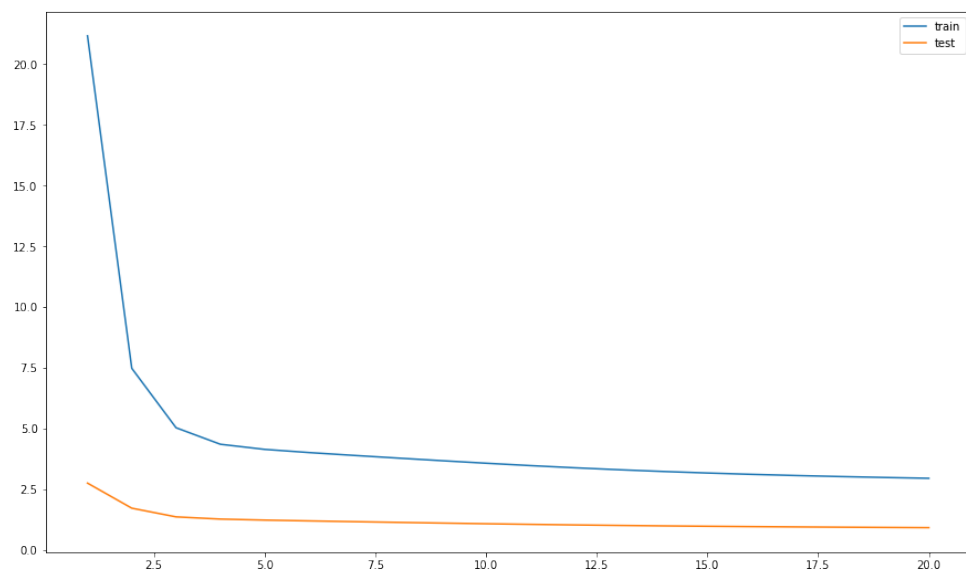


图 7 PyTorch 实现 GRU 损失函数曲线图

模型在在测试集上测试模型效果如图 8 所示。

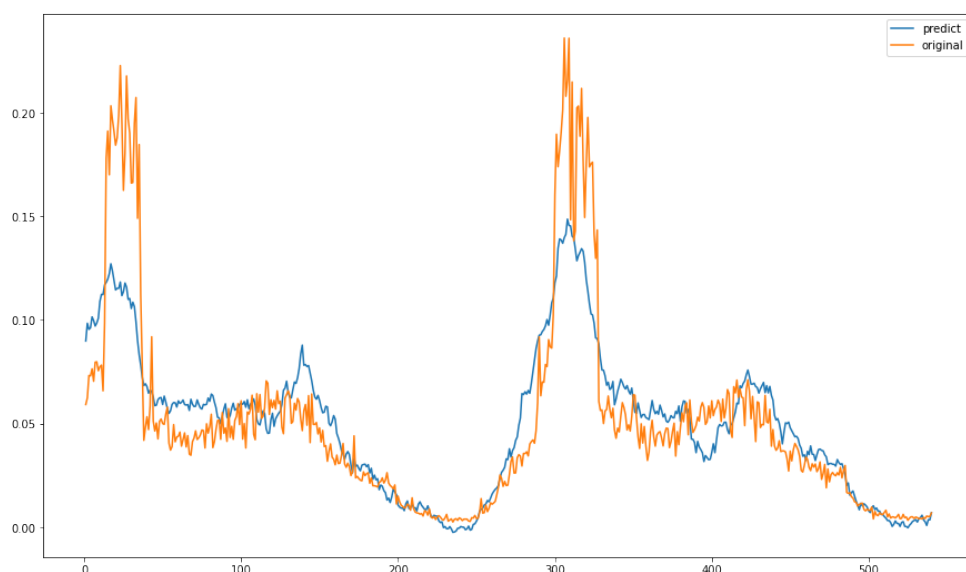


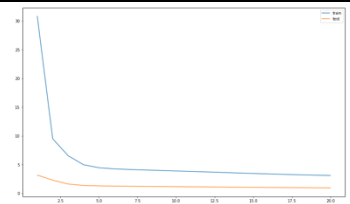
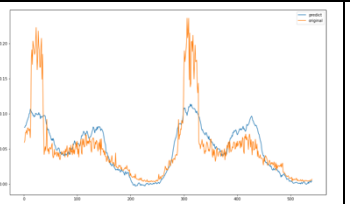
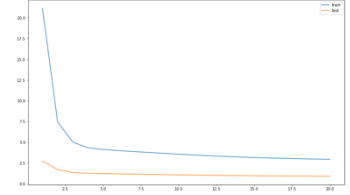
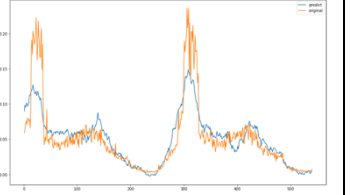
图 8 PyTorch 实现 GRU 预测准确率曲线图

训练 20 个 epoch 所需时间约为 121.17 秒。

(2) 设计实验，对比分析 LSTM 和 GRU 在相同数据集上的结果。

在保证模型层数等其他超参数相同（hidden_size=256, lr=0.1, epoch=20, window_size=6）的情况下，对 LSTM 和 GRU 的结果进行对比。

表 5 前馈神经网络与卷积神经网络结果对比

网络	损失函数曲线图	预测准确率曲线图	Test Loss	Times(s)
LSTM			0.9380	150.47
GRU			0.9104	121.17

可以看到，相较于 LSTM 而言，GRU 的性能更高，且训练时间也相对较短。

六、实验心得体会

通过本次实验，对循环神经网络的原理和实现有了更扎实的理解和掌握，弥补了我在之前科研任务中依赖卷积神经网络而对循环神经网络等模型的知识欠缺。

在实验报告撰写的过程中，可以对实验的目的、问题和解决方案进行明确的梳理，进一步在编码实验的基础上获得了提升，做到了理论和实践的齐头并进。

七、参考文献

- [1] 北京交通大学 《深度学习》课程组，实验 4 循环神经网络实验，北京交通大学《深度学习》课件

八、附录

无