

研究生《深度学习》课程

实验报告

|  |  |
| --- | --- |
| **实验名称：** | **循环神经网络实验** |
| **姓 名：** | **束天成** |
| **学 号：** | **22121501** |
| **日 期：** | **2022/8/19** |

## 一、实验内容

本次实验为循环神经网络的基本实验，本次实验主要包含了循环神经网络实验，既包含了循环神经网络的定义和应用，也包含了使用面向对象的方式调用循环神经网络，编程的语言为Python，其目的是了解和掌握循环神经网络的基本原理，了解面向对象的实验方法，并将其运用到实际操作中。

#### 1.1 循环神经网络基本实验

循环神经网络的基本数据处理实验主要是为了了解理解序列数据处理方法，学会使用面向对象编程中的缺失代码，并使用torch自带数据工具将数据封装为dataloader，以便于后面实验完成调用。

#### 1.2 实现RNN、LSTM和GRU

循环神经网络的本质也是调用一系列的基本卷积神经网络基本模型从而实现对模型的处理。本次实验要求分别采用手动方式以及调用接口方式实现RNN、LSTM和GRU，并在至少一种数据集上进行实验，通过对比实验各种结果数据，验证循环神经网络的可行性。

#### 1.3 分析对比实验

## 从训练时间、预测精度、Loss 变化等角度分析实验结果，并通过设置不同的参数值，分析对比不同的参数对实验模型结果的影响。

## 二、实验设计

本次实验主要内容都为指定内容，均指定了实验需要的数据矩阵维度，矩阵构造方式，运算函数，使用的方法以及需要分析的内容，不包含过多需要自主设计的模型和方法等，因此不再赘述。

## 三、实验环境及实验数据集

Torch版本：1.5.1

操作系统：Windows10

硬件：

CPU：i5 9300h GPU：GTX1650

数据集：torch.nn，Fashion-MNIST数据集，nn.BCEWithLogitsLoss函数等

## 四、实验过程

实验过程包括在编写代码时一些需要注意的事项，可附代码片段进行说明；还应该包括在训练模型时进行的一些步骤、参数设置等内容。

注：为了代码片段尽量的美观、统一，建议附代码片段时只附加关键的片段，不要全部粘贴，并尽量使用下面提供的网站进行代码高亮等格式转换后再粘贴。

<http://www.planetb.ca/syntax-highlight-word>

#### 4.1 循环神经网络数据预处理

**4.1.1**理解序列数据处理方法，补全面向对象编程中的缺失代码，并使用torch

自带数据工具将数据封装为dataloader

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.utils.data as data
5. import warnings
6. from torch.utils.data import DataLoader,Dataset
7. warnings.filterwarnings("ignore")
8. import random
9. from IPython import display
10. from matplotlib import pyplot as plt
11. from PIL import Image
12. import os
13. from torch import nn
14. import torch.optim as optim
15. from torch.nn import init
16. import torch.nn.functional as F
17. import time
18. import pandas as pd
19. from sklearn.utils import shuffle
20. import math
21. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
22. import datasets
23. import utils
24. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
25. def mape\_fn(y, pred):
26. mask = y != 0
27. y = y[mask]
28. pred = pred[mask]
29. mape = np.abs((y - pred) / y)
30. mape = np.mean(mape) \* 100
31. return mape
32. def eval(y, pred):
33. y = y.cpu().numpy()
34. pred = pred.cpu().numpy()
35. mse = mse\_fn(y, pred)
36. rmse = math.sqrt(mse)
37. mae = mae\_fn(y, pred)
38. mape = mape\_fn(y, pred)
39. return [rmse, mae, mape]
40. *# 定义dataset*
41. class my\_Dataset(data.Dataset):
42. def \_\_init\_\_(self, features, labels):
43. self.X = features
44. self.y = labels
45. def \_\_getitem\_\_(self, index):
46. return self.X[index], self.y[index]
47. def \_\_len\_\_(self):
48. return self.X.shape[0]
50. class TrafficDataset:
51. def \_\_init\_\_(self):
52. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
53. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
54. *# 数据标准化*
55. self.min = self.raw\_data.min()
56. self.max = self.raw\_data.max()
57. self.data = (self.raw\_data - self.min) / (self.max - self.min)
58. def denormalize(self, x):
59. return x \* (self.max - self.min) + self.min
60. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
61. train\_x = []
62. train\_y = []
63. val\_x = []
64. val\_y = []
65. test\_x = []
66. test\_y = []
67. window\_size = 12
68. len\_train = int(self.data.shape[0] \* 0.6)
69. train\_seqs = self.data[:len\_train]
70. for i in range(train\_seqs.shape[0] - window\_size):
71. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
72. train\_y.append(train\_seqs[i + window\_size].squeeze())
73. len\_val = int(self.data.shape[0] \* 0.8)
74. val\_seqs = self.data[len\_train:len\_val]
75. for i in range(val\_seqs.shape[0] - window\_size):
76. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
77. val\_y.append(train\_seqs[i + window\_size].squeeze())
78. test\_seqs = self.data[len\_val:]
79. for i in range(test\_seqs.shape[0] - window\_size):
80. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
81. test\_y.append(test\_seqs[i + window\_size].squeeze())
82. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
83. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
84. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
85. return train\_set, val\_set, test\_set
86. *#补全相关的代码*
87. batch\_size = 64
88. TrafficData = TrafficDataset()
89. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
90. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
91. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. print(train\_loader)
94. print(val\_loader)
95. print(test\_loader)

#### 4.2 采用手动方式以及调用接口方式实现RNN、LSTM和GRU

**4.2.1** 手动实现RNN

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.utils.data as data
5. import warnings
6. from torch.utils.data import DataLoader,Dataset
7. warnings.filterwarnings("ignore")
8. import random
9. from IPython import display
10. from matplotlib import pyplot as plt
11. from PIL import Image
12. import os
13. from torch import nn
14. import torch.optim as optim
15. from torch.nn import init
16. import torch.nn.functional as F
17. import time
18. import pandas as pd
19. from sklearn.utils import shuffle
20. import math
21. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
22. import datasets
23. import utils
24. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
25. def mape\_fn(y, pred):
26. mask = y != 0
27. y = y[mask]
28. pred = pred[mask]
29. mape = np.abs((y - pred) / y)
30. mape = np.mean(mape) \* 100
31. return mape
32. def eval(y, pred):
33. y = y.cpu().numpy()
34. pred = pred.cpu().numpy()
35. mse = mse\_fn(y, pred)
36. rmse = math.sqrt(mse)
37. mae = mae\_fn(y, pred)
38. mape = mape\_fn(y, pred)
39. return [rmse, mae, mape]
40. *# 定义dataset*
41. class my\_Dataset(data.Dataset):
42. def \_\_init\_\_(self, features, labels):
43. self.X = features
44. self.y = labels
45. def \_\_getitem\_\_(self, index):
46. return self.X[index], self.y[index]
47. def \_\_len\_\_(self):
48. return self.X.shape[0]
50. class TrafficDataset:
51. def \_\_init\_\_(self):
52. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
53. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
54. *# 数据标准化*
55. self.min = self.raw\_data.min()
56. self.max = self.raw\_data.max()
57. self.data = (self.raw\_data - self.min) / (self.max - self.min)
58. def denormalize(self, x):
59. return x \* (self.max - self.min) + self.min
60. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
61. train\_x = []
62. train\_y = []
63. val\_x = []
64. val\_y = []
65. test\_x = []
66. test\_y = []
67. window\_size = 12
68. len\_train = int(self.data.shape[0] \* 0.6)
69. train\_seqs = self.data[:len\_train]
70. for i in range(train\_seqs.shape[0] - window\_size):
71. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
72. train\_y.append(train\_seqs[i + window\_size].squeeze())
73. len\_val = int(self.data.shape[0] \* 0.8)
74. val\_seqs = self.data[len\_train:len\_val]
75. for i in range(val\_seqs.shape[0] - window\_size):
76. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
77. val\_y.append(train\_seqs[i + window\_size].squeeze())
78. test\_seqs = self.data[len\_val:]
79. for i in range(test\_seqs.shape[0] - window\_size):
80. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
81. test\_y.append(test\_seqs[i + window\_size].squeeze())
82. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
83. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
84. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
85. return train\_set, val\_set, test\_set
86. batch\_size = 64
87. TrafficData = TrafficDataset()
88. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
89. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
90. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
91. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. print(train\_loader)
93. print(val\_loader)
94. print(test\_loader)
95. *###########前面是数据读取部分，已完成勿动###########*
96. *###########前面是数据读取部分，已完成勿动###########*
97. class MyRNN(nn.Module):
98. def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):
99. """
100. :param input\_size: 指定输入数据的维度。例如，对于简单的时间序列预测问题，每一步的输入均为一个采样值，因此input\_size=1.
101. :param hidden\_size: 指定隐藏状态的维度。这个值并不受输入和输出控制，但会影响模型的容量。
102. :param output\_size: 指定输出数据的维度。此值取决于具体的预测要求。例如，对简单的时间序列预测问题，output\_size=1.
103. """
104. super().\_\_init\_\_()
105. self.hidden\_size = hidden\_size
107. *# 可学习参数的维度设置，可以类比一下全连接网络的实现。其维度取决于输入数据的维度，以及指定的隐藏状态维度。*
108. self.w\_h = nn.Parameter(torch.rand(input\_size, hidden\_size))
109. self.u\_h = nn.Parameter(torch.rand(hidden\_size, hidden\_size))
110. self.b\_h = nn.Parameter(torch.zeros(hidden\_size))
112. self.w\_y = nn.Parameter(torch.rand(hidden\_size, output\_size))
113. self.b\_y = nn.Parameter(torch.zeros(output\_size))
115. *# 准备激活函数。Dropout函数可选。*
116. self.tanh = nn.Tanh()
117. self.leaky\_relu = nn.LeakyReLU()
119. *# 可选：使用性能更好的参数初始化函数*
120. for param in self.parameters():
121. if param.dim() > 1:
122. nn.init.xavier\_uniform\_(param)
124. def forward(self, x):
125. """
126. :param x: 输入序列。一般来说，此输入包含三个维度：batch，序列长度，以及每条数据的特征。
127. """
128. batch\_size = x.size(0)
129. seq\_len = x.size(1)
131. *# 初始化隐藏状态，一般设为全0。由于是内部新建的变量，需要同步设备位置。*
132. h = torch.zeros(batch\_size, self.hidden\_size).to(x.device)
133. *# RNN实际上只能一步一步处理序列。因此需要用循环迭代。*
134. y\_list = []
135. for i in range(seq\_len):
136. h = self.tanh(torch.matmul(x[:, i, :], self.w\_h) +
137. torch.matmul(h, self.u\_h) + self.b\_h)  *# (batch\_size, hidden\_size)*
138. y = self.leaky\_relu(torch.matmul(h, self.w\_y) + self.b\_y)  *# (batch\_size, output\_size)*
139. y\_list.append(y)
140. *# 一般来说，RNN的返回值为最后一步的隐藏状态，以及每一步的输出状态。*
141. return torch.stack(y\_list, dim=1),h
142. *# 测试函数（用于分类）*
143. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
144. rmse, mae, mape = 0, 0, 0
145. batch\_count = 0
146. total\_loss = 0.0
147. net.eval()
148. if output\_model is not None:
149. output\_model.eval()
150. for X, Y in data\_iter:
151. X = X.to(device).float()
152. Y = Y.to(device).float()
153. output,hidden = net(X)
154. if output\_model is not None:
155. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
156. else:
157. y\_hat = output[:, -1, :].squeeze(-1)
158. loss = loss\_fn(y\_hat, Y)
159. Y = denormalize\_fn(Y)
160. y\_hat = denormalize\_fn(y\_hat)
161. a, b, c = eval(Y.detach(), y\_hat.detach())
162. rmse += a
163. mae += b
164. mape += c
165. total\_loss += loss.detach().cpu().numpy().tolist()
166. batch\_count += 1
167. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
168. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
169. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
170. train\_loss\_lst = []
171. val\_loss\_lst = []
172. train\_score\_lst = []
173. val\_score\_lst = []
174. epoch\_time = []
175. best\_epoch = 0
176. best\_val\_rmse = 9999
177. early\_stop\_flag = 0
178. for epoch in range(num\_epoch):
179. net.train()
180. if output\_model is not None:
181. output\_model.train()
182. epoch\_loss = 0
183. batch\_count = 0
184. batch\_time = []
185. rmse, mae, mape = 0, 0, 0
186. for X, Y in train\_iter:
187. batch\_s = time.time()
188. X = X.to(device).float()
189. Y = Y.to(device).float()
190. optimizer.zero\_grad()
191. output, hidden = net(X)
192. if output\_model is not None:
193. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
194. else:
195. y\_hat = output[:, -1, :].squeeze(-1)
196. loss = loss\_fn(y\_hat, Y)
197. loss.backward()
198. optimizer.step()
199. Y = denormalize\_fn(Y)
200. y\_hat = denormalize\_fn(y\_hat)
201. a, b, c = eval(Y.detach(), y\_hat.detach())
202. rmse += a
203. mae += b
204. mape += c
205. epoch\_loss += loss.detach().cpu().numpy().tolist()
206. batch\_count += 1
207. *# sample\_num += X.shape[0]*
208. batch\_time.append(time.time() - batch\_s)
209. if is\_print and is\_print\_batch:
210. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
211. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
212. train\_loss = epoch\_loss / batch\_count
213. train\_loss\_lst.append(train\_loss)
214. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
215. *# 验证集*
216. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
217. val\_score\_lst.append(val\_score)
218. val\_loss\_lst.append(val\_loss)
219. epoch\_time.append(np.array(batch\_time).sum())
220. *# 打印本轮训练结果*
221. if is\_print:
222. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
223. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
224. *# 早停*
225. if val\_score[0] < best\_val\_rmse:
226. best\_val\_rmse = val\_score[0]
227. best\_epoch = epoch
228. early\_stop\_flag = 0
229. else:
230. early\_stop\_flag += 1
231. if early\_stop\_flag == early\_stop:
232. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
233. break
234. *# 输出最终训练结果*
235. print(f'\n{"\*" \* 40}\nFinal result:')
236. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
237. f'at epoch {best\_epoch}')
238. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
239. print()
240. *# 计算测试集效果*
241. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
242. print('Test result:')
243. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
244. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
245. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
246. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
247. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
248. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
249. plt.xlabel(x\_label)
250. plt.ylabel(y\_label)
251. plt.legend()
252. plt.show()
253. def plot\_metric(score\_log):
254. score\_log = np.array(score\_log)
255. plt.figure(figsize=(10, 6), dpi=300)
256. plt.subplot(2, 2, 1)
257. plt.plot(score\_log[:, 0], c='#d28ad4')
258. plt.ylabel('RMSE')
259. plt.subplot(2, 2, 2)
260. plt.plot(score\_log[:, 1], c='#e765eb')
261. plt.ylabel('MAE')
262. plt.subplot(2, 2, 3)
263. plt.plot(score\_log[:, 2], c='#6b016d')
264. plt.ylabel('MAPE')
265. plt.show()
266. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
267. input\_size = train\_set.X.shape[-1]
268. hidden\_size = 64
269. output\_size = 1
270. seq\_len = 12
271. lr = 0.0001
272. epochs = 80
273. loss\_func = nn.MSELoss()
274. my\_rnn = MyRNN(input\_size,hidden\_size,output\_size).to(device)
275. optimizer = torch.optim.Adam(my\_rnn. parameters(),lr)
276. train\_loss\_lst,val\_loss\_lst,train\_score\_lst,val\_score\_lst,stop\_epoch = train(my\_rnn,train\_loader,val\_loader,test\_loader,loss\_func,TrafficData.denormalize,optimizer,epochs,early\_stop=20,device=device,output\_model=None)
277. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
278. plot\_metric(train\_score\_lst)

**4.2.2**调用接口实现RNN

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.utils.data as data
6. import warnings
7. from torch.utils.data import DataLoader,Dataset
8. warnings.filterwarnings("ignore")
9. import random
10. from IPython import display
11. from matplotlib import pyplot as plt
12. from PIL import Image
13. import os
14. from torch import nn
15. import torch.optim as optim
16. from torch.nn import init
17. import torch.nn.functional as F
18. import time
19. import pandas as pd
20. from sklearn.utils import shuffle
21. import math
22. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
23. import datasets
24. import utils
25. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
26. def mape\_fn(y, pred):
27. mask = y != 0
28. y = y[mask]
29. pred = pred[mask]
30. mape = np.abs((y - pred) / y)
31. mape = np.mean(mape) \* 100
32. return mape
33. def eval(y, pred):
34. y = y.cpu().numpy()
35. pred = pred.cpu().numpy()
36. mse = mse\_fn(y, pred)
37. rmse = math.sqrt(mse)
38. mae = mae\_fn(y, pred)
39. mape = mape\_fn(y, pred)
40. return [rmse, mae, mape]
41. *# 定义dataset*
42. class my\_Dataset(data.Dataset):
43. def \_\_init\_\_(self, features, labels):
44. self.X = features
45. self.y = labels
46. def \_\_getitem\_\_(self, index):
47. return self.X[index], self.y[index]
48. def \_\_len\_\_(self):
49. return self.X.shape[0]
51. class TrafficDataset:
52. def \_\_init\_\_(self):
53. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
54. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
55. *# 数据标准化*
56. self.min = self.raw\_data.min()
57. self.max = self.raw\_data.max()
58. self.data = (self.raw\_data - self.min) / (self.max - self.min)
59. def denormalize(self, x):
60. return x \* (self.max - self.min) + self.min
61. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
62. train\_x = []
63. train\_y = []
64. val\_x = []
65. val\_y = []
66. test\_x = []
67. test\_y = []
68. window\_size = 12
69. len\_train = int(self.data.shape[0] \* 0.6)
70. train\_seqs = self.data[:len\_train]
71. for i in range(train\_seqs.shape[0] - window\_size):
72. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
73. train\_y.append(train\_seqs[i + window\_size].squeeze())
74. len\_val = int(self.data.shape[0] \* 0.8)
75. val\_seqs = self.data[len\_train:len\_val]
76. for i in range(val\_seqs.shape[0] - window\_size):
77. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
78. val\_y.append(train\_seqs[i + window\_size].squeeze())
79. test\_seqs = self.data[len\_val:]
80. for i in range(test\_seqs.shape[0] - window\_size):
81. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
82. test\_y.append(test\_seqs[i + window\_size].squeeze())
83. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
84. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
85. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
86. return train\_set, val\_set, test\_set
87. batch\_size = 64
88. TrafficData = TrafficDataset()
89. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
90. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
91. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. print(train\_loader)
94. print(val\_loader)
95. print(test\_loader)
96. *###########前面是数据读取部分，已完成勿动###########*
97. *###########前面是数据读取部分，已完成勿动###########*
98. nn\_rnn = nn.RNN(input\_size=input\_size,hidden\_size=hidden\_size,num\_layers=1,batch\_first=True). to(device)
99. out\_linear = nn.Sequential(nn.Linear (hidden\_size,1),
100. nn.LeakyReLU ()).to(device)
101. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()),lr)
102. *# 测试函数（用于分类）*
103. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
104. rmse, mae, mape = 0, 0, 0
105. batch\_count = 0
106. total\_loss = 0.0
107. net.eval()
108. if output\_model is not None:
109. output\_model.eval()
110. for X, Y in data\_iter:
111. X = X.to(device).float()
112. Y = Y.to(device).float()
113. output,hidden = net(X)
114. if output\_model is not None:
115. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
116. else:
117. y\_hat = output[:, -1, :].squeeze(-1)
118. loss = loss\_fn(y\_hat, Y)
119. Y = denormalize\_fn(Y)
120. y\_hat = denormalize\_fn(y\_hat)
121. a, b, c = eval(Y.detach(), y\_hat.detach())
122. rmse += a
123. mae += b
124. mape += c
125. total\_loss += loss.detach().cpu().numpy().tolist()
126. batch\_count += 1
127. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
128. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
129. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
130. train\_loss\_lst = []
131. val\_loss\_lst = []
132. train\_score\_lst = []
133. val\_score\_lst = []
134. epoch\_time = []
135. best\_epoch = 0
136. best\_val\_rmse = 9999
137. early\_stop\_flag = 0
138. for epoch in range(num\_epoch):
139. net.train()
140. if output\_model is not None:
141. output\_model.train()
142. epoch\_loss = 0
143. batch\_count = 0
144. batch\_time = []
145. rmse, mae, mape = 0, 0, 0
146. for X, Y in train\_iter:
147. batch\_s = time.time()
148. X = X.to(device).float()
149. Y = Y.to(device).float()
150. optimizer.zero\_grad()
151. output, hidden = net(X)
152. if output\_model is not None:
153. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
154. else:
155. y\_hat = output[:, -1, :].squeeze(-1)
156. loss = loss\_fn(y\_hat, Y)
157. loss.backward()
158. optimizer.step()
159. Y = denormalize\_fn(Y)
160. y\_hat = denormalize\_fn(y\_hat)
161. a, b, c = eval(Y.detach(), y\_hat.detach())
162. rmse += a
163. mae += b
164. mape += c
165. epoch\_loss += loss.detach().cpu().numpy().tolist()
166. batch\_count += 1
167. *# sample\_num += X.shape[0]*
168. batch\_time.append(time.time() - batch\_s)
169. if is\_print and is\_print\_batch:
170. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
171. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
172. train\_loss = epoch\_loss / batch\_count
173. train\_loss\_lst.append(train\_loss)
174. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
175. *# 验证集*
176. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
177. val\_score\_lst.append(val\_score)
178. val\_loss\_lst.append(val\_loss)
179. epoch\_time.append(np.array(batch\_time).sum())
180. *# 打印本轮训练结果*
181. if is\_print:
182. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
183. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
184. *# 早停*
185. if val\_score[0] < best\_val\_rmse:
186. best\_val\_rmse = val\_score[0]
187. best\_epoch = epoch
188. early\_stop\_flag = 0
189. else:
190. early\_stop\_flag += 1
191. if early\_stop\_flag == early\_stop:
192. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
193. break
194. *# 输出最终训练结果*
195. print(f'\n{"\*" \* 40}\nFinal result:')
196. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
197. f'at epoch {best\_epoch}')
198. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
199. print()
200. *# 计算测试集效果*
201. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
202. print('Test result:')
203. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
204. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
205. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
206. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
207. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
208. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
209. plt.xlabel(x\_label)
210. plt.ylabel(y\_label)
211. plt.legend()
212. plt.show()
213. def plot\_metric(score\_log):
214. score\_log = np.array(score\_log)
215. plt.figure(figsize=(10, 6), dpi=300)
216. plt.subplot(2, 2, 1)
217. plt.plot(score\_log[:, 0], c='#d28ad4')
218. plt.ylabel('RMSE')
219. plt.subplot(2, 2, 2)
220. plt.plot(score\_log[:, 1], c='#e765eb')
221. plt.ylabel('MAE')
222. plt.subplot(2, 2, 3)
223. plt.plot(score\_log[:, 2], c='#6b016d')
224. plt.ylabel('MAPE')
225. plt.show()
226. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
227. input\_size = train\_set.X.shape[-1]
228. hidden\_size = 64
229. output\_size = 1
230. seq\_len = 12
231. lr = 0.0001
232. epochs = 80
233. loss\_func = nn.MSELoss()
234. my\_rnn = MyRNN(input\_size,hidden\_size,output\_size).to(device)
235. optimizer = torch.optim.Adam(my\_rnn. parameters(),lr)
236. train\_loss\_lst,val\_loss\_lst,train\_score\_lst,val\_score\_lst,stop\_epoch = train(my\_rnn,train\_loader,val\_loader,test\_loader,loss\_func,TrafficData.denormalize,optimizer,epochs,early\_stop=20,device=device,output\_model=None)
237. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
238. plot\_metric(train\_score\_lst)

**4.2.3**手动实现LSTM

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.utils.data as data
5. import warnings
6. from torch.utils.data import DataLoader,Dataset
7. warnings.filterwarnings("ignore")
8. import random
9. from IPython import display
10. from matplotlib import pyplot as plt
11. from PIL import Image
12. import os
13. from torch import nn
14. import torch.optim as optim
15. from torch.nn import init
16. import torch.nn.functional as F
17. import time
18. import pandas as pd
19. from sklearn.utils import shuffle
20. import math
21. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
22. *#import datasets*
23. *#import utils*
24. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
25. def mape\_fn(y, pred):
26. mask = y != 0
27. y = y[mask]
28. pred = pred[mask]
29. mape = np.abs((y - pred) / y)
30. mape = np.mean(mape) \* 100
31. return mape
32. def eval(y, pred):
33. y = y.cpu().numpy()
34. pred = pred.cpu().numpy()
35. mse = mse\_fn(y, pred)
36. rmse = math.sqrt(mse)
37. mae = mae\_fn(y, pred)
38. mape = mape\_fn(y, pred)
39. return [rmse, mae, mape]
40. *# 定义dataset*
41. class my\_Dataset(data.Dataset):
42. def \_\_init\_\_(self, features, labels):
43. self.X = features
44. self.y = labels
45. def \_\_getitem\_\_(self, index):
46. return self.X[index], self.y[index]
47. def \_\_len\_\_(self):
48. return self.X.shape[0]
50. class TrafficDataset:
51. def \_\_init\_\_(self):
52. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
53. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
54. *# 数据标准化*
55. self.min = self.raw\_data.min()
56. self.max = self.raw\_data.max()
57. self.data = (self.raw\_data - self.min) / (self.max - self.min)
58. def denormalize(self, x):
59. return x \* (self.max - self.min) + self.min
60. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
61. train\_x = []
62. train\_y = []
63. val\_x = []
64. val\_y = []
65. test\_x = []
66. test\_y = []
67. window\_size = 12
68. len\_train = int(self.data.shape[0] \* 0.6)
69. train\_seqs = self.data[:len\_train]
70. for i in range(train\_seqs.shape[0] - window\_size):
71. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
72. train\_y.append(train\_seqs[i + window\_size].squeeze())
73. len\_val = int(self.data.shape[0] \* 0.8)
74. val\_seqs = self.data[len\_train:len\_val]
75. for i in range(val\_seqs.shape[0] - window\_size):
76. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
77. val\_y.append(train\_seqs[i + window\_size].squeeze())
78. test\_seqs = self.data[len\_val:]
79. for i in range(test\_seqs.shape[0] - window\_size):
80. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
81. test\_y.append(test\_seqs[i + window\_size].squeeze())
82. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
83. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
84. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
85. return train\_set, val\_set, test\_set
86. batch\_size = 64
87. TrafficData = TrafficDataset()
88. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
89. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
90. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
91. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. print(train\_loader)
93. print(val\_loader)
94. print(test\_loader)
95. *###########前面是数据读取部分，已完成勿动###########*
96. *###########前面是数据读取部分，已完成勿动###########*
97. class My\_LSTM(nn. Module):
98. def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):
99. super().\_\_init\_\_()
100. self.hidden\_size = hidden\_size
101. self.gates = nn.Linear(input\_size + hidden\_size, hidden\_size \* 4)
102. self.sigmoid = nn.Sigmoid()
103. self.tanh = nn. Tanh()
104. self.output = nn.Sequential(
105. nn.Linear(hidden\_size, hidden\_size // 2),
106. nn.ReLU(),
107. nn.Linear(hidden\_size // 2, output\_size)
108. )
109. for param in self.parameters():
110. if param.dim() > 1:
111. nn.init.xavier\_uniform\_(param)
112. def forward(self, x):
113. batch\_size = x.size(0)
114. seq\_len = x.size(1)
115. h, c = (torch.zeros(batch\_size, self.hidden\_size).to(x.device) for \_ in range(2))
116. y\_list = []
117. for i in range(seq\_len):
118. forget\_gate, input\_gate, output\_gate, candidate\_cell =                 self.gates(torch.cat([x[:, i, :], h], dim=-1)).chunk(4, -1)
119. forget\_gate, input\_gate, output\_gate = (self.sigmoid(g)
120. for g in (forget\_gate, input\_gate, output\_gate))
121. c = forget\_gate \* c + input\_gate \* self.tanh(candidate\_cell)
122. h = output\_gate \* self.tanh(c)
123. y\_list.append(self.output(h))
124. return torch.stack(y\_list, dim=1), (h, c)
125. *# 测试函数（用于分类）*
126. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
127. rmse, mae, mape = 0, 0, 0
128. batch\_count = 0
129. total\_loss = 0.0
130. net.eval()
131. if output\_model is not None:
132. output\_model.eval()
133. for X, Y in data\_iter:
134. X = X.to(device).float()
135. Y = Y.to(device).float()
136. output,hidden = net(X)
137. if output\_model is not None:
138. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
139. else:
140. y\_hat = output[:, -1, :].squeeze(-1)
141. loss = loss\_fn(y\_hat, Y)
142. Y = denormalize\_fn(Y)
143. y\_hat = denormalize\_fn(y\_hat)
144. a, b, c = eval(Y.detach(), y\_hat.detach())
145. rmse += a
146. mae += b
147. mape += c
148. total\_loss += loss.detach().cpu().numpy().tolist()
149. batch\_count += 1
150. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
151. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
152. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
153. train\_loss\_lst = []
154. val\_loss\_lst = []
155. train\_score\_lst = []
156. val\_score\_lst = []
157. epoch\_time = []
158. best\_epoch = 0
159. best\_val\_rmse = 9999
160. early\_stop\_flag = 0
161. for epoch in range(num\_epoch):
162. net.train()
163. if output\_model is not None:
164. output\_model.train()
165. epoch\_loss = 0
166. batch\_count = 0
167. batch\_time = []
168. rmse, mae, mape = 0, 0, 0
169. for X, Y in train\_iter:
170. batch\_s = time.time()
171. X = X.to(device).float()
172. Y = Y.to(device).float()
173. optimizer.zero\_grad()
174. output, hidden = net(X)
175. if output\_model is not None:
176. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
177. else:
178. y\_hat = output[:, -1, :].squeeze(-1)
179. loss = loss\_fn(y\_hat, Y)
180. loss.backward()
181. optimizer.step()
182. Y = denormalize\_fn(Y)
183. y\_hat = denormalize\_fn(y\_hat)
184. a, b, c = eval(Y.detach(), y\_hat.detach())
185. rmse += a
186. mae += b
187. mape += c
188. epoch\_loss += loss.detach().cpu().numpy().tolist()
189. batch\_count += 1
190. *# sample\_num += X.shape[0]*
191. batch\_time.append(time.time() - batch\_s)
192. if is\_print and is\_print\_batch:
193. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
194. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
195. train\_loss = epoch\_loss / batch\_count
196. train\_loss\_lst.append(train\_loss)
197. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
198. *# 验证集*
199. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
200. val\_score\_lst.append(val\_score)
201. val\_loss\_lst.append(val\_loss)
202. epoch\_time.append(np.array(batch\_time).sum())
203. *# 打印本轮训练结果*
204. if is\_print:
205. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
206. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
207. *# 早停*
208. if val\_score[0] < best\_val\_rmse:
209. best\_val\_rmse = val\_score[0]
210. best\_epoch = epoch
211. early\_stop\_flag = 0
212. else:
213. early\_stop\_flag += 1
214. if early\_stop\_flag == early\_stop:
215. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
216. break
217. *# 输出最终训练结果*
218. print(f'\n{"\*" \* 40}\nFinal result:')
219. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
220. f'at epoch {best\_epoch}')
221. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
222. print()
223. *# 计算测试集效果*
224. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
225. print('Test result:')
226. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
227. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
228. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
229. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
230. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
231. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
232. plt.xlabel(x\_label)
233. plt.ylabel(y\_label)
234. plt.legend()
235. plt.show()
236. def plot\_metric(score\_log):
237. score\_log = np.array(score\_log)
238. plt.figure(figsize=(10, 6), dpi=300)
239. plt.subplot(2, 2, 1)
240. plt.plot(score\_log[:, 0], c='#d28ad4')
241. plt.ylabel('RMSE')
242. plt.subplot(2, 2, 2)
243. plt.plot(score\_log[:, 1], c='#e765eb')
244. plt.ylabel('MAE')
245. plt.subplot(2, 2, 3)
246. plt.plot(score\_log[:, 2], c='#6b016d')
247. plt.ylabel('MAPE')
248. plt.show()
249. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
250. input\_size = train\_set.X.shape[-1]
251. hidden\_size = 64
252. output\_size = 1
253. seq\_len = 12
254. lr = 0.0001
255. epochs = 80
256. loss\_func = nn.MSELoss()
257. my\_rnn = My\_LSTM(input\_size, hidden\_size, output\_size).to(device)
258. optimizer = torch.optim.Adam(my\_rnn. parameters(),lr)
259. train\_loss\_lst,val\_loss\_lst,train\_score\_lst,val\_score\_lst,stop\_epoch = train(my\_rnn,train\_loader,val\_loader,test\_loader,loss\_func,TrafficData.denormalize,optimizer,epochs,early\_stop=20,device=device,output\_model=None)
260. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
261. plot\_metric(train\_score\_lst)

**4.2.4**调用接口实现LSTM

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.nn as nn
6. import torch.utils.data as data
7. import warnings
8. from torch.utils.data import DataLoader,Dataset
9. warnings.filterwarnings("ignore")
10. import random
11. from IPython import display
12. from matplotlib import pyplot as plt
13. from PIL import Image
14. import os
15. from torch import nn
16. import torch.optim as optim
17. from torch.nn import init
18. import torch.nn.functional as F
19. import time
20. import pandas as pd
21. from sklearn.utils import shuffle
22. import math
23. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
24. import datasets
25. import utils
26. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
27. def mape\_fn(y, pred):
28. mask = y != 0
29. y = y[mask]
30. pred = pred[mask]
31. mape = np.abs((y - pred) / y)
32. mape = np.mean(mape) \* 100
33. return mape
34. def eval(y, pred):
35. y = y.cpu().numpy()
36. pred = pred.cpu().numpy()
37. mse = mse\_fn(y, pred)
38. rmse = math.sqrt(mse)
39. mae = mae\_fn(y, pred)
40. mape = mape\_fn(y, pred)
41. return [rmse, mae, mape]
42. *# 定义dataset*
43. class my\_Dataset(data.Dataset):
44. def \_\_init\_\_(self, features, labels):
45. self.X = features
46. self.y = labels
47. def \_\_getitem\_\_(self, index):
48. return self.X[index], self.y[index]
49. def \_\_len\_\_(self):
50. return self.X.shape[0]
52. class TrafficDataset:
53. def \_\_init\_\_(self):
54. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
55. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
56. *# 数据标准化*
57. self.min = self.raw\_data.min()
58. self.max = self.raw\_data.max()
59. self.data = (self.raw\_data - self.min) / (self.max - self.min)
60. def denormalize(self, x):
61. return x \* (self.max - self.min) + self.min
62. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
63. train\_x = []
64. train\_y = []
65. val\_x = []
66. val\_y = []
67. test\_x = []
68. test\_y = []
69. window\_size = 12
70. len\_train = int(self.data.shape[0] \* 0.6)
71. train\_seqs = self.data[:len\_train]
72. for i in range(train\_seqs.shape[0] - window\_size):
73. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
74. train\_y.append(train\_seqs[i + window\_size].squeeze())
75. len\_val = int(self.data.shape[0] \* 0.8)
76. val\_seqs = self.data[len\_train:len\_val]
77. for i in range(val\_seqs.shape[0] - window\_size):
78. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
79. val\_y.append(train\_seqs[i + window\_size].squeeze())
80. test\_seqs = self.data[len\_val:]
81. for i in range(test\_seqs.shape[0] - window\_size):
82. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
83. test\_y.append(test\_seqs[i + window\_size].squeeze())
84. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
85. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
86. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
87. return train\_set, val\_set, test\_set
88. batch\_size = 64
89. TrafficData = TrafficDataset()
90. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
91. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
94. print(train\_loader)
95. print(val\_loader)
96. print(test\_loader)
97. *###########前面是数据读取部分，已完成勿动###########*
98. *###########前面是数据读取部分，已完成勿动###########*
99. *# 测试函数（用于分类）*
100. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
101. rmse, mae, mape = 0, 0, 0
102. batch\_count = 0
103. total\_loss = 0.0
104. net.eval()
105. if output\_model is not None:
106. output\_model.eval()
107. for X, Y in data\_iter:
108. X = X.to(device).float()
109. Y = Y.to(device).float()
110. output,hidden = net(X)
111. if output\_model is not None:
112. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
113. else:
114. y\_hat = output[:, -1, :].squeeze(-1)
115. loss = loss\_fn(y\_hat, Y)
116. Y = denormalize\_fn(Y)
117. y\_hat = denormalize\_fn(y\_hat)
118. a, b, c = eval(Y.detach(), y\_hat.detach())
119. rmse += a
120. mae += b
121. mape += c
122. total\_loss += loss.detach().cpu().numpy().tolist()
123. batch\_count += 1
124. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
125. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
126. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
127. train\_loss\_lst = []
128. val\_loss\_lst = []
129. train\_score\_lst = []
130. val\_score\_lst = []
131. epoch\_time = []
132. best\_epoch = 0
133. best\_val\_rmse = 9999
134. early\_stop\_flag = 0
135. for epoch in range(num\_epoch):
136. net.train()
137. if output\_model is not None:
138. output\_model.train()
139. epoch\_loss = 0
140. batch\_count = 0
141. batch\_time = []
142. rmse, mae, mape = 0, 0, 0
143. for X, Y in train\_iter:
144. batch\_s = time.time()
145. X = X.to(device).float()
146. Y = Y.to(device).float()
147. optimizer.zero\_grad()
148. output, hidden = net(X)
149. if output\_model is not None:
150. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
151. else:
152. y\_hat = output[:, -1, :].squeeze(-1)
153. loss = loss\_fn(y\_hat, Y)
154. loss.backward()
155. optimizer.step()
156. Y = denormalize\_fn(Y)
157. y\_hat = denormalize\_fn(y\_hat)
158. a, b, c = eval(Y.detach(), y\_hat.detach())
159. rmse += a
160. mae += b
161. mape += c
162. epoch\_loss += loss.detach().cpu().numpy().tolist()
163. batch\_count += 1
164. *# sample\_num += X.shape[0]*
165. batch\_time.append(time.time() - batch\_s)
166. if is\_print and is\_print\_batch:
167. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
168. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
169. train\_loss = epoch\_loss / batch\_count
170. train\_loss\_lst.append(train\_loss)
171. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
172. *# 验证集*
173. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
174. val\_score\_lst.append(val\_score)
175. val\_loss\_lst.append(val\_loss)
176. epoch\_time.append(np.array(batch\_time).sum())
177. *# 打印本轮训练结果*
178. if is\_print:
179. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
180. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
181. *# 早停*
182. if val\_score[0] < best\_val\_rmse:
183. best\_val\_rmse = val\_score[0]
184. best\_epoch = epoch
185. early\_stop\_flag = 0
186. else:
187. early\_stop\_flag += 1
188. if early\_stop\_flag == early\_stop:
189. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
190. break
191. *# 输出最终训练结果*
192. print(f'\n{"\*" \* 40}\nFinal result:')
193. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
194. f'at epoch {best\_epoch}')
195. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
196. print()
197. *# 计算测试集效果*
198. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
199. print('Test result:')
200. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
201. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
202. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
203. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
204. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
205. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
206. plt.xlabel(x\_label)
207. plt.ylabel(y\_label)
208. plt.legend()
209. plt.show()
210. def plot\_metric(score\_log):
211. score\_log = np.array(score\_log)
212. plt.figure(figsize=(10, 6), dpi=300)
213. plt.subplot(2, 2, 1)
214. plt.plot(score\_log[:, 0], c='#d28ad4')
215. plt.ylabel('RMSE')
216. plt.subplot(2, 2, 2)
217. plt.plot(score\_log[:, 1], c='#e765eb')
218. plt.ylabel('MAE')
219. plt.subplot(2, 2, 3)
220. plt.plot(score\_log[:, 2], c='#6b016d')
221. plt.ylabel('MAPE')
222. plt.show()
223. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
224. input\_size = train\_set.X.shape[-1]
225. hidden\_size = 64
226. output\_size = 1
227. seq\_len = 12
228. lr = 0.0001
229. epochs = 80
230. loss\_func = nn.MSELoss()
231. nn\_rnn = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=1, batch\_first=True).to(device)
232. out\_linear = nn.Sequential(nn.Linear(hidden\_size, 1),
233. nn.LeakyReLU()).to(device)
234. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()), lr)
235. train\_loss\_lst, val\_loss\_lst,     train\_score\_lst, val\_score\_lst, stop\_epoch = train(nn\_rnn, train\_loader, val\_loader, test\_loader,
236. loss\_func, TrafficData.denormalize, optimizer, epochs,
237. early\_stop=20, device=device, output\_model=out\_linear)
238. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
239. plot\_metric(train\_score\_lst)

**4.2.5**手动实现GRU

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.utils.data as data
5. import warnings
6. from torch.utils.data import DataLoader,Dataset
7. warnings.filterwarnings("ignore")
8. import random
9. from IPython import display
10. from matplotlib import pyplot as plt
11. from PIL import Image
12. import os
13. from torch import nn
14. import torch.optim as optim
15. from torch.nn import init
16. import torch.nn.functional as F
17. import time
18. import pandas as pd
19. from sklearn.utils import shuffle
20. import math
21. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
22. *#import datasets*
23. *#import utils*
24. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
25. def mape\_fn(y, pred):
26. mask = y != 0
27. y = y[mask]
28. pred = pred[mask]
29. mape = np.abs((y - pred) / y)
30. mape = np.mean(mape) \* 100
31. return mape
32. def eval(y, pred):
33. y = y.cpu().numpy()
34. pred = pred.cpu().numpy()
35. mse = mse\_fn(y, pred)
36. rmse = math.sqrt(mse)
37. mae = mae\_fn(y, pred)
38. mape = mape\_fn(y, pred)
39. return [rmse, mae, mape]
40. *# 定义dataset*
41. class my\_Dataset(data.Dataset):
42. def \_\_init\_\_(self, features, labels):
43. self.X = features
44. self.y = labels
45. def \_\_getitem\_\_(self, index):
46. return self.X[index], self.y[index]
47. def \_\_len\_\_(self):
48. return self.X.shape[0]
50. class TrafficDataset:
51. def \_\_init\_\_(self):
52. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
53. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
54. *# 数据标准化*
55. self.min = self.raw\_data.min()
56. self.max = self.raw\_data.max()
57. self.data = (self.raw\_data - self.min) / (self.max - self.min)
58. def denormalize(self, x):
59. return x \* (self.max - self.min) + self.min
60. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
61. train\_x = []
62. train\_y = []
63. val\_x = []
64. val\_y = []
65. test\_x = []
66. test\_y = []
67. window\_size = 12
68. len\_train = int(self.data.shape[0] \* 0.6)
69. train\_seqs = self.data[:len\_train]
70. for i in range(train\_seqs.shape[0] - window\_size):
71. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
72. train\_y.append(train\_seqs[i + window\_size].squeeze())
73. len\_val = int(self.data.shape[0] \* 0.8)
74. val\_seqs = self.data[len\_train:len\_val]
75. for i in range(val\_seqs.shape[0] - window\_size):
76. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
77. val\_y.append(train\_seqs[i + window\_size].squeeze())
78. test\_seqs = self.data[len\_val:]
79. for i in range(test\_seqs.shape[0] - window\_size):
80. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
81. test\_y.append(test\_seqs[i + window\_size].squeeze())
82. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
83. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
84. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
85. return train\_set, val\_set, test\_set
86. batch\_size = 64
87. TrafficData = TrafficDataset()
88. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
89. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
90. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
91. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. print(train\_loader)
93. print(val\_loader)
94. print(test\_loader)
95. *###########前面是数据读取部分，已完成勿动###########*
96. *###########前面是数据读取部分，已完成勿动###########*
97. class MyGRU(nn. Module):
98. def \_\_init\_\_(self, input\_size, hidden\_size, output\_size):
99. super().\_\_init\_\_()
100. self.hidden\_size = hidden\_size
101. self.gates = nn.Linear(input\_size + hidden\_size, hidden\_size \* 4)
102. self.sigmoid = nn.Sigmoid()
103. self.tanh = nn. Tanh()
104. self.output = nn.Sequential(
105. nn.Linear(hidden\_size, hidden\_size // 2),
106. nn.ReLU(),
107. nn.Linear(hidden\_size // 2, output\_size)
108. )
109. for param in self.parameters():
110. if param.dim() > 1:
111. nn.init.xavier\_uniform\_(param)
112. def forward(self, x):
113. batch\_size = x.size(0)
114. seq\_len = x.size(1)
115. h = torch.zeros(batch\_size, self.hidden\_size).to(x.device)
116. y\_list = []
117. for i in range(seq\_len):
118. forget\_gate, input\_gate, output\_gate, candidate\_cell =                 self.gates(torch.cat([x[:, i, :], h], dim=-1)).chunk(4, -1)
119. forget\_gate, input\_gate, output\_gate = (self.sigmoid(g)
120. for g in (forget\_gate, input\_gate, output\_gate))
122. h = (1-forget\_gate)\*h+forget\_gate\*candidate\_cell
123. y\_list.append(self.output(h))
124. return torch.stack(y\_list, dim=1),h
125. *# 测试函数（用于分类）*
126. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
127. rmse, mae, mape = 0, 0, 0
128. batch\_count = 0
129. total\_loss = 0.0
130. net.eval()
131. if output\_model is not None:
132. output\_model.eval()
133. for X, Y in data\_iter:
134. X = X.to(device).float()
135. Y = Y.to(device).float()
136. output,hidden = net(X)
137. if output\_model is not None:
138. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
139. else:
140. y\_hat = output[:, -1, :].squeeze(-1)
141. loss = loss\_fn(y\_hat, Y)
142. Y = denormalize\_fn(Y)
143. y\_hat = denormalize\_fn(y\_hat)
144. a, b, c = eval(Y.detach(), y\_hat.detach())
145. rmse += a
146. mae += b
147. mape += c
148. total\_loss += loss.detach().cpu().numpy().tolist()
149. batch\_count += 1
150. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
151. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
152. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
153. train\_loss\_lst = []
154. val\_loss\_lst = []
155. train\_score\_lst = []
156. val\_score\_lst = []
157. epoch\_time = []
158. best\_epoch = 0
159. best\_val\_rmse = 9999
160. early\_stop\_flag = 0
161. for epoch in range(num\_epoch):
162. net.train()
163. if output\_model is not None:
164. output\_model.train()
165. epoch\_loss = 0
166. batch\_count = 0
167. batch\_time = []
168. rmse, mae, mape = 0, 0, 0
169. for X, Y in train\_iter:
170. batch\_s = time.time()
171. X = X.to(device).float()
172. Y = Y.to(device).float()
173. optimizer.zero\_grad()
174. output, hidden = net(X)
175. if output\_model is not None:
176. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
177. else:
178. y\_hat = output[:, -1, :].squeeze(-1)
179. loss = loss\_fn(y\_hat, Y)
180. loss.backward()
181. optimizer.step()
182. Y = denormalize\_fn(Y)
183. y\_hat = denormalize\_fn(y\_hat)
184. a, b, c = eval(Y.detach(), y\_hat.detach())
185. rmse += a
186. mae += b
187. mape += c
188. epoch\_loss += loss.detach().cpu().numpy().tolist()
189. batch\_count += 1
190. *# sample\_num += X.shape[0]*
191. batch\_time.append(time.time() - batch\_s)
192. if is\_print and is\_print\_batch:
193. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
194. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
195. train\_loss = epoch\_loss / batch\_count
196. train\_loss\_lst.append(train\_loss)
197. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
198. *# 验证集*
199. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
200. val\_score\_lst.append(val\_score)
201. val\_loss\_lst.append(val\_loss)
202. epoch\_time.append(np.array(batch\_time).sum())
203. *# 打印本轮训练结果*
204. if is\_print:
205. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
206. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
207. *# 早停*
208. if val\_score[0] < best\_val\_rmse:
209. best\_val\_rmse = val\_score[0]
210. best\_epoch = epoch
211. early\_stop\_flag = 0
212. else:
213. early\_stop\_flag += 1
214. if early\_stop\_flag == early\_stop:
215. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
216. break
217. *# 输出最终训练结果*
218. print(f'\n{"\*" \* 40}\nFinal result:')
219. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
220. f'at epoch {best\_epoch}')
221. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
222. print()
223. *# 计算测试集效果*
224. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
225. print('Test result:')
226. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
227. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
228. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
229. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
230. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
231. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
232. plt.xlabel(x\_label)
233. plt.ylabel(y\_label)
234. plt.legend()
235. plt.show()
236. def plot\_metric(score\_log):
237. score\_log = np.array(score\_log)
238. plt.figure(figsize=(10, 6), dpi=300)
239. plt.subplot(2, 2, 1)
240. plt.plot(score\_log[:, 0], c='#d28ad4')
241. plt.ylabel('RMSE')
242. plt.subplot(2, 2, 2)
243. plt.plot(score\_log[:, 1], c='#e765eb')
244. plt.ylabel('MAE')
245. plt.subplot(2, 2, 3)
246. plt.plot(score\_log[:, 2], c='#6b016d')
247. plt.ylabel('MAPE')
248. plt.show()
249. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
250. input\_size = train\_set.X.shape[-1]
251. hidden\_size = 64
252. output\_size = 1
253. seq\_len = 12
254. lr = 0.0001
255. epochs = 80
256. loss\_func = nn.MSELoss()
257. my\_rnn = MyGRU(input\_size, hidden\_size, output\_size).to(device)
258. optimizer = torch.optim.Adam(my\_rnn. parameters(),lr)
259. train\_loss\_lst,val\_loss\_lst,train\_score\_lst,val\_score\_lst,stop\_epoch = train(my\_rnn,train\_loader,val\_loader,test\_loader,loss\_func,TrafficData.denormalize,optimizer,epochs,early\_stop=20,device=device,output\_model=None)
260. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
261. plot\_metric(train\_score\_lst)

**4.2.6**调用接口实现GRU

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.nn as nn
6. import torch.utils.data as data
7. import warnings
8. from torch.utils.data import DataLoader,Dataset
9. warnings.filterwarnings("ignore")
10. import random
11. from IPython import display
12. from matplotlib import pyplot as plt
13. from PIL import Image
14. import os
15. from torch import nn
16. import torch.optim as optim
17. from torch.nn import init
18. import torch.nn.functional as F
19. import time
20. import pandas as pd
21. from sklearn.utils import shuffle
22. import math
23. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
24. import datasets
25. import utils
26. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
27. def mape\_fn(y, pred):
28. mask = y != 0
29. y = y[mask]
30. pred = pred[mask]
31. mape = np.abs((y - pred) / y)
32. mape = np.mean(mape) \* 100
33. return mape
34. def eval(y, pred):
35. y = y.cpu().numpy()
36. pred = pred.cpu().numpy()
37. mse = mse\_fn(y, pred)
38. rmse = math.sqrt(mse)
39. mae = mae\_fn(y, pred)
40. mape = mape\_fn(y, pred)
41. return [rmse, mae, mape]
42. *# 定义dataset*
43. class my\_Dataset(data.Dataset):
44. def \_\_init\_\_(self, features, labels):
45. self.X = features
46. self.y = labels
47. def \_\_getitem\_\_(self, index):
48. return self.X[index], self.y[index]
49. def \_\_len\_\_(self):
50. return self.X.shape[0]
52. class TrafficDataset:
53. def \_\_init\_\_(self):
54. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
55. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
56. *# 数据标准化*
57. self.min = self.raw\_data.min()
58. self.max = self.raw\_data.max()
59. self.data = (self.raw\_data - self.min) / (self.max - self.min)
60. def denormalize(self, x):
61. return x \* (self.max - self.min) + self.min
62. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
63. train\_x = []
64. train\_y = []
65. val\_x = []
66. val\_y = []
67. test\_x = []
68. test\_y = []
69. window\_size = 12
70. len\_train = int(self.data.shape[0] \* 0.6)
71. train\_seqs = self.data[:len\_train]
72. for i in range(train\_seqs.shape[0] - window\_size):
73. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
74. train\_y.append(train\_seqs[i + window\_size].squeeze())
75. len\_val = int(self.data.shape[0] \* 0.8)
76. val\_seqs = self.data[len\_train:len\_val]
77. for i in range(val\_seqs.shape[0] - window\_size):
78. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
79. val\_y.append(train\_seqs[i + window\_size].squeeze())
80. test\_seqs = self.data[len\_val:]
81. for i in range(test\_seqs.shape[0] - window\_size):
82. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
83. test\_y.append(test\_seqs[i + window\_size].squeeze())
84. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
85. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
86. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
87. return train\_set, val\_set, test\_set
88. batch\_size = 64
89. TrafficData = TrafficDataset()
90. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
91. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
94. print(train\_loader)
95. print(val\_loader)
96. print(test\_loader)
97. *###########前面是数据读取部分，已完成勿动###########*
98. *###########前面是数据读取部分，已完成勿动###########*
99. *# 测试函数（用于分类）*
100. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
101. rmse, mae, mape = 0, 0, 0
102. batch\_count = 0
103. total\_loss = 0.0
104. net.eval()
105. if output\_model is not None:
106. output\_model.eval()
107. for X, Y in data\_iter:
108. X = X.to(device).float()
109. Y = Y.to(device).float()
110. output,hidden = net(X)
111. if output\_model is not None:
112. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
113. else:
114. y\_hat = output[:, -1, :].squeeze(-1)
115. loss = loss\_fn(y\_hat, Y)
116. Y = denormalize\_fn(Y)
117. y\_hat = denormalize\_fn(y\_hat)
118. a, b, c = eval(Y.detach(), y\_hat.detach())
119. rmse += a
120. mae += b
121. mape += c
122. total\_loss += loss.detach().cpu().numpy().tolist()
123. batch\_count += 1
124. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
125. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
126. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
127. train\_loss\_lst = []
128. val\_loss\_lst = []
129. train\_score\_lst = []
130. val\_score\_lst = []
131. epoch\_time = []
132. best\_epoch = 0
133. best\_val\_rmse = 9999
134. early\_stop\_flag = 0
135. for epoch in range(num\_epoch):
136. net.train()
137. if output\_model is not None:
138. output\_model.train()
139. epoch\_loss = 0
140. batch\_count = 0
141. batch\_time = []
142. rmse, mae, mape = 0, 0, 0
143. for X, Y in train\_iter:
144. batch\_s = time.time()
145. X = X.to(device).float()
146. Y = Y.to(device).float()
147. optimizer.zero\_grad()
148. output, hidden = net(X)
149. if output\_model is not None:
150. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
151. else:
152. y\_hat = output[:, -1, :].squeeze(-1)
153. loss = loss\_fn(y\_hat, Y)
154. loss.backward()
155. optimizer.step()
156. Y = denormalize\_fn(Y)
157. y\_hat = denormalize\_fn(y\_hat)
158. a, b, c = eval(Y.detach(), y\_hat.detach())
159. rmse += a
160. mae += b
161. mape += c
162. epoch\_loss += loss.detach().cpu().numpy().tolist()
163. batch\_count += 1
164. *# sample\_num += X.shape[0]*
165. batch\_time.append(time.time() - batch\_s)
166. if is\_print and is\_print\_batch:
167. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
168. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
169. train\_loss = epoch\_loss / batch\_count
170. train\_loss\_lst.append(train\_loss)
171. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
172. *# 验证集*
173. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
174. val\_score\_lst.append(val\_score)
175. val\_loss\_lst.append(val\_loss)
176. epoch\_time.append(np.array(batch\_time).sum())
177. *# 打印本轮训练结果*
178. if is\_print:
179. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
180. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
181. *# 早停*
182. if val\_score[0] < best\_val\_rmse:
183. best\_val\_rmse = val\_score[0]
184. best\_epoch = epoch
185. early\_stop\_flag = 0
186. else:
187. early\_stop\_flag += 1
188. if early\_stop\_flag == early\_stop:
189. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
190. break
191. *# 输出最终训练结果*
192. print(f'\n{"\*" \* 40}\nFinal result:')
193. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
194. f'at epoch {best\_epoch}')
195. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
196. print()
197. *# 计算测试集效果*
198. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
199. print('Test result:')
200. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
201. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
202. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
203. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
204. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
205. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
206. plt.xlabel(x\_label)
207. plt.ylabel(y\_label)
208. plt.legend()
209. plt.show()
210. def plot\_metric(score\_log):
211. score\_log = np.array(score\_log)
212. plt.figure(figsize=(10, 6), dpi=300)
213. plt.subplot(2, 2, 1)
214. plt.plot(score\_log[:, 0], c='#d28ad4')
215. plt.ylabel('RMSE')
216. plt.subplot(2, 2, 2)
217. plt.plot(score\_log[:, 1], c='#e765eb')
218. plt.ylabel('MAE')
219. plt.subplot(2, 2, 3)
220. plt.plot(score\_log[:, 2], c='#6b016d')
221. plt.ylabel('MAPE')
222. plt.show()
223. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
224. input\_size = train\_set.X.shape[-1]
225. hidden\_size = 64
226. output\_size = 1
227. seq\_len = 12
228. lr = 0.0001
229. epochs = 80
230. loss\_func = nn.MSELoss()
231. nn\_rnn = nn.GRU(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=1, batch\_first=True).to(device)
232. out\_linear = nn.Sequential(nn.Linear(hidden\_size, 1),
233. nn.LeakyReLU()).to(device)
234. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()), lr)
235. train\_loss\_lst, val\_loss\_lst,     train\_score\_lst, val\_score\_lst, stop\_epoch = train(nn\_rnn, train\_loader, val\_loader, test\_loader,
236. loss\_func, TrafficData.denormalize, optimizer, epochs,
237. early\_stop=20, device=device, output\_model=out\_linear)
238. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
239. plot\_metric(train\_score\_lst)

#### 4.3 对比实验

**4.3.1** 从训练时间、预测精度、Loss变化等角度对比分析RNN、LSTM和GRU在相同数据集上的实验结果（最好使用图表展示）

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.nn as nn
6. import torch.utils.data as data
7. import warnings
8. from torch.utils.data import DataLoader,Dataset
9. warnings.filterwarnings("ignore")
10. import random
11. from IPython import display
12. from matplotlib import pyplot as plt
13. from PIL import Image
14. import os
15. from torch import nn
16. import torch.optim as optim
17. from torch.nn import init
18. import torch.nn.functional as F
19. import time
20. import pandas as pd
21. from sklearn.utils import shuffle
22. import math
23. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
24. import datasets
25. import utils
26. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
27. def mape\_fn(y, pred):
28. mask = y != 0
29. y = y[mask]
30. pred = pred[mask]
31. mape = np.abs((y - pred) / y)
32. mape = np.mean(mape) \* 100
33. return mape
34. def eval(y, pred):
35. y = y.cpu().numpy()
36. pred = pred.cpu().numpy()
37. mse = mse\_fn(y, pred)
38. rmse = math.sqrt(mse)
39. mae = mae\_fn(y, pred)
40. mape = mape\_fn(y, pred)
41. return [rmse, mae, mape]
42. *# 定义dataset*
43. class my\_Dataset(data.Dataset):
44. def \_\_init\_\_(self, features, labels):
45. self.X = features
46. self.y = labels
47. def \_\_getitem\_\_(self, index):
48. return self.X[index], self.y[index]
49. def \_\_len\_\_(self):
50. return self.X.shape[0]
52. class TrafficDataset:
53. def \_\_init\_\_(self):
54. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
55. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
56. *# 数据标准化*
57. self.min = self.raw\_data.min()
58. self.max = self.raw\_data.max()
59. self.data = (self.raw\_data - self.min) / (self.max - self.min)
60. def denormalize(self, x):
61. return x \* (self.max - self.min) + self.min
62. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
63. train\_x = []
64. train\_y = []
65. val\_x = []
66. val\_y = []
67. test\_x = []
68. test\_y = []
69. window\_size = 12
70. len\_train = int(self.data.shape[0] \* 0.6)
71. train\_seqs = self.data[:len\_train]
72. for i in range(train\_seqs.shape[0] - window\_size):
73. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
74. train\_y.append(train\_seqs[i + window\_size].squeeze())
75. len\_val = int(self.data.shape[0] \* 0.8)
76. val\_seqs = self.data[len\_train:len\_val]
77. for i in range(val\_seqs.shape[0] - window\_size):
78. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
79. val\_y.append(train\_seqs[i + window\_size].squeeze())
80. test\_seqs = self.data[len\_val:]
81. for i in range(test\_seqs.shape[0] - window\_size):
82. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
83. test\_y.append(test\_seqs[i + window\_size].squeeze())
84. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
85. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
86. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
87. return train\_set, val\_set, test\_set
88. batch\_size = 64
89. TrafficData = TrafficDataset()
90. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
91. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
94. print(train\_loader)
95. print(val\_loader)
96. print(test\_loader)
97. *###########前面是数据读取部分，已完成勿动###########*
98. *###########前面是数据读取部分，已完成勿动###########*
99. *# 测试函数（用于分类）*
100. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
101. rmse, mae, mape = 0, 0, 0
102. batch\_count = 0
103. total\_loss = 0.0
104. net.eval()
105. if output\_model is not None:
106. output\_model.eval()
107. for X, Y in data\_iter:
108. X = X.to(device).float()
109. Y = Y.to(device).float()
110. output,hidden = net(X)
111. if output\_model is not None:
112. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
113. else:
114. y\_hat = output[:, -1, :].squeeze(-1)
115. loss = loss\_fn(y\_hat, Y)
116. Y = denormalize\_fn(Y)
117. y\_hat = denormalize\_fn(y\_hat)
118. a, b, c = eval(Y.detach(), y\_hat.detach())
119. rmse += a
120. mae += b
121. mape += c
122. total\_loss += loss.detach().cpu().numpy().tolist()
123. batch\_count += 1
124. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
125. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
126. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
127. train\_loss\_lst = []
128. val\_loss\_lst = []
129. train\_score\_lst = []
130. val\_score\_lst = []
131. epoch\_time = []
132. best\_epoch = 0
133. best\_val\_rmse = 9999
134. early\_stop\_flag = 0
135. for epoch in range(num\_epoch):
136. net.train()
137. if output\_model is not None:
138. output\_model.train()
139. epoch\_loss = 0
140. batch\_count = 0
141. batch\_time = []
142. rmse, mae, mape = 0, 0, 0
143. for X, Y in train\_iter:
144. batch\_s = time.time()
145. X = X.to(device).float()
146. Y = Y.to(device).float()
147. optimizer.zero\_grad()
148. output, hidden = net(X)
149. if output\_model is not None:
150. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
151. else:
152. y\_hat = output[:, -1, :].squeeze(-1)
153. loss = loss\_fn(y\_hat, Y)
154. loss.backward()
155. optimizer.step()
156. Y = denormalize\_fn(Y)
157. y\_hat = denormalize\_fn(y\_hat)
158. a, b, c = eval(Y.detach(), y\_hat.detach())
159. rmse += a
160. mae += b
161. mape += c
162. epoch\_loss += loss.detach().cpu().numpy().tolist()
163. batch\_count += 1
164. *# sample\_num += X.shape[0]*
165. batch\_time.append(time.time() - batch\_s)
166. if is\_print and is\_print\_batch:
167. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
168. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
169. train\_loss = epoch\_loss / batch\_count
170. train\_loss\_lst.append(train\_loss)
171. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
172. *# 验证集*
173. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
174. val\_score\_lst.append(val\_score)
175. val\_loss\_lst.append(val\_loss)
176. epoch\_time.append(np.array(batch\_time).sum())
177. *# 打印本轮训练结果*
178. if is\_print:
179. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
180. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
181. *# 早停*
182. if val\_score[0] < best\_val\_rmse:
183. best\_val\_rmse = val\_score[0]
184. best\_epoch = epoch
185. early\_stop\_flag = 0
186. else:
187. early\_stop\_flag += 1
188. if early\_stop\_flag == early\_stop:
189. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
190. break
191. *# 输出最终训练结果*
192. print(f'\n{"\*" \* 40}\nFinal result:')
193. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
194. f'at epoch {best\_epoch}')
195. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
196. print()
197. *# 计算测试集效果*
198. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
199. print('Test result:')
200. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
201. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
202. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
203. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
204. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
205. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
206. plt.xlabel(x\_label)
207. plt.ylabel(y\_label)
208. plt.legend()
209. plt.show()
210. def plot\_metric(score\_log):
211. score\_log = np.array(score\_log)
212. plt.figure(figsize=(10, 6), dpi=300)
213. plt.subplot(2, 2, 1)
214. plt.plot(score\_log[:, 0], c='#d28ad4')
215. plt.ylabel('RMSE')
216. plt.subplot(2, 2, 2)
217. plt.plot(score\_log[:, 1], c='#e765eb')
218. plt.ylabel('MAE')
219. plt.subplot(2, 2, 3)
220. plt.plot(score\_log[:, 2], c='#6b016d')
221. plt.ylabel('MAPE')
222. plt.show()
223. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
224. input\_size = train\_set.X.shape[-1]
225. hidden\_size = 64
226. output\_size = 1
227. seq\_len = 12
228. lr = 0.0001
229. epochs = 80
230. loss\_func = nn.MSELoss()
231. nn\_rnn = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=1, batch\_first=True).to(device)
232. out\_linear = nn.Sequential(nn.Linear(hidden\_size, 1),
233. nn.LeakyReLU()).to(device)
234. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()), lr)
235. train\_loss\_lst, val\_loss\_lst,     train\_score\_lst, val\_score\_lst, stop\_epoch = train(nn\_rnn, train\_loader, val\_loader, test\_loader,
236. loss\_func, TrafficData.denormalize, optimizer, epochs,
237. early\_stop=20, device=device, output\_model=out\_linear)
238. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
239. plot\_metric(train\_score\_lst)

#### 4.4 超参数实验

**4.4.1** 在LSTM上进行不同超参数的对比分析（hidden\_size）

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.nn as nn
6. import torch.utils.data as data
7. import warnings
8. from torch.utils.data import DataLoader,Dataset
9. warnings.filterwarnings("ignore")
10. import random
11. from IPython import display
12. from matplotlib import pyplot as plt
13. from PIL import Image
14. import os
15. from torch import nn
16. import torch.optim as optim
17. from torch.nn import init
18. import torch.nn.functional as F
19. import time
20. import pandas as pd
21. from sklearn.utils import shuffle
22. import math
23. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
24. import datasets
25. import utils
26. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
27. def mape\_fn(y, pred):
28. mask = y != 0
29. y = y[mask]
30. pred = pred[mask]
31. mape = np.abs((y - pred) / y)
32. mape = np.mean(mape) \* 100
33. return mape
34. def eval(y, pred):
35. y = y.cpu().numpy()
36. pred = pred.cpu().numpy()
37. mse = mse\_fn(y, pred)
38. rmse = math.sqrt(mse)
39. mae = mae\_fn(y, pred)
40. mape = mape\_fn(y, pred)
41. return [rmse, mae, mape]
42. *# 定义dataset*
43. class my\_Dataset(data.Dataset):
44. def \_\_init\_\_(self, features, labels):
45. self.X = features
46. self.y = labels
47. def \_\_getitem\_\_(self, index):
48. return self.X[index], self.y[index]
49. def \_\_len\_\_(self):
50. return self.X.shape[0]
52. class TrafficDataset:
53. def \_\_init\_\_(self):
54. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
55. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
56. *# 数据标准化*
57. self.min = self.raw\_data.min()
58. self.max = self.raw\_data.max()
59. self.data = (self.raw\_data - self.min) / (self.max - self.min)
60. def denormalize(self, x):
61. return x \* (self.max - self.min) + self.min
62. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
63. train\_x = []
64. train\_y = []
65. val\_x = []
66. val\_y = []
67. test\_x = []
68. test\_y = []
69. window\_size = 12
70. len\_train = int(self.data.shape[0] \* 0.6)
71. train\_seqs = self.data[:len\_train]
72. for i in range(train\_seqs.shape[0] - window\_size):
73. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
74. train\_y.append(train\_seqs[i + window\_size].squeeze())
75. len\_val = int(self.data.shape[0] \* 0.8)
76. val\_seqs = self.data[len\_train:len\_val]
77. for i in range(val\_seqs.shape[0] - window\_size):
78. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
79. val\_y.append(train\_seqs[i + window\_size].squeeze())
80. test\_seqs = self.data[len\_val:]
81. for i in range(test\_seqs.shape[0] - window\_size):
82. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
83. test\_y.append(test\_seqs[i + window\_size].squeeze())
84. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
85. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
86. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
87. return train\_set, val\_set, test\_set
88. batch\_size = 64
89. TrafficData = TrafficDataset()
90. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
91. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
94. print(train\_loader)
95. print(val\_loader)
96. print(test\_loader)
97. *###########前面是数据读取部分，已完成勿动###########*
98. *###########前面是数据读取部分，已完成勿动###########*
99. *# 测试函数（用于分类）*
100. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
101. rmse, mae, mape = 0, 0, 0
102. batch\_count = 0
103. total\_loss = 0.0
104. net.eval()
105. if output\_model is not None:
106. output\_model.eval()
107. for X, Y in data\_iter:
108. X = X.to(device).float()
109. Y = Y.to(device).float()
110. output,hidden = net(X)
111. if output\_model is not None:
112. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
113. else:
114. y\_hat = output[:, -1, :].squeeze(-1)
115. loss = loss\_fn(y\_hat, Y)
116. Y = denormalize\_fn(Y)
117. y\_hat = denormalize\_fn(y\_hat)
118. a, b, c = eval(Y.detach(), y\_hat.detach())
119. rmse += a
120. mae += b
121. mape += c
122. total\_loss += loss.detach().cpu().numpy().tolist()
123. batch\_count += 1
124. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
125. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
126. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
127. train\_loss\_lst = []
128. val\_loss\_lst = []
129. train\_score\_lst = []
130. val\_score\_lst = []
131. epoch\_time = []
132. best\_epoch = 0
133. best\_val\_rmse = 9999
134. early\_stop\_flag = 0
135. for epoch in range(num\_epoch):
136. net.train()
137. if output\_model is not None:
138. output\_model.train()
139. epoch\_loss = 0
140. batch\_count = 0
141. batch\_time = []
142. rmse, mae, mape = 0, 0, 0
143. for X, Y in train\_iter:
144. batch\_s = time.time()
145. X = X.to(device).float()
146. Y = Y.to(device).float()
147. optimizer.zero\_grad()
148. output, hidden = net(X)
149. if output\_model is not None:
150. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
151. else:
152. y\_hat = output[:, -1, :].squeeze(-1)
153. loss = loss\_fn(y\_hat, Y)
154. loss.backward()
155. optimizer.step()
156. Y = denormalize\_fn(Y)
157. y\_hat = denormalize\_fn(y\_hat)
158. a, b, c = eval(Y.detach(), y\_hat.detach())
159. rmse += a
160. mae += b
161. mape += c
162. epoch\_loss += loss.detach().cpu().numpy().tolist()
163. batch\_count += 1
164. *# sample\_num += X.shape[0]*
165. batch\_time.append(time.time() - batch\_s)
166. if is\_print and is\_print\_batch:
167. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
168. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
169. train\_loss = epoch\_loss / batch\_count
170. train\_loss\_lst.append(train\_loss)
171. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
172. *# 验证集*
173. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
174. val\_score\_lst.append(val\_score)
175. val\_loss\_lst.append(val\_loss)
176. epoch\_time.append(np.array(batch\_time).sum())
177. *# 打印本轮训练结果*
178. if is\_print:
179. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
180. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
181. *# 早停*
182. if val\_score[0] < best\_val\_rmse:
183. best\_val\_rmse = val\_score[0]
184. best\_epoch = epoch
185. early\_stop\_flag = 0
186. else:
187. early\_stop\_flag += 1
188. if early\_stop\_flag == early\_stop:
189. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
190. break
191. *# 输出最终训练结果*
192. print(f'\n{"\*" \* 40}\nFinal result:')
193. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
194. f'at epoch {best\_epoch}')
195. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
196. print()
197. *# 计算测试集效果*
198. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
199. print('Test result:')
200. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
201. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
202. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
203. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
204. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
205. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
206. plt.xlabel(x\_label)
207. plt.ylabel(y\_label)
208. plt.legend()
209. plt.show()
210. def plot\_metric(score\_log):
211. score\_log = np.array(score\_log)
212. plt.figure(figsize=(10, 6), dpi=300)
213. plt.subplot(2, 2, 1)
214. plt.plot(score\_log[:, 0], c='#d28ad4')
215. plt.ylabel('RMSE')
216. plt.subplot(2, 2, 2)
217. plt.plot(score\_log[:, 1], c='#e765eb')
218. plt.ylabel('MAE')
219. plt.subplot(2, 2, 3)
220. plt.plot(score\_log[:, 2], c='#6b016d')
221. plt.ylabel('MAPE')
222. plt.show()
223. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
224. input\_size = train\_set.X.shape[-1]
225. hidden\_size = 32
226. output\_size = 1
227. seq\_len = 12
228. lr = 0.0001
229. epochs = 80
230. loss\_func = nn.MSELoss()
231. nn\_rnn = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=1, batch\_first=True).to(device)
232. out\_linear = nn.Sequential(nn.Linear(hidden\_size, 1),
233. nn.LeakyReLU()).to(device)
234. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()), lr)
235. train\_loss\_lst, val\_loss\_lst, \
236. train\_score\_lst, val\_score\_lst, stop\_epoch = train(nn\_rnn, train\_loader, val\_loader, test\_loader,
237. loss\_func, TrafficData.denormalize, optimizer, epochs,
238. early\_stop=20, device=device, output\_model=out\_linear)
239. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
240. plot\_metric(train\_score\_lst)

**4.4.2** 在LSTM上进行不同超参数的对比分析（batch\_size）

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.nn as nn
6. import torch.utils.data as data
7. import warnings
8. from torch.utils.data import DataLoader,Dataset
9. warnings.filterwarnings("ignore")
10. import random
11. from IPython import display
12. from matplotlib import pyplot as plt
13. from PIL import Image
14. import os
15. from torch import nn
16. import torch.optim as optim
17. from torch.nn import init
18. import torch.nn.functional as F
19. import time
20. import pandas as pd
21. from sklearn.utils import shuffle
22. import math
23. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
24. import datasets
25. import utils
26. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
27. def mape\_fn(y, pred):
28. mask = y != 0
29. y = y[mask]
30. pred = pred[mask]
31. mape = np.abs((y - pred) / y)
32. mape = np.mean(mape) \* 100
33. return mape
34. def eval(y, pred):
35. y = y.cpu().numpy()
36. pred = pred.cpu().numpy()
37. mse = mse\_fn(y, pred)
38. rmse = math.sqrt(mse)
39. mae = mae\_fn(y, pred)
40. mape = mape\_fn(y, pred)
41. return [rmse, mae, mape]
42. *# 定义dataset*
43. class my\_Dataset(data.Dataset):
44. def \_\_init\_\_(self, features, labels):
45. self.X = features
46. self.y = labels
47. def \_\_getitem\_\_(self, index):
48. return self.X[index], self.y[index]
49. def \_\_len\_\_(self):
50. return self.X.shape[0]
52. class TrafficDataset:
53. def \_\_init\_\_(self):
54. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
55. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
56. *# 数据标准化*
57. self.min = self.raw\_data.min()
58. self.max = self.raw\_data.max()
59. self.data = (self.raw\_data - self.min) / (self.max - self.min)
60. def denormalize(self, x):
61. return x \* (self.max - self.min) + self.min
62. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
63. train\_x = []
64. train\_y = []
65. val\_x = []
66. val\_y = []
67. test\_x = []
68. test\_y = []
69. window\_size = 12
70. len\_train = int(self.data.shape[0] \* 0.6)
71. train\_seqs = self.data[:len\_train]
72. for i in range(train\_seqs.shape[0] - window\_size):
73. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
74. train\_y.append(train\_seqs[i + window\_size].squeeze())
75. len\_val = int(self.data.shape[0] \* 0.8)
76. val\_seqs = self.data[len\_train:len\_val]
77. for i in range(val\_seqs.shape[0] - window\_size):
78. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
79. val\_y.append(train\_seqs[i + window\_size].squeeze())
80. test\_seqs = self.data[len\_val:]
81. for i in range(test\_seqs.shape[0] - window\_size):
82. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
83. test\_y.append(test\_seqs[i + window\_size].squeeze())
84. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
85. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
86. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
87. return train\_set, val\_set, test\_set
88. batch\_size = 128
89. TrafficData = TrafficDataset()
90. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
91. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
94. print(train\_loader)
95. print(val\_loader)
96. print(test\_loader)
97. *###########前面是数据读取部分，已完成勿动###########*
98. *###########前面是数据读取部分，已完成勿动###########*
99. *# 测试函数（用于分类）*
100. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
101. rmse, mae, mape = 0, 0, 0
102. batch\_count = 0
103. total\_loss = 0.0
104. net.eval()
105. if output\_model is not None:
106. output\_model.eval()
107. for X, Y in data\_iter:
108. X = X.to(device).float()
109. Y = Y.to(device).float()
110. output,hidden = net(X)
111. if output\_model is not None:
112. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
113. else:
114. y\_hat = output[:, -1, :].squeeze(-1)
115. loss = loss\_fn(y\_hat, Y)
116. Y = denormalize\_fn(Y)
117. y\_hat = denormalize\_fn(y\_hat)
118. a, b, c = eval(Y.detach(), y\_hat.detach())
119. rmse += a
120. mae += b
121. mape += c
122. total\_loss += loss.detach().cpu().numpy().tolist()
123. batch\_count += 1
124. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
125. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
126. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
127. train\_loss\_lst = []
128. val\_loss\_lst = []
129. train\_score\_lst = []
130. val\_score\_lst = []
131. epoch\_time = []
132. best\_epoch = 0
133. best\_val\_rmse = 9999
134. early\_stop\_flag = 0
135. for epoch in range(num\_epoch):
136. net.train()
137. if output\_model is not None:
138. output\_model.train()
139. epoch\_loss = 0
140. batch\_count = 0
141. batch\_time = []
142. rmse, mae, mape = 0, 0, 0
143. for X, Y in train\_iter:
144. batch\_s = time.time()
145. X = X.to(device).float()
146. Y = Y.to(device).float()
147. optimizer.zero\_grad()
148. output, hidden = net(X)
149. if output\_model is not None:
150. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
151. else:
152. y\_hat = output[:, -1, :].squeeze(-1)
153. loss = loss\_fn(y\_hat, Y)
154. loss.backward()
155. optimizer.step()
156. Y = denormalize\_fn(Y)
157. y\_hat = denormalize\_fn(y\_hat)
158. a, b, c = eval(Y.detach(), y\_hat.detach())
159. rmse += a
160. mae += b
161. mape += c
162. epoch\_loss += loss.detach().cpu().numpy().tolist()
163. batch\_count += 1
164. *# sample\_num += X.shape[0]*
165. batch\_time.append(time.time() - batch\_s)
166. if is\_print and is\_print\_batch:
167. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
168. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
169. train\_loss = epoch\_loss / batch\_count
170. train\_loss\_lst.append(train\_loss)
171. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
172. *# 验证集*
173. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
174. val\_score\_lst.append(val\_score)
175. val\_loss\_lst.append(val\_loss)
176. epoch\_time.append(np.array(batch\_time).sum())
177. *# 打印本轮训练结果*
178. if is\_print:
179. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
180. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
181. *# 早停*
182. if val\_score[0] < best\_val\_rmse:
183. best\_val\_rmse = val\_score[0]
184. best\_epoch = epoch
185. early\_stop\_flag = 0
186. else:
187. early\_stop\_flag += 1
188. if early\_stop\_flag == early\_stop:
189. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
190. break
191. *# 输出最终训练结果*
192. print(f'\n{"\*" \* 40}\nFinal result:')
193. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
194. f'at epoch {best\_epoch}')
195. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
196. print()
197. *# 计算测试集效果*
198. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
199. print('Test result:')
200. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
201. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
202. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
203. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
204. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
205. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
206. plt.xlabel(x\_label)
207. plt.ylabel(y\_label)
208. plt.legend()
209. plt.show()
210. def plot\_metric(score\_log):
211. score\_log = np.array(score\_log)
212. plt.figure(figsize=(10, 6), dpi=300)
213. plt.subplot(2, 2, 1)
214. plt.plot(score\_log[:, 0], c='#d28ad4')
215. plt.ylabel('RMSE')
216. plt.subplot(2, 2, 2)
217. plt.plot(score\_log[:, 1], c='#e765eb')
218. plt.ylabel('MAE')
219. plt.subplot(2, 2, 3)
220. plt.plot(score\_log[:, 2], c='#6b016d')
221. plt.ylabel('MAPE')
222. plt.show()
223. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
224. input\_size = train\_set.X.shape[-1]
225. hidden\_size = 64
226. output\_size = 1
227. seq\_len = 12
228. lr = 0.0001
229. epochs = 80
230. loss\_func = nn.MSELoss()
231. nn\_rnn = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=1, batch\_first=True).to(device)
232. out\_linear = nn.Sequential(nn.Linear(hidden\_size, 1),
233. nn.LeakyReLU()).to(device)
234. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()), lr)
235. train\_loss\_lst, val\_loss\_lst, \
236. train\_score\_lst, val\_score\_lst, stop\_epoch = train(nn\_rnn, train\_loader, val\_loader, test\_loader,
237. loss\_func, TrafficData.denormalize, optimizer, epochs,
238. early\_stop=20, device=device, output\_model=out\_linear)
239. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
240. plot\_metric(train\_score\_lst)

**4.4.3** 在LSTM上进行不同超参数的对比分析（lr）

1. import numpy as np
2. import pandas as pd
3. import torch
4. import torch.nn as nn
5. import torch.nn as nn
6. import torch.utils.data as data
7. import warnings
8. from torch.utils.data import DataLoader,Dataset
9. warnings.filterwarnings("ignore")
10. import random
11. from IPython import display
12. from matplotlib import pyplot as plt
13. from PIL import Image
14. import os
15. from torch import nn
16. import torch.optim as optim
17. from torch.nn import init
18. import torch.nn.functional as F
19. import time
20. import pandas as pd
21. from sklearn.utils import shuffle
22. import math
23. from sklearn.metrics import mean\_squared\_error as mse, mean\_absolute\_error as mae
24. import datasets
25. import utils
26. from sklearn.metrics import mean\_squared\_error as mse\_fn, mean\_absolute\_error as mae\_fn
27. def mape\_fn(y, pred):
28. mask = y != 0
29. y = y[mask]
30. pred = pred[mask]
31. mape = np.abs((y - pred) / y)
32. mape = np.mean(mape) \* 100
33. return mape
34. def eval(y, pred):
35. y = y.cpu().numpy()
36. pred = pred.cpu().numpy()
37. mse = mse\_fn(y, pred)
38. rmse = math.sqrt(mse)
39. mae = mae\_fn(y, pred)
40. mape = mape\_fn(y, pred)
41. return [rmse, mae, mape]
42. *# 定义dataset*
43. class my\_Dataset(data.Dataset):
44. def \_\_init\_\_(self, features, labels):
45. self.X = features
46. self.y = labels
47. def \_\_getitem\_\_(self, index):
48. return self.X[index], self.y[index]
49. def \_\_len\_\_(self):
50. return self.X.shape[0]
52. class TrafficDataset:
53. def \_\_init\_\_(self):
54. self.raw\_data = np.load(r'C:\Users\20693\Desktop\实验4\_22121501\_束天成\dataset\Traffic\Traffic.npz')['data']
55. *# self.raw\_data = pd.DataFrame(self.raw\_data)*
56. *# 数据标准化*
57. self.min = self.raw\_data.min()
58. self.max = self.raw\_data.max()
59. self.data = (self.raw\_data - self.min) / (self.max - self.min)
60. def denormalize(self, x):
61. return x \* (self.max - self.min) + self.min
62. def construct\_set(self, train\_por=0.6, test\_por=0.2, window\_size=12, label=0):
63. train\_x = []
64. train\_y = []
65. val\_x = []
66. val\_y = []
67. test\_x = []
68. test\_y = []
69. window\_size = 12
70. len\_train = int(self.data.shape[0] \* 0.6)
71. train\_seqs = self.data[:len\_train]
72. for i in range(train\_seqs.shape[0] - window\_size):
73. train\_x.append(train\_seqs[i:i + window\_size].squeeze())
74. train\_y.append(train\_seqs[i + window\_size].squeeze())
75. len\_val = int(self.data.shape[0] \* 0.8)
76. val\_seqs = self.data[len\_train:len\_val]
77. for i in range(val\_seqs.shape[0] - window\_size):
78. val\_x.append(train\_seqs[i:i + window\_size].squeeze())
79. val\_y.append(train\_seqs[i + window\_size].squeeze())
80. test\_seqs = self.data[len\_val:]
81. for i in range(test\_seqs.shape[0] - window\_size):
82. test\_x.append(test\_seqs[i:i + window\_size].squeeze())
83. test\_y.append(test\_seqs[i + window\_size].squeeze())
84. train\_set = my\_Dataset(torch.Tensor(train\_x).unsqueeze(-1), torch.Tensor(train\_y))
85. val\_set = my\_Dataset(torch.Tensor(val\_x).unsqueeze(-1), torch.Tensor(val\_y))
86. test\_set = my\_Dataset(torch.Tensor(test\_x).unsqueeze(-1), torch.Tensor(test\_y))
87. return train\_set, val\_set, test\_set
88. batch\_size = 64
89. TrafficData = TrafficDataset()
90. train\_set,val\_set,test\_set = TrafficData.construct\_set(0.6,0.2,12)
91. train\_loader = data.DataLoader(train\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
92. val\_loader = data.DataLoader(val\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
93. test\_loader = data.DataLoader(test\_set,batch\_size=batch\_size,shuffle=True,num\_workers=0,drop\_last=True)
94. print(train\_loader)
95. print(val\_loader)
96. print(test\_loader)
97. *###########前面是数据读取部分，已完成勿动###########*
98. *###########前面是数据读取部分，已完成勿动###########*
99. *# 测试函数（用于分类）*
100. def test(net, output\_model, data\_iter, loss\_fn, denormalize\_fn, device='cpu'):
101. rmse, mae, mape = 0, 0, 0
102. batch\_count = 0
103. total\_loss = 0.0
104. net.eval()
105. if output\_model is not None:
106. output\_model.eval()
107. for X, Y in data\_iter:
108. X = X.to(device).float()
109. Y = Y.to(device).float()
110. output,hidden = net(X)
111. if output\_model is not None:
112. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze(-1)
113. else:
114. y\_hat = output[:, -1, :].squeeze(-1)
115. loss = loss\_fn(y\_hat, Y)
116. Y = denormalize\_fn(Y)
117. y\_hat = denormalize\_fn(y\_hat)
118. a, b, c = eval(Y.detach(), y\_hat.detach())
119. rmse += a
120. mae += b
121. mape += c
122. total\_loss += loss.detach().cpu().numpy().tolist()
123. batch\_count += 1
124. return [rmse / batch\_count, mae / batch\_count, mape / batch\_count], total\_loss / batch\_count
125. def train(net, train\_iter, val\_iter, test\_iter, loss\_fn, denormalize\_fn, optimizer, num\_epoch,
126. early\_stop=10, device='cpu', output\_model=None, is\_print=True, is\_print\_batch=False):
127. train\_loss\_lst = []
128. val\_loss\_lst = []
129. train\_score\_lst = []
130. val\_score\_lst = []
131. epoch\_time = []
132. best\_epoch = 0
133. best\_val\_rmse = 9999
134. early\_stop\_flag = 0
135. for epoch in range(num\_epoch):
136. net.train()
137. if output\_model is not None:
138. output\_model.train()
139. epoch\_loss = 0
140. batch\_count = 0
141. batch\_time = []
142. rmse, mae, mape = 0, 0, 0
143. for X, Y in train\_iter:
144. batch\_s = time.time()
145. X = X.to(device).float()
146. Y = Y.to(device).float()
147. optimizer.zero\_grad()
148. output, hidden = net(X)
149. if output\_model is not None:
150. y\_hat = output\_model(output[:, -1, :].squeeze(-1)).squeeze()
151. else:
152. y\_hat = output[:, -1, :].squeeze(-1)
153. loss = loss\_fn(y\_hat, Y)
154. loss.backward()
155. optimizer.step()
156. Y = denormalize\_fn(Y)
157. y\_hat = denormalize\_fn(y\_hat)
158. a, b, c = eval(Y.detach(), y\_hat.detach())
159. rmse += a
160. mae += b
161. mape += c
162. epoch\_loss += loss.detach().cpu().numpy().tolist()
163. batch\_count += 1
164. *# sample\_num += X.shape[0]*
165. batch\_time.append(time.time() - batch\_s)
166. if is\_print and is\_print\_batch:
167. print('epoch-batch: %d-%d, train loss %.4f, time use %.3fs' %
168. (epoch + 1, batch\_count, epoch\_loss, batch\_time[-1]))
169. train\_loss = epoch\_loss / batch\_count
170. train\_loss\_lst.append(train\_loss)
171. train\_score\_lst.append([rmse/batch\_count, mae/batch\_count, mape/batch\_count])
172. *# 验证集*
173. val\_score, val\_loss = test(net, output\_model, val\_iter, loss\_fn, denormalize\_fn, device)
174. val\_score\_lst.append(val\_score)
175. val\_loss\_lst.append(val\_loss)
176. epoch\_time.append(np.array(batch\_time).sum())
177. *# 打印本轮训练结果*
178. if is\_print:
179. print('\*\*\* epoch%d, train loss %.4f, train rmse %.4f, val loss %.4f, val rmse %.6f, time use %.3fs' %
180. (epoch + 1, train\_loss, train\_score\_lst[-1][0], val\_loss, val\_score[0], epoch\_time[-1]))
181. *# 早停*
182. if val\_score[0] < best\_val\_rmse:
183. best\_val\_rmse = val\_score[0]
184. best\_epoch = epoch
185. early\_stop\_flag = 0
186. else:
187. early\_stop\_flag += 1
188. if early\_stop\_flag == early\_stop:
189. print(f'\nThe model has not been improved for {early\_stop} rounds. Stop early!')
190. break
191. *# 输出最终训练结果*
192. print(f'\n{"\*" \* 40}\nFinal result:')
193. print(f'Get best validation rmse {np.array(val\_score\_lst)[:, 0].min() :.4f} '
194. f'at epoch {best\_epoch}')
195. print(f'Total time {np.array(epoch\_time).sum():.2f}s')
196. print()
197. *# 计算测试集效果*
198. test\_score, test\_loss = test(net, output\_model, test\_iter, loss\_fn, denormalize\_fn, device)
199. print('Test result:')
200. print(f'Test RMSE: {test\_score[0]}    Test MAE: {test\_score[1]}    Test MAPE: {test\_score[2]}')
201. return train\_loss\_lst, val\_loss\_lst, train\_score\_lst, val\_score\_lst, epoch
202. def visualize(num\_epochs, train\_data, test\_data, x\_label='epoch', y\_label='loss'):
203. x = np.arange(0, num\_epochs + 1).astype(dtype=np.int)
204. plt.plot(x, train\_data, label=f"train\_{y\_label}", linewidth=1.5)
205. plt.plot(x, test\_data, label=f"val\_{y\_label}", linewidth=1.5)
206. plt.xlabel(x\_label)
207. plt.ylabel(y\_label)
208. plt.legend()
209. plt.show()
210. def plot\_metric(score\_log):
211. score\_log = np.array(score\_log)
212. plt.figure(figsize=(10, 6), dpi=300)
213. plt.subplot(2, 2, 1)
214. plt.plot(score\_log[:, 0], c='#d28ad4')
215. plt.ylabel('RMSE')
216. plt.subplot(2, 2, 2)
217. plt.plot(score\_log[:, 1], c='#e765eb')
218. plt.ylabel('MAE')
219. plt.subplot(2, 2, 3)
220. plt.plot(score\_log[:, 2], c='#6b016d')
221. plt.ylabel('MAPE')
222. plt.show()
223. device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")
224. input\_size = train\_set.X.shape[-1]
225. hidden\_size = 64
226. output\_size = 1
227. seq\_len = 12
228. lr = 0.001
229. epochs = 80
230. loss\_func = nn.MSELoss()
231. nn\_rnn = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, num\_layers=1, batch\_first=True).to(device)
232. out\_linear = nn.Sequential(nn.Linear(hidden\_size, 1),
233. nn.LeakyReLU()).to(device)
234. optimizer = torch.optim.Adam(list(nn\_rnn.parameters()) + list(out\_linear.parameters()), lr)
235. train\_loss\_lst, val\_loss\_lst, \
236. train\_score\_lst, val\_score\_lst, stop\_epoch = train(nn\_rnn, train\_loader, val\_loader, test\_loader,
237. loss\_func, TrafficData.denormalize, optimizer, epochs,
238. early\_stop=20, device=device, output\_model=out\_linear)
239. visualize(stop\_epoch, train\_loss\_lst, val\_loss\_lst, y\_label='Loss')
240. plot\_metric(train\_score\_lst)

## 五、实验结果

实验结果包括程序运行结果以及对结果的分析，尽量用图表展示实验结果，并且通过结果进行相关的分析。

#### 5.1 循环神经网络数据预处理

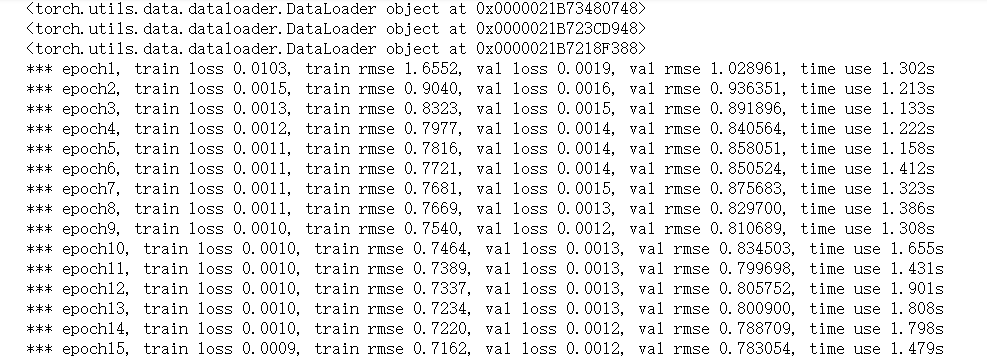
**5.1.1 实验4.1.1运行结果**

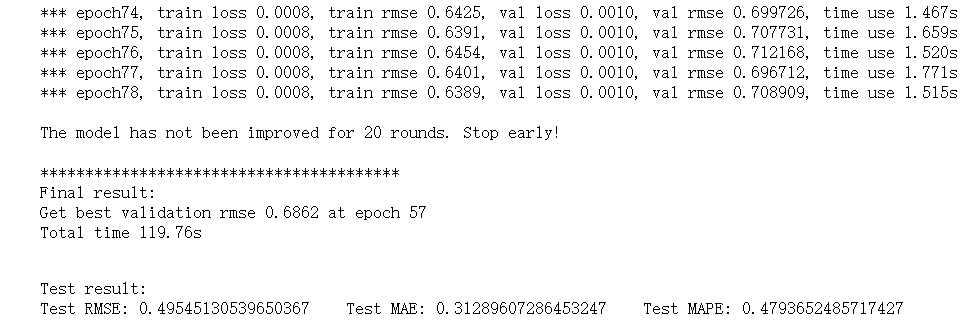


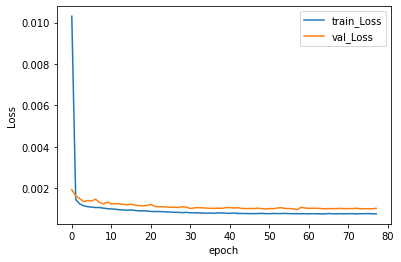
根据实验结果可以得知，在数据采用面向对象的方式进行封装的时候，若需要对数据进行输出，输出的内容以存储于某一存储空间方式的形式输出，这种输出恰好是面向对象语言输出的基本形式，由此可以判断数据的预处理正确。

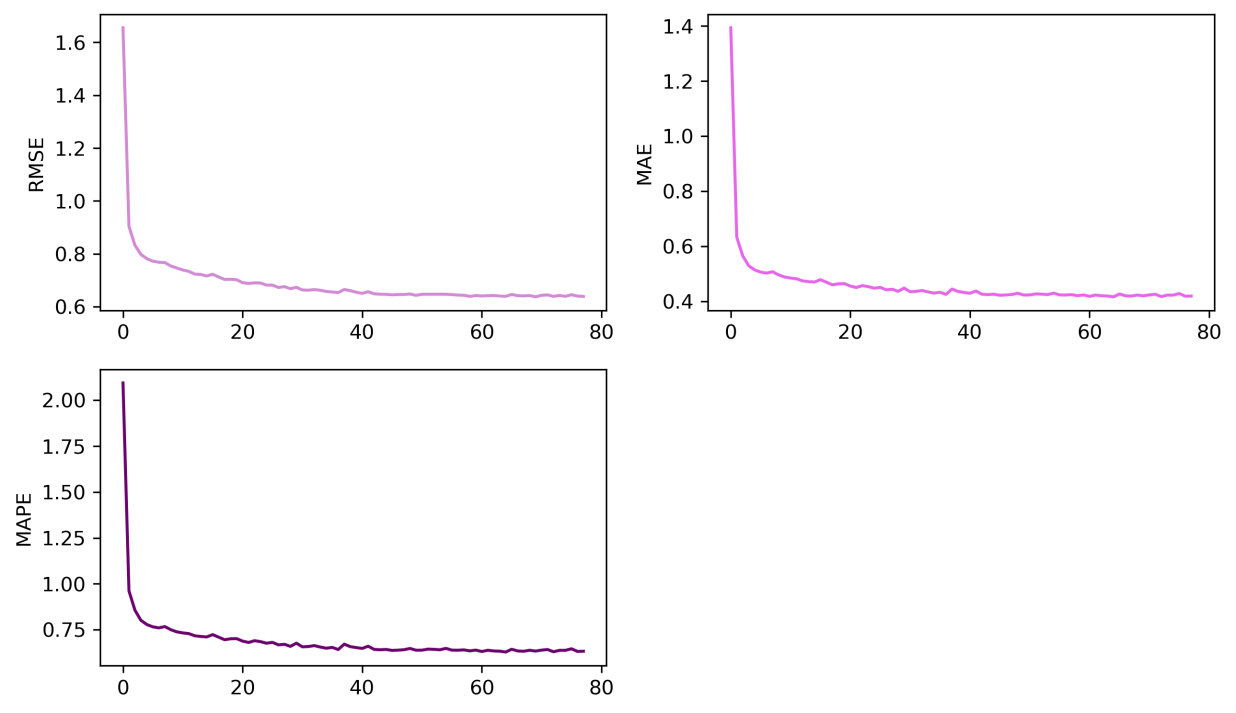
#### 5.2 采用手动方式以及调用接口方式实现RNN、LSTM和GRU

**5.2.1 实验4.2.1运行结果**



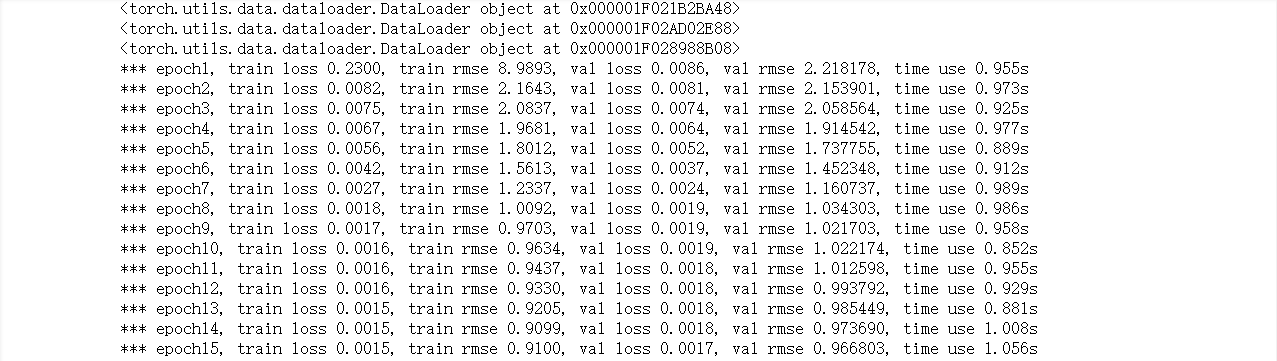


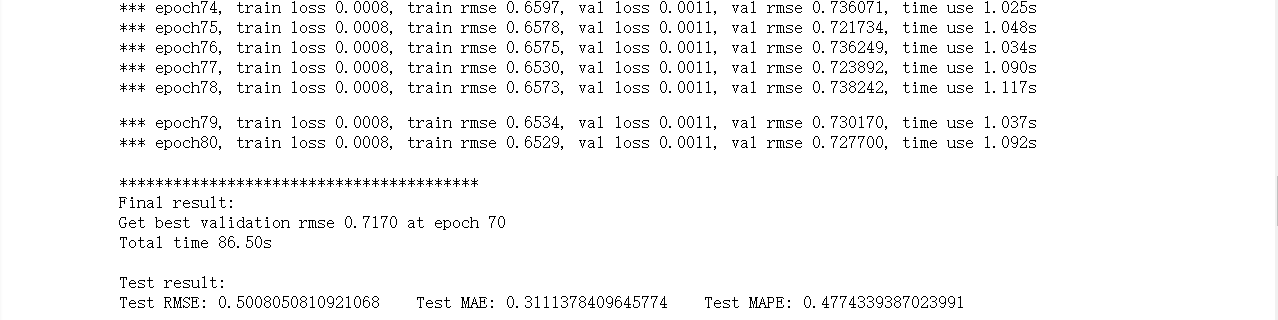


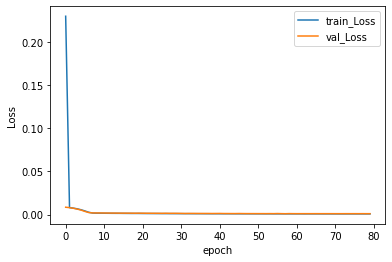


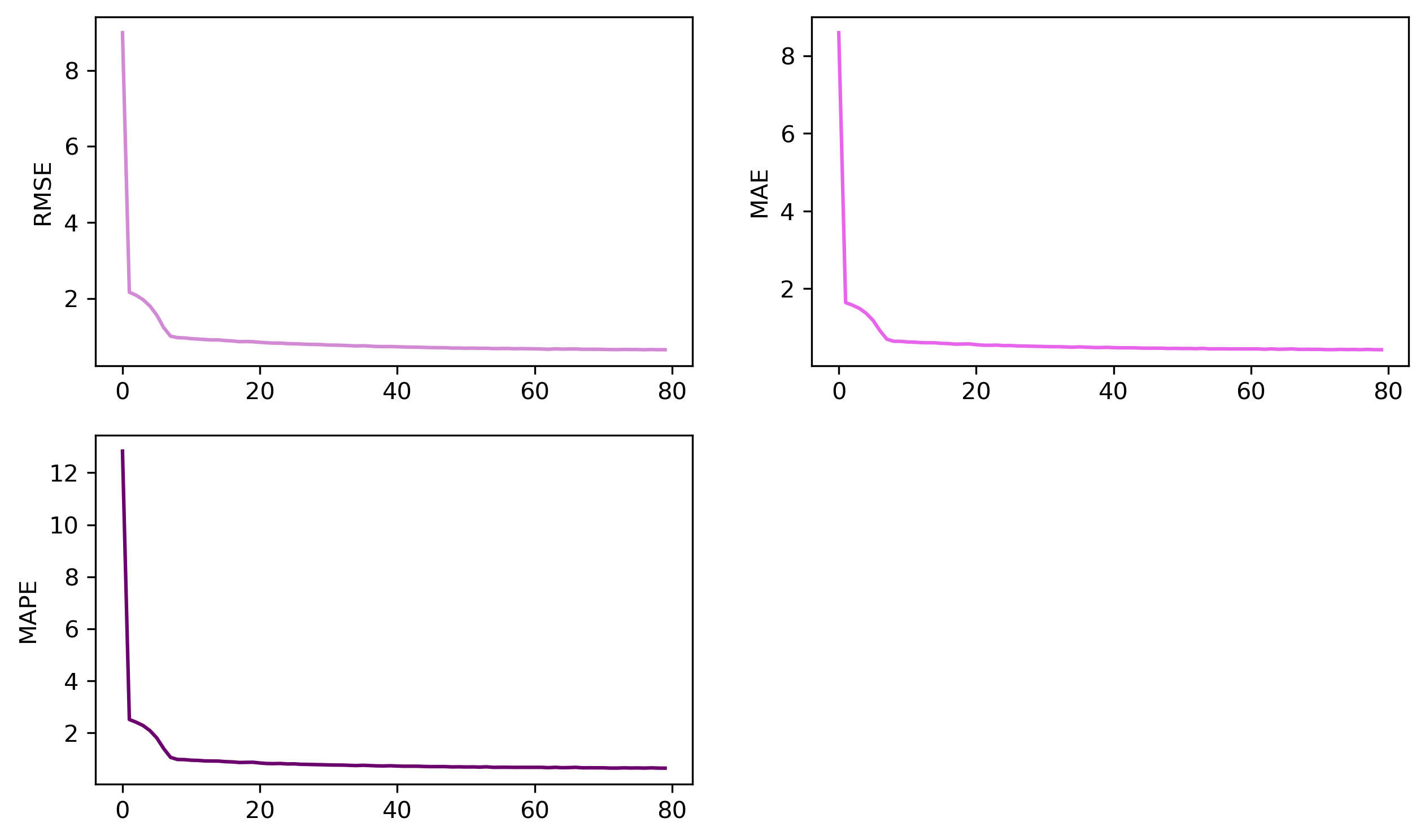
**分析：**这个实验是采用手动方式实现循环神经网络的实验，一共消耗时间119s。根据实验结果来看，实验的训练损失和测试损失基本接近，无过多出入，三个评价指标也可以看到在MAE评估模型下的数值最小，RMSE和MAPE基本上接近，接下来还需要验证使用调用接口方式下模型的好坏。

**5.2.2实验4.2.2运行结果**



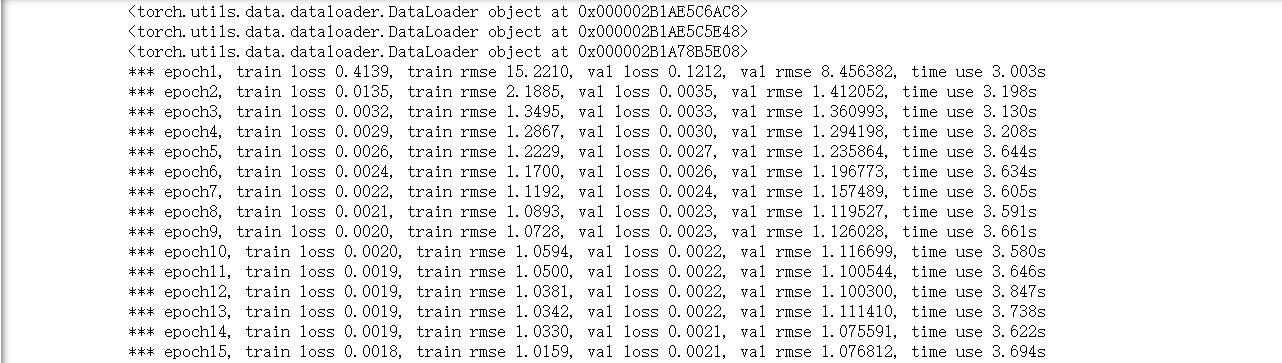


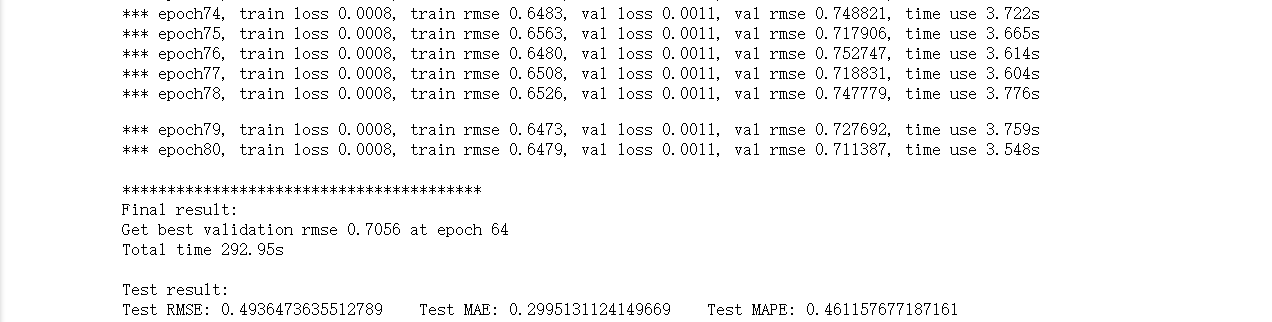


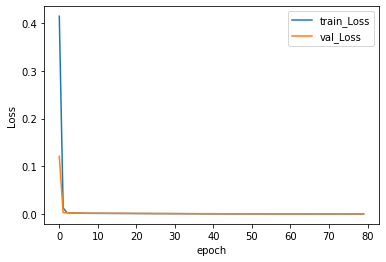


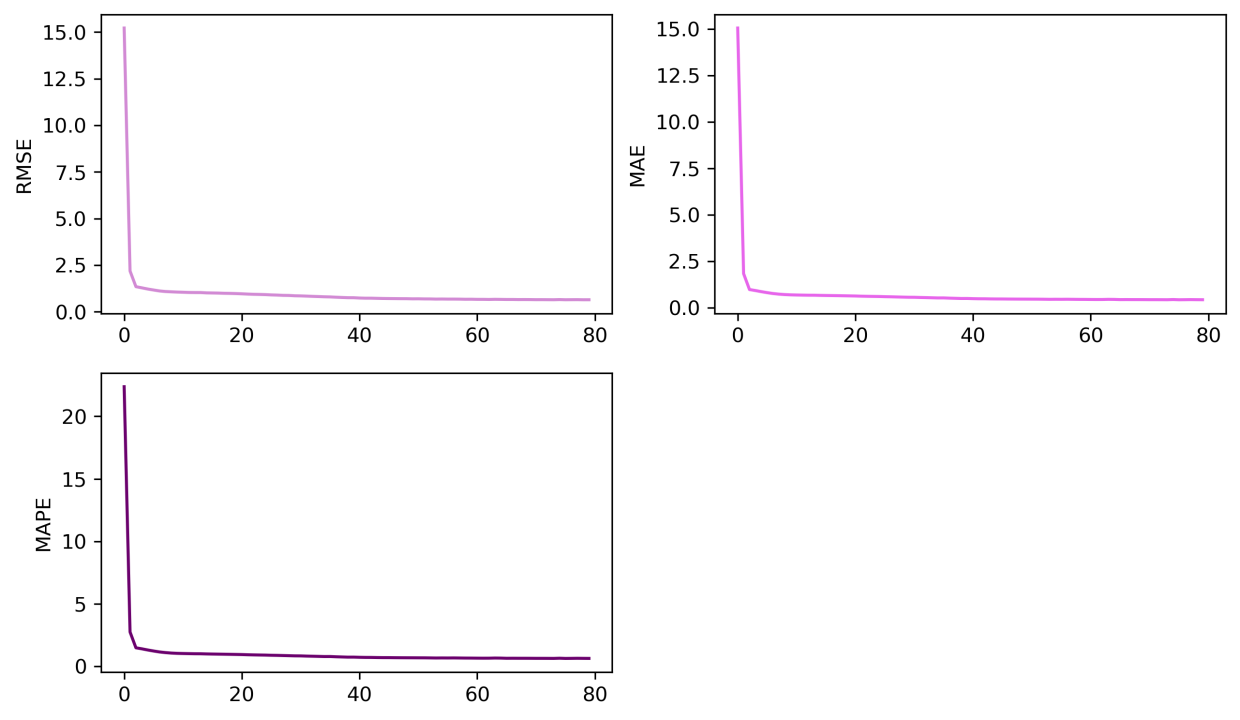
**分析：**这个实验是采用调用接口的方式实现循环神经网络的实验，一共消耗时间86.5s。根据实验结果来看，实验的训练损失和测试损失基本接近，与手动实现的基本相近，无过多出入，三个评价指标也可以看到在MAE评估模型下的数值最小，RMSE和MAPE基本上接近，但是总体数值略大于手动实现，模型在70轮时达到了一个较好地训练效果，因此手动实现的模型略差。

**5.2.3 实验4.2.3运行结果**



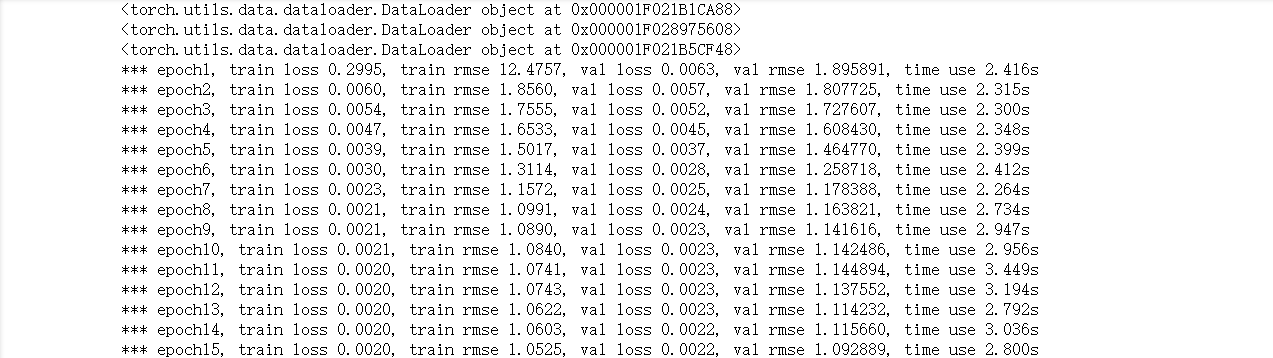


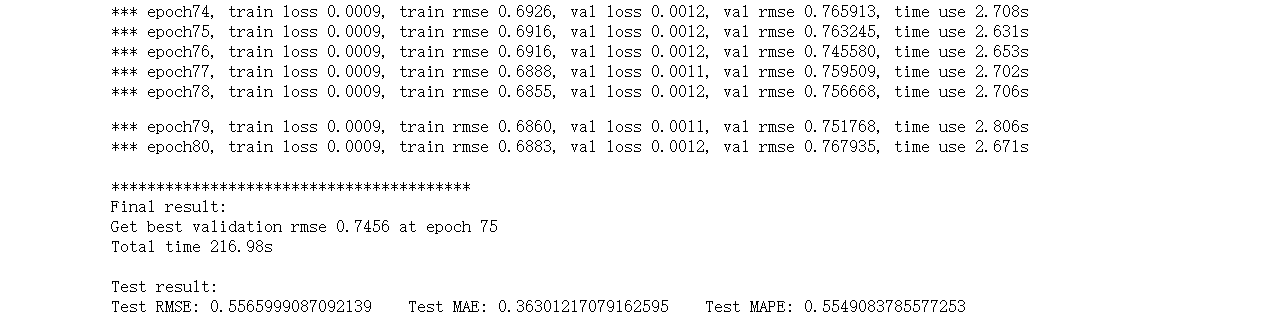


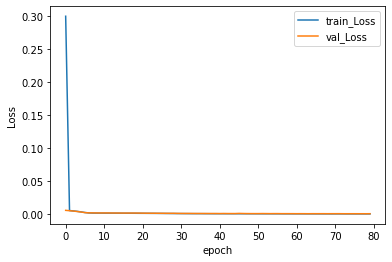


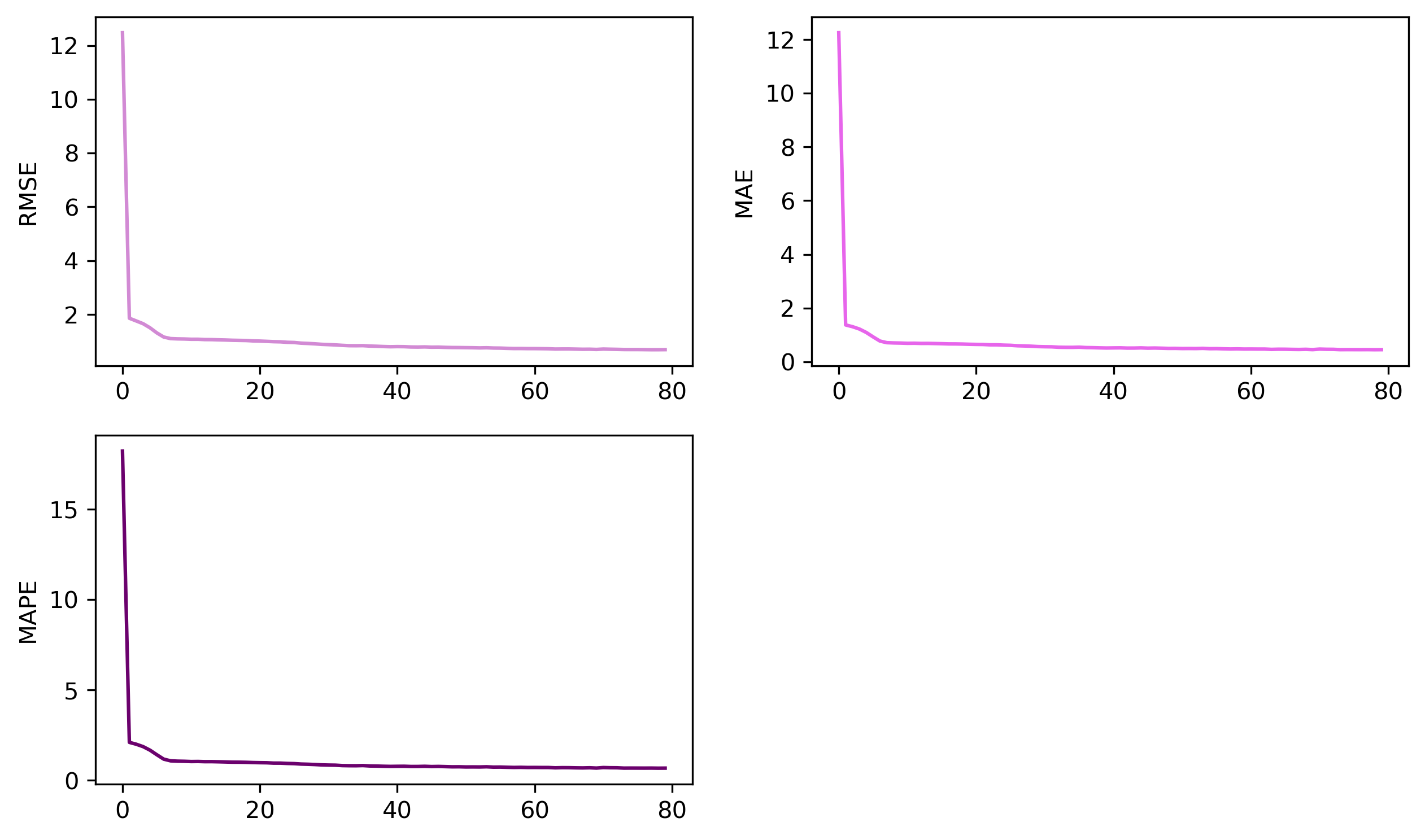
**分析：**这个实验是采用手动的方式采用LSTM实现循环神经网络的实验，一共消耗时间292s，消耗时间长是最大的特点。根据实验结果来看，实验的训练损失和测试损失基本接近，和使用卷积神经网络的方法结果基本相似，三个评价指标也可以看到在MSE，RMSE和MAPE基本上接近，但是总体数值略大于，模型没有提前停止，反而在一定的次数达到较好地训练效果。

**5.2.4 实验4.2.4运行结果**



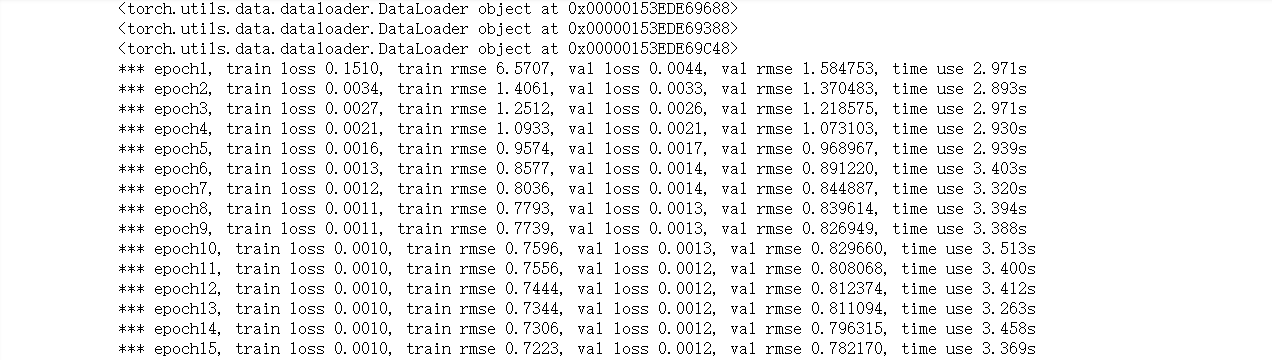


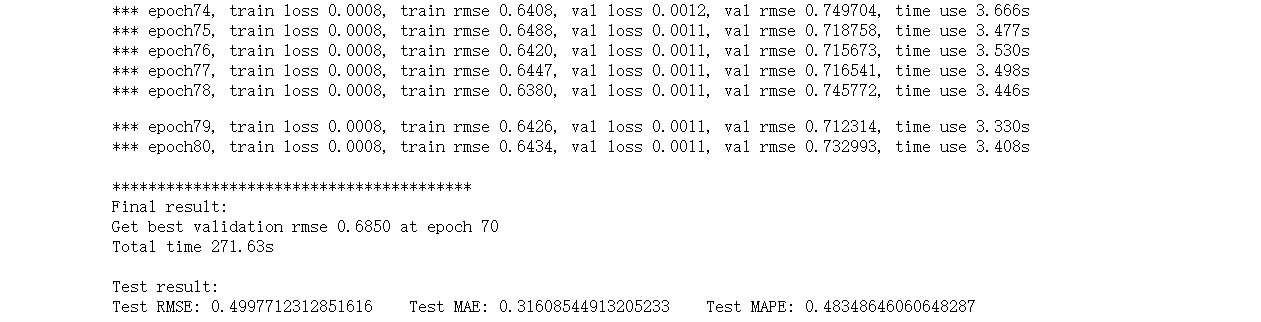


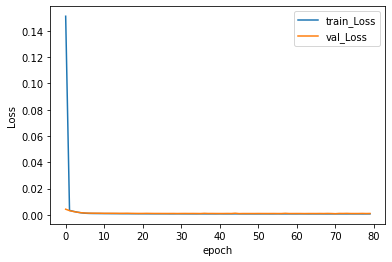


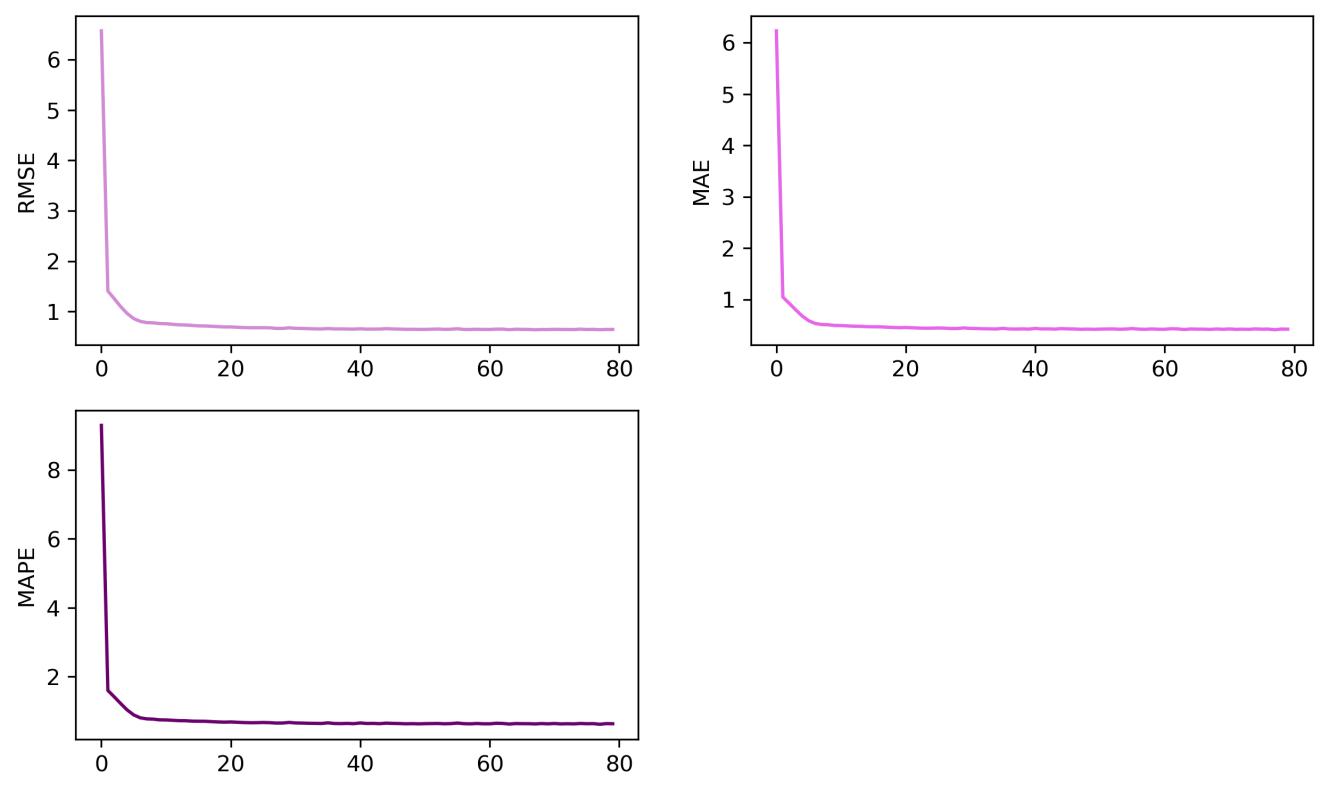
**分析：**这个实验是采用调用接口的方式采用LSTM实现循环神经网络的实验，一共消耗时间216.98s，消耗时间更短是最大的特点，因为调用自带接口省去了许多自定义时不必要的麻烦。根据实验结果来看，实验的训练损失和测试损失基本接近，和使用卷积神经网络的方法结果基本相似，但是略有较小毛刺，三个评价指标也可以看到在MSE，RMSE和MAPE基本上接近，但是总体数值更小，模型没有提前停止，在一定的次数达到较好地训练效果。可以认定接口实现的效果好。

**5.2.5 实验4.2.5运行结果**



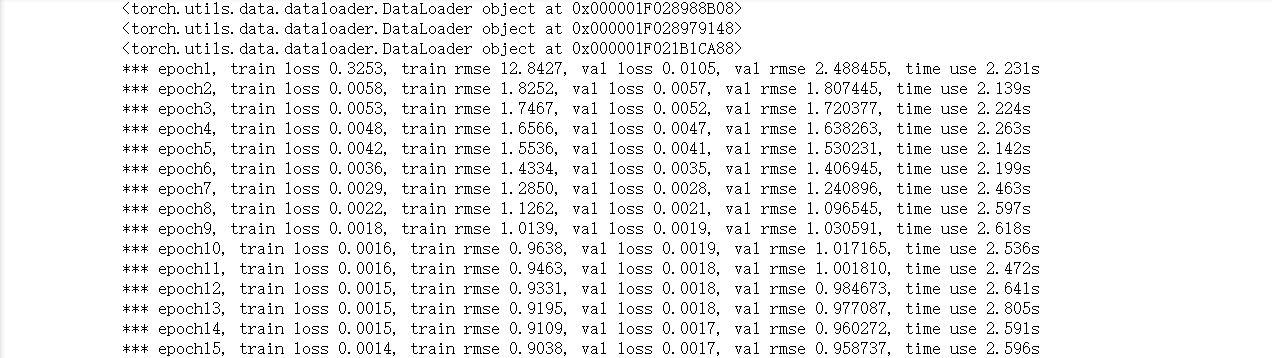


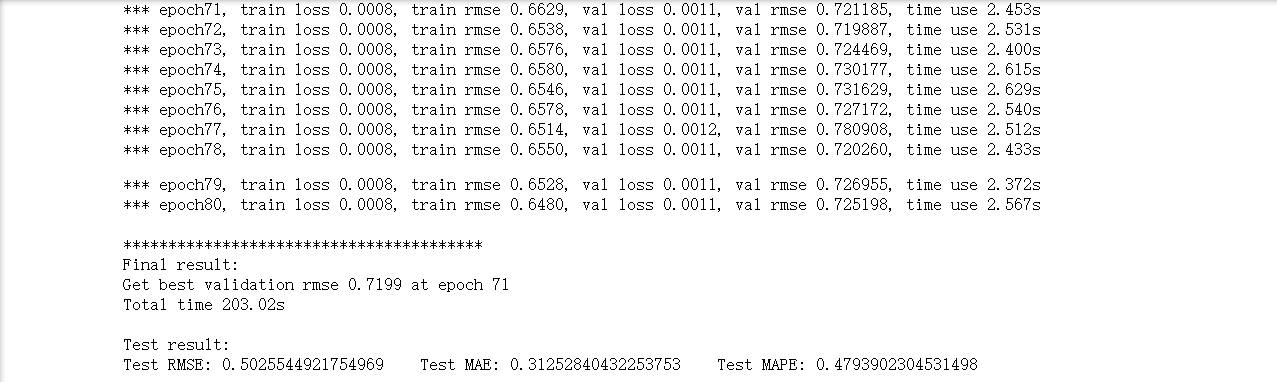


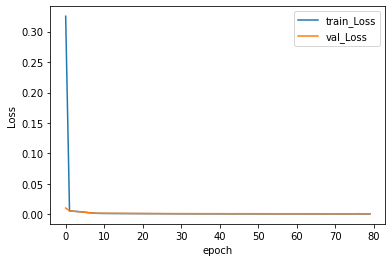


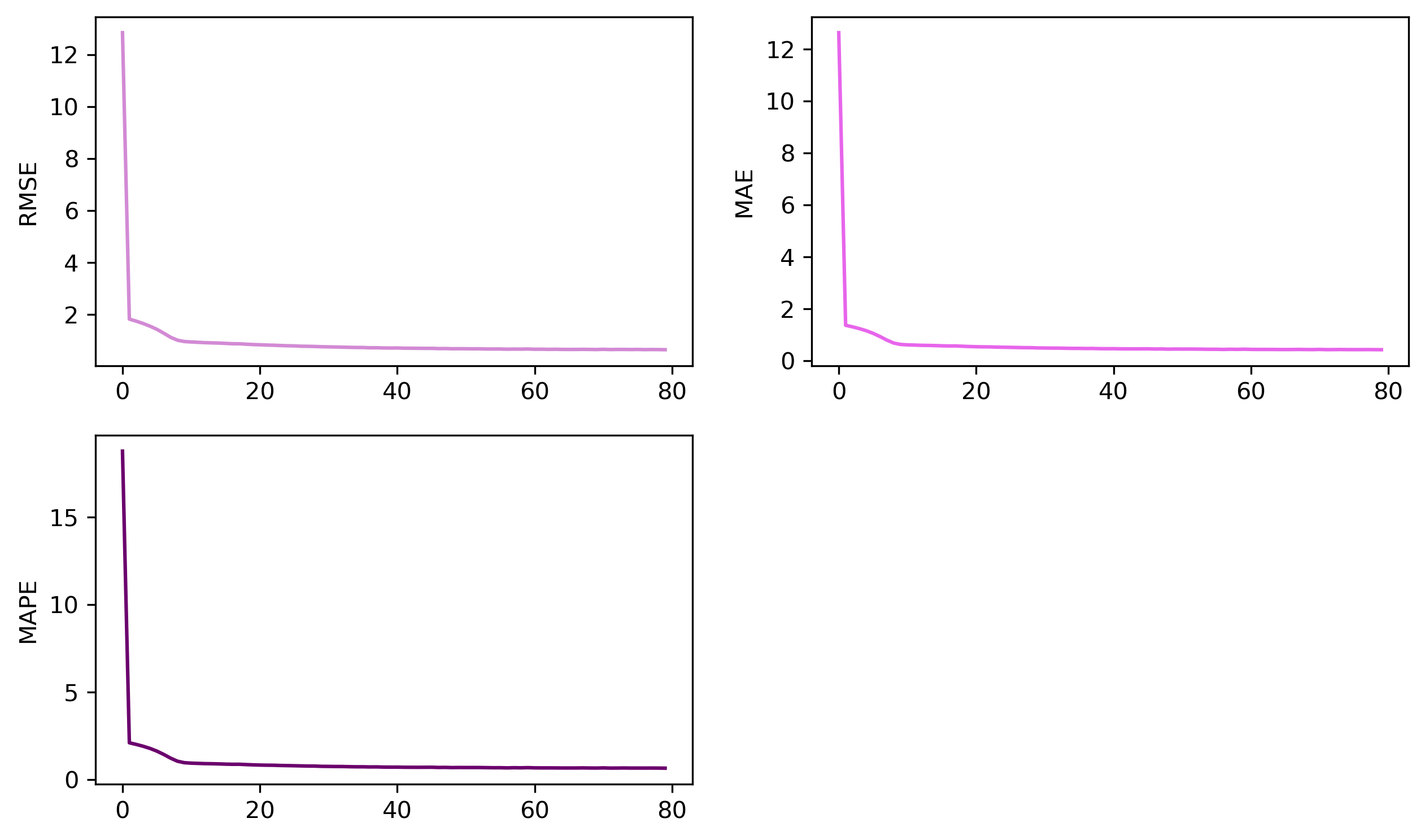
**分析：**这个实验是采用手动的方式采用GRU实现循环神经网络的实验，一共消耗时间271s，消耗时间略短于LSTM网络。根据实验结果来看，实验的训练损失和测试损失基本接近，和使用卷积神经网络的方法结果基本相似，三个评价指标也可以看到在MSE，MAPE基本上接近，模型没有提前停止，在一定的次数达到较好地训练效果，总体看略慢于LSTM网络。

**5.2.6 实验4.2.6运行结果**



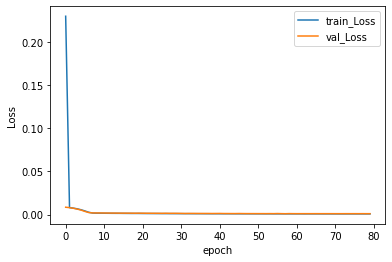


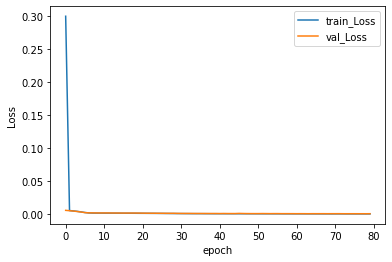


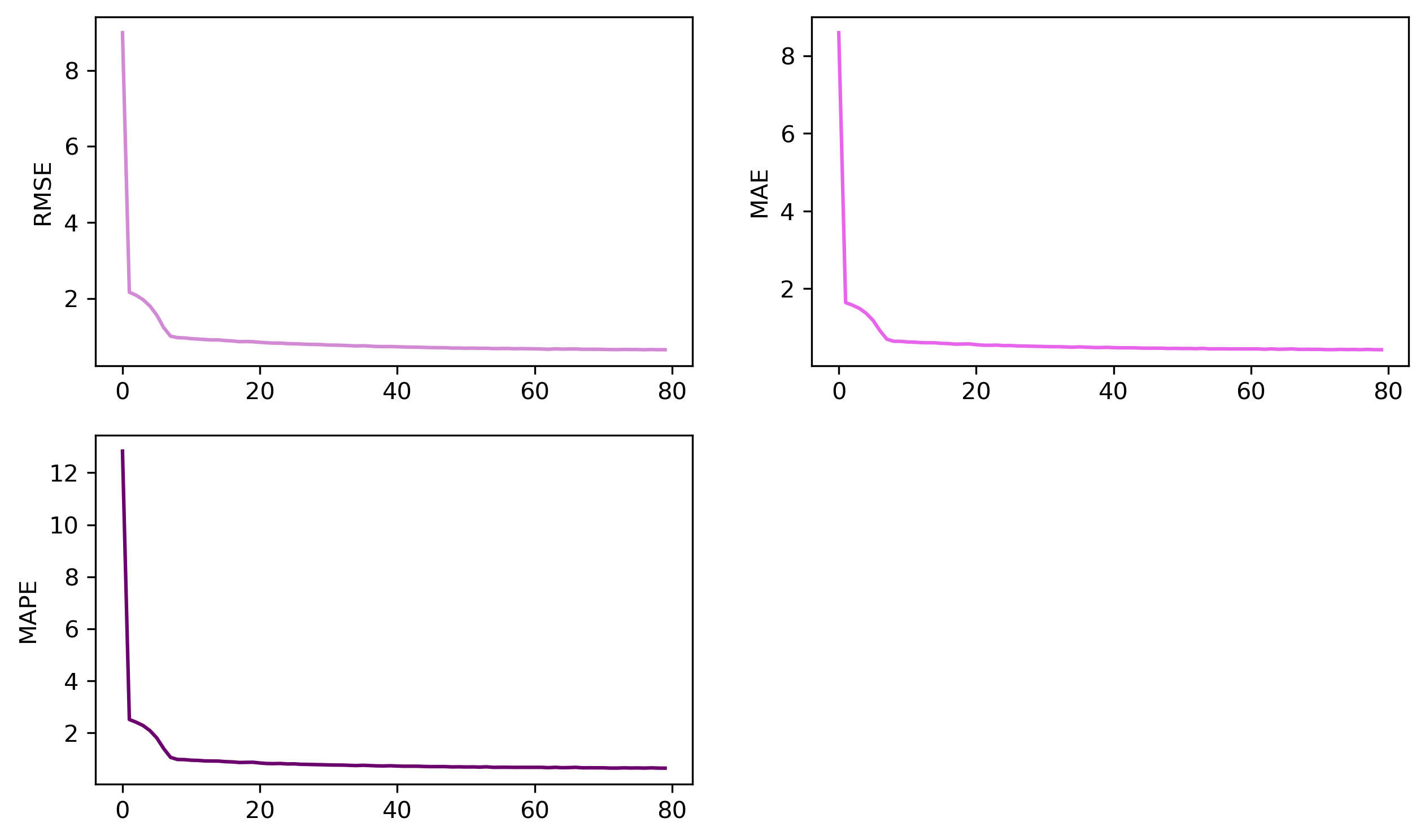
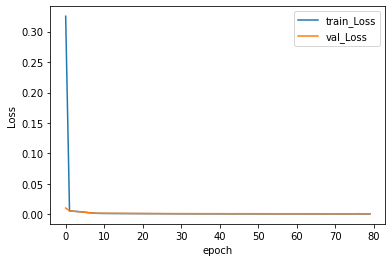


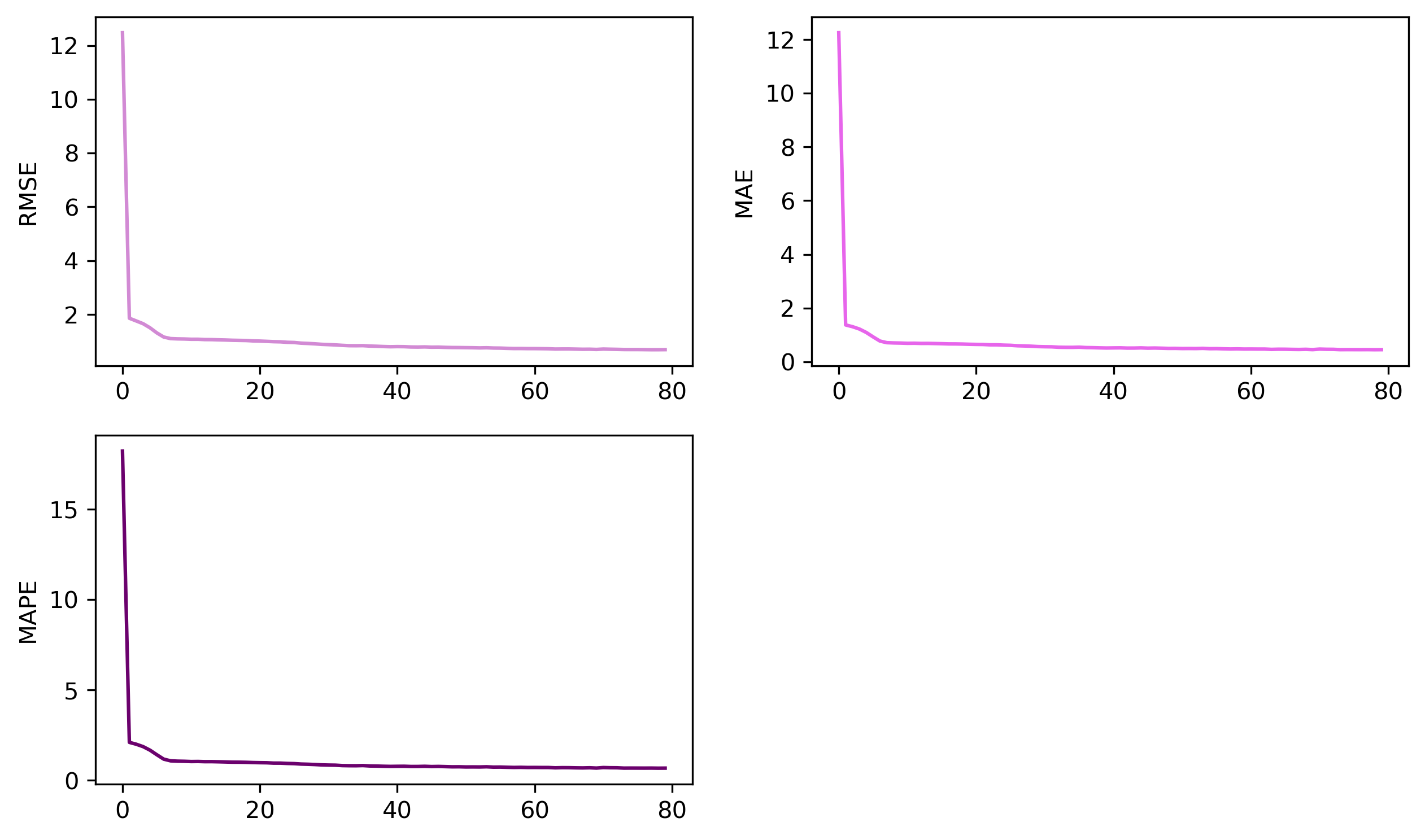
**分析：**这个实验是采用调用接口的方式采用GRU实现循环神经网络的实验，一共消耗时间203.02s，消耗时间短是最大的特点，因为调用自带接口省去了许多自定义时不必要的麻烦。根据实验结果来看，实验的训练损失和测试损失基本接近，和使用卷积神经网络的方法结果基本相似，损失也更加平滑，实验效果也相对更好，三个评价指标也可以看到在MSE，RMSE和MAPE基本上比较小，但是总体数值更小，在71轮次数达到最佳训练效果。可以认定接口实现的效果好。

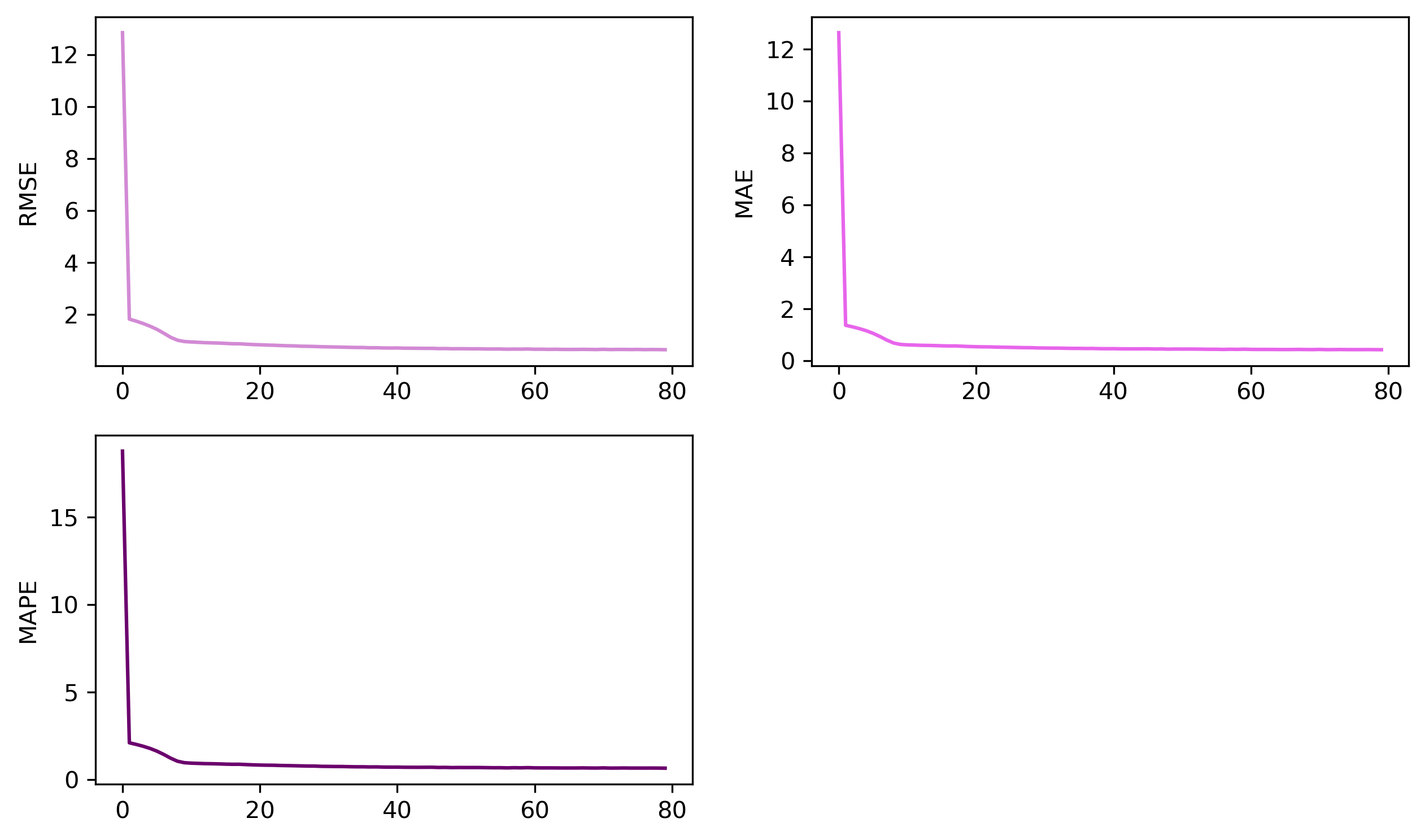
#### 5.3 模型对比





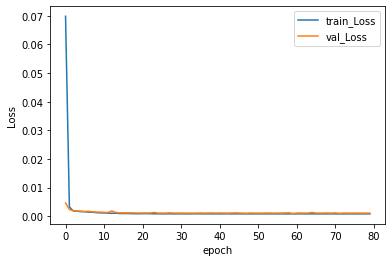
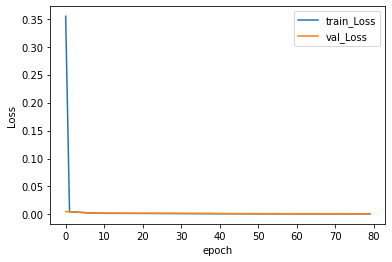


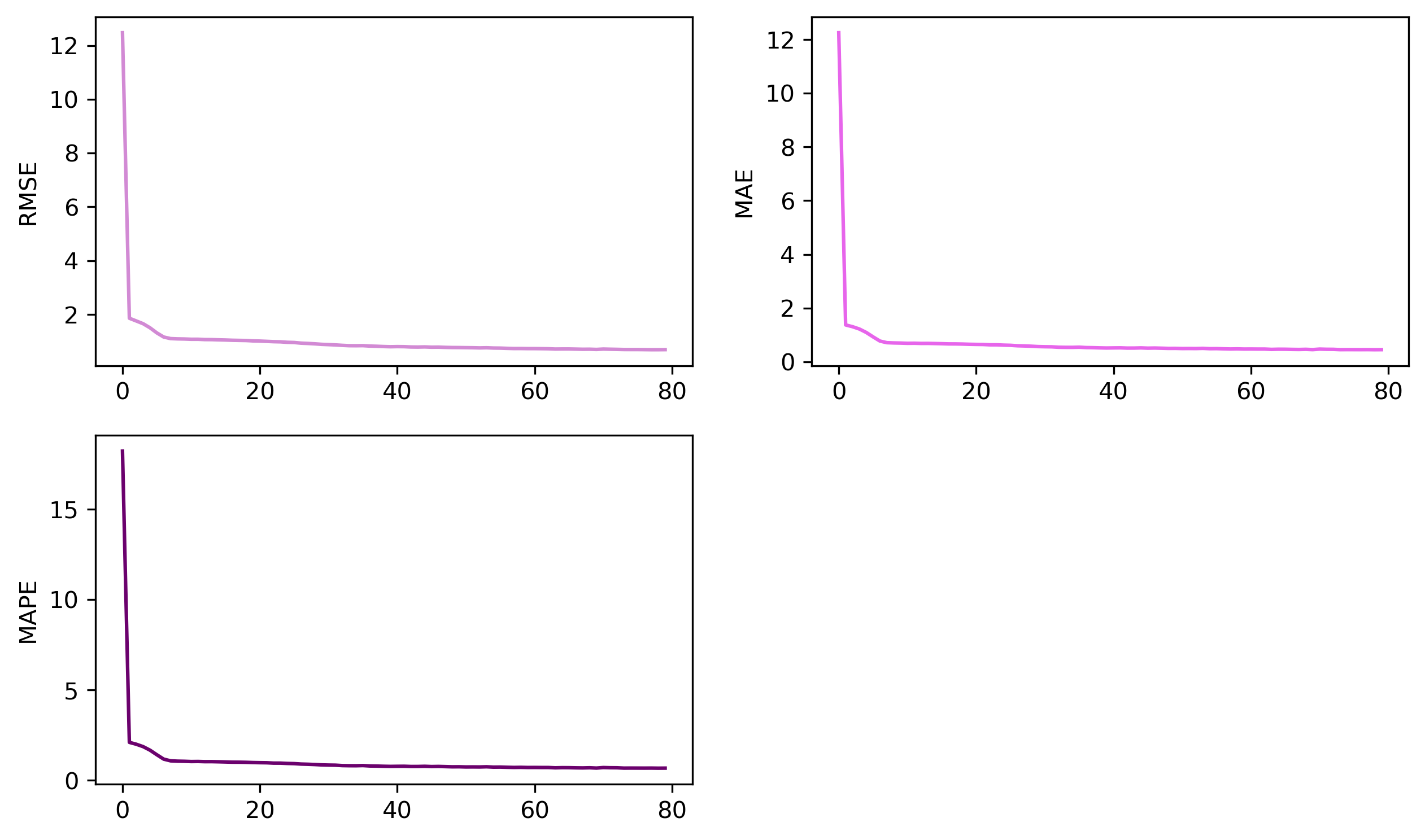


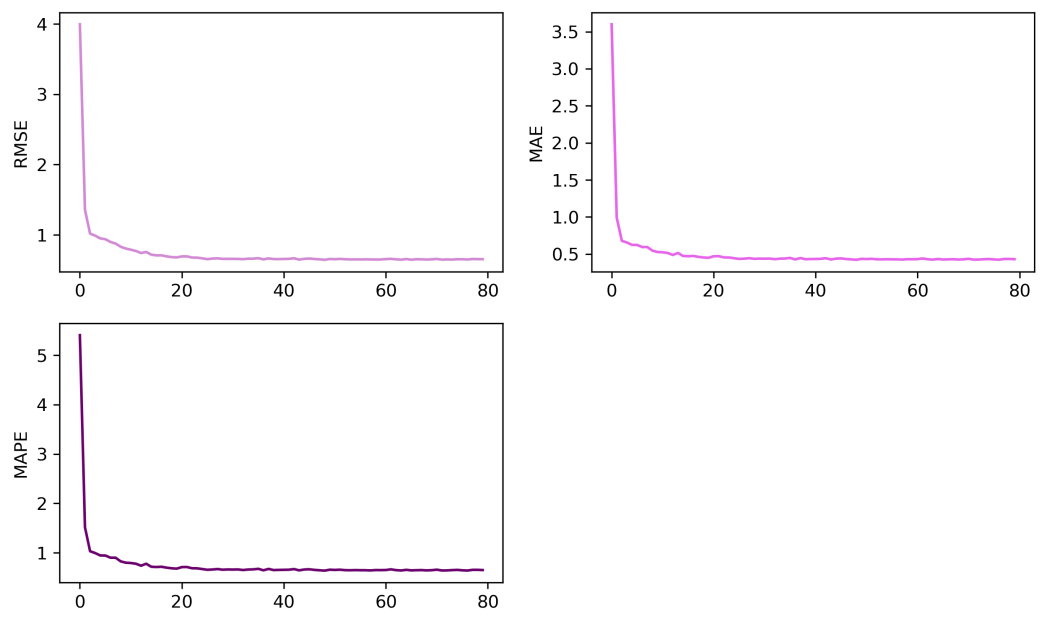


**分析：**单从训练函数和预测函数的损失来看，RNN的效果略好于LSTM，LSTM的效果略好于，是由于RNN采用了循环网络机制，提高计算效率，LSTM从原理的角度上优化性能略差于LSTM。GRU在71轮次数达到最佳训练效果，RNN在75轮次数达到最佳训练效果，LSTM在70轮达到了最佳训练效果；从训练时间入手，RNN的时间小于GRU的时间，GRU的时间又小于LSTM的时间，从三个封装测试函数入手而言，似乎RNN的效果和GRU的效果类似，由此在本次训练中，RNN在模型优化的效果方面略好于其他模型。

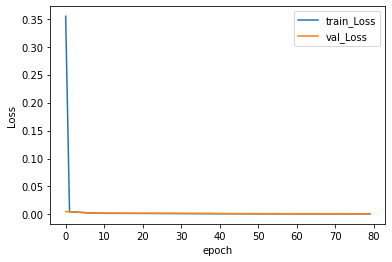
#### 5.4 LSTM的超参数对比

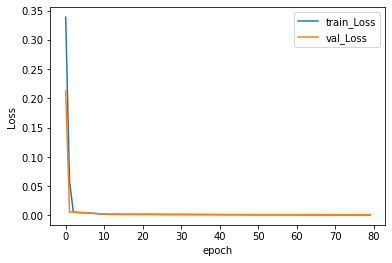


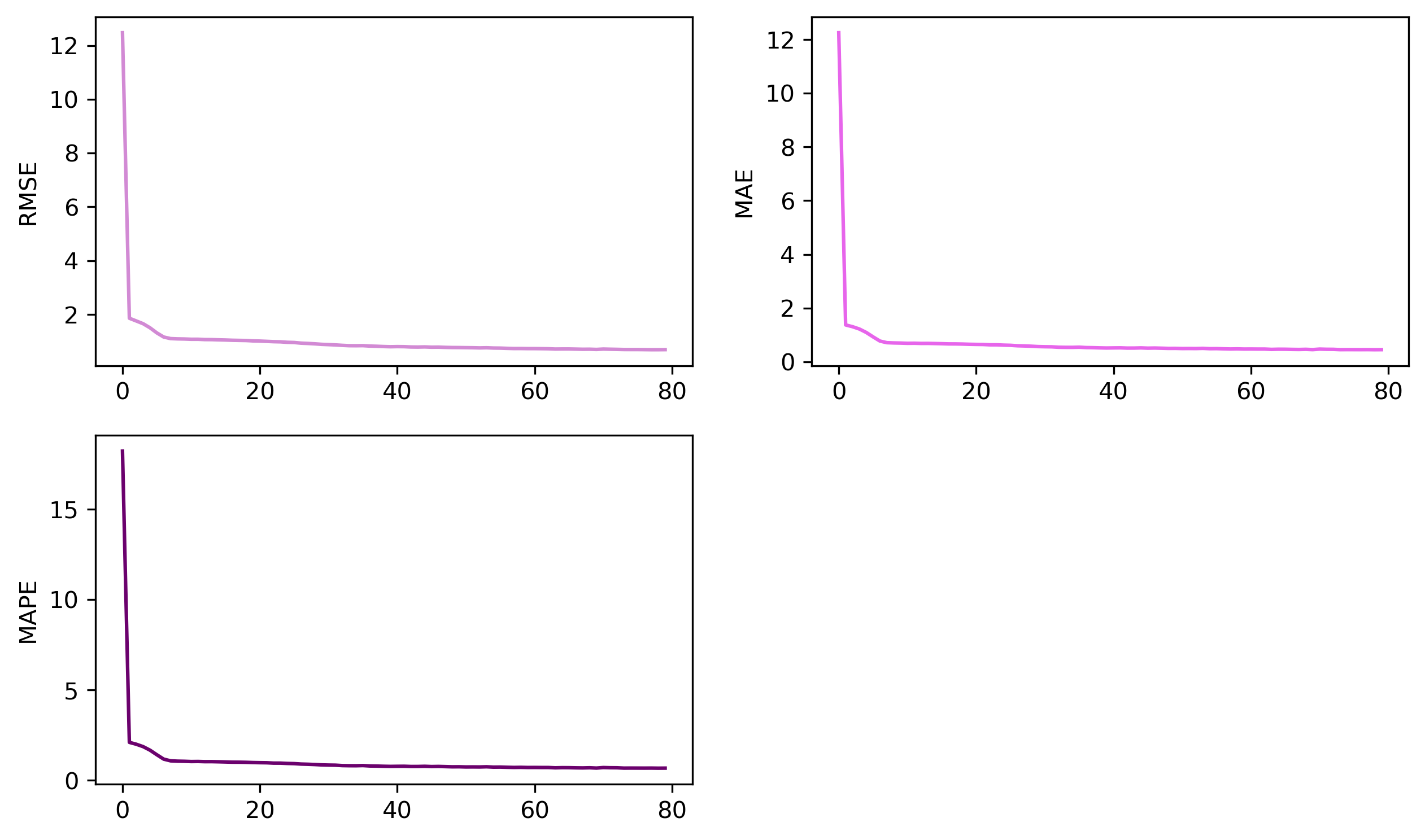


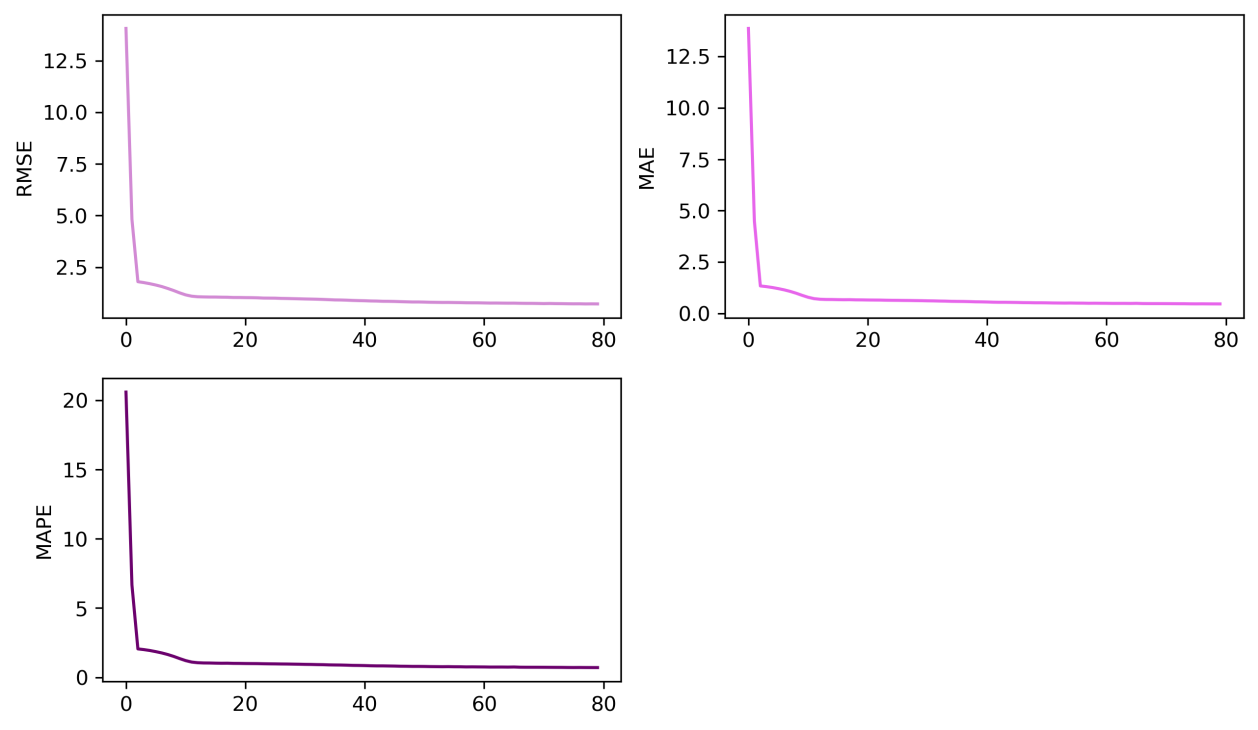


**分析**：如图是选用不同的学习率（lr=0.0001/0.001）的情况下两个不同训练函数的模型，其中当学习率较大时，预测的损失并没有学习率较小时平滑，在三个评估函数中，较大的学习率在评估函数上也存在一定的抖动现象，这主要原因是采用的学习率较大，导致对模型的评估不准确，从而使测试结果处于一个不断波动的情况，尽管如此，由于学习率还是相对较小的，因此总体而言差异不大。

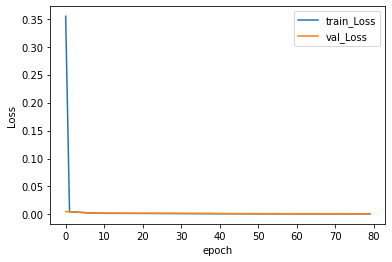


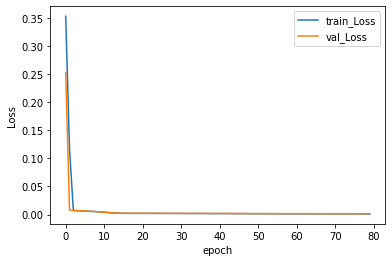


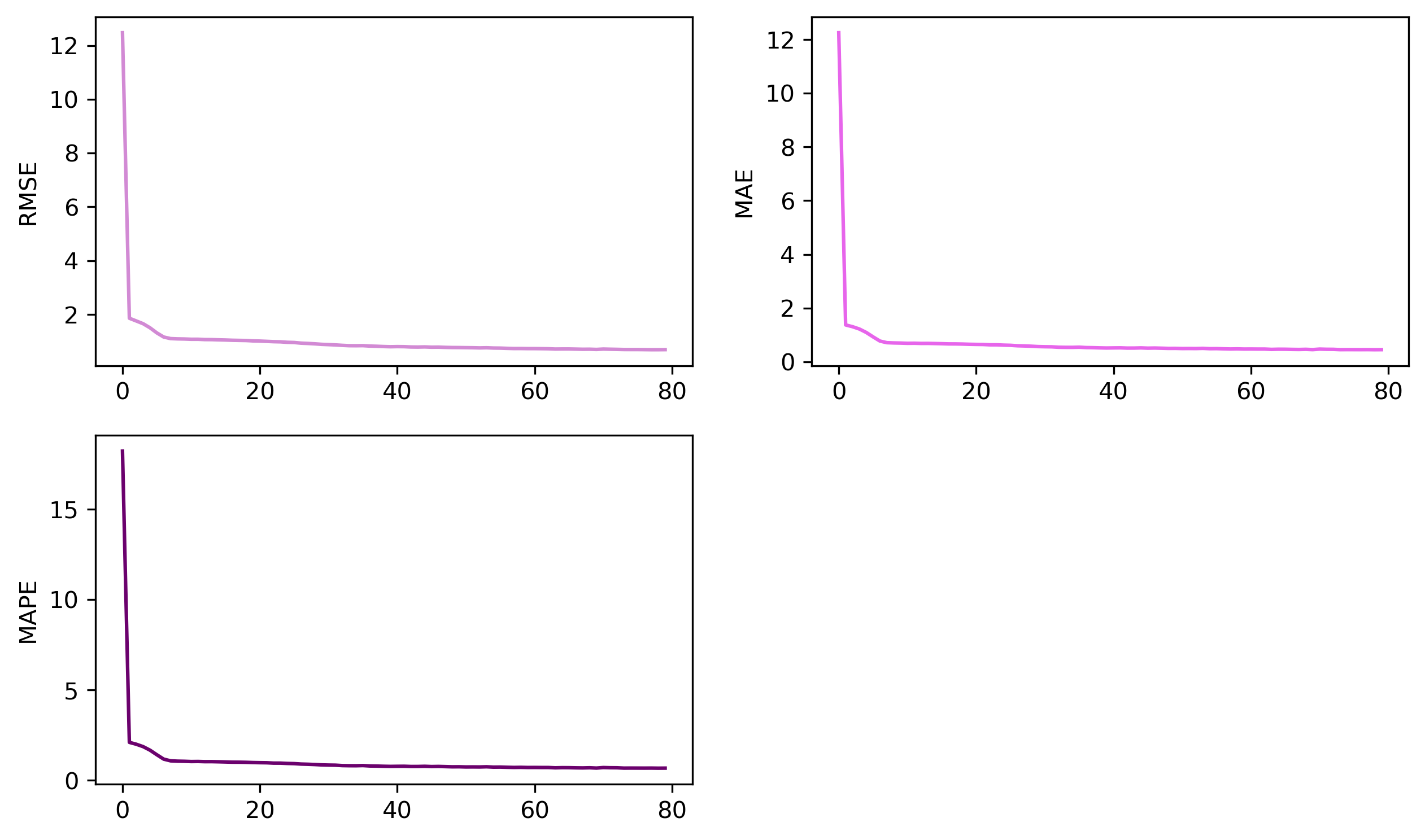


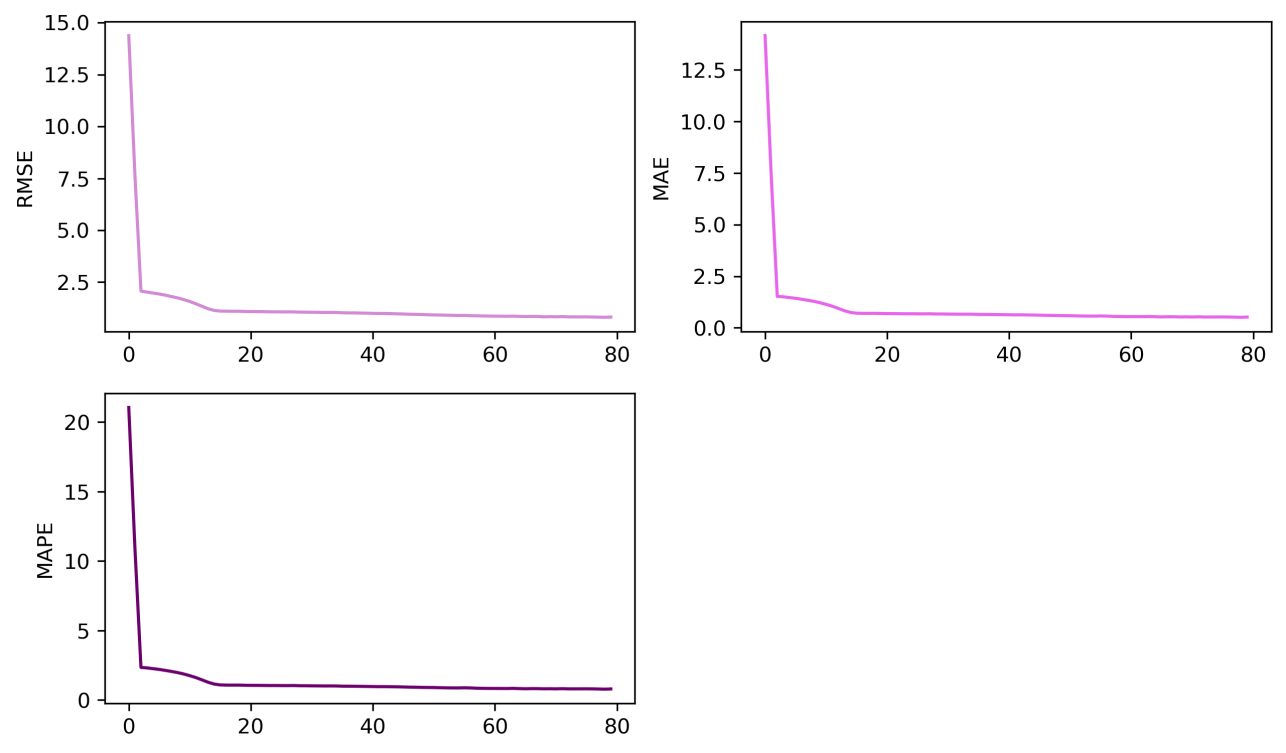


**分析**：如图是选用不同的隐藏层（hidden\_size=64/32）的情况下两个不同训练函数的模型，其中当隐藏较小时，训练的精度相对较低，但是隐藏层较少会减少数据处理的复杂度与时间，因此32个隐藏层只要180s左右的处理时间，比64个隐藏层减少了约50s，对应的三个评价函数的下降速度也是相对较慢的，在对最终数据影响不大的情况下还是有一定的参考价值。









**分析**：如图是选用不同的读取速度（batch\_size=64/128）的情况下两个不同训练函数的模型，虽然读取的速度快可以将采样时间减少50s以上，但是其最大的缺点在训练精度上的不足，其他参数下的训练损失多在0.001左右，而较大读取速度下的训练损失达到了0.0015左右，当应用到其他复杂情况时还是有所不足的。该训练模型同样在训练评价函数较大，下降速率慢，这个问题和减少隐藏层出现的问题一样。因此在条件允许的情况下，需要进一步减少读取样本的数量。所有的参数和结果将以ipynb的方式保存。

## 六、实验心得体会

这次实验是卷积神经网络实验，我们需要使用卷积神经网络，长短期记忆神经网络和门循环神经网络完成对数据的预测。相比于之前的其他实验，这次验考验的是对神经网络知识的综合应用，既包含了神经网络的定义，也需要掌握一定的面向对象编程语言的知识去解决实际问题。相对而言，这次的实验比之前的实验更有难度，相对参考的代码也并不是很多。

而我这次实验也完全参考了PPT，摆脱了对CSDN的依赖， 真正利用所学的知识去解决实际问题。当然在这个过程中也是遇到了很多问题，最开始碰到的问题就数组越界的问题，产生这部分问题的主要原因就是在模型定义的时候没有数据的返回类型与数据维度的问题，从而产生了数据的不匹配。当然我参考了相关内容也知道了，最大的问题就是出现在了我返回的时候。返回数组顺序颠倒，我修改了相关的代码，成功的解决了相关问题。第二个问题就是在模型预测方面，模型的预测输出数据量不匹配的问题。出现这个问题的原因并不是因为代码报错，而是因为在模型定义的时候，格式的对齐问题由于格式的不对，其从而导致了数据的赋值出现了错误，这就导致了输出的不匹配。后来我仔细检查了代码，重新一行一行的验证，逐行输出，从而找到了问题的所在。

最后就是采取调用接口实现相关功能的时候，由于我在数据定义的时候没有按照相关的格式进行输入，从而导致了有些地方出现了相关数据没有定义的报错。这也是Python语言最大的特点之一，这就是对语言的要求较高，只有认真细致才能慢工出细活。这次实验我真正是依靠自己完成的，并没有参考过多的CSDN代码，这是我能力提升的一个过程，当然在这个过程中我也发现了自己的不足之处，就是不够细致。在以后的学习，工作和生活中，我会更加谨慎细致，认真处理好每一个细节问题。

## 七、参考文献

[1] https://blog.csdn.net/qq\_37534947/article/details/108179408

[2] 动⼿学深度学习-PyTorch

## 八、附录

需要补充说明的内容，如无可略。

**实验报告填写要求**

1. 正文要求小四号宋体，行间距1.5倍；

2. 英文要求小四号Times New Roman；

3. 在实验内容、实验过程、实验结果三部分需要针对当次实验不同的实验内容分别填写（模版以实验一为例），实验设计中如有必要也可以分开填写；

4. 实验报告配图的每幅图应有编号和标题，编号和标题应位于图下方处，居中，中文用五号宋体；

5. 表格应为三线表，每个表格应有编号和标题，编号和标题应写在表格上方正中，距正文段前0.5倍行距。表格中量与单位之间用“/”分隔，编号与标题中的中文用五号宋体；

6. 图、表、公式、算式等，一律用阿拉伯数字分别依序连续编排序号。其标注形式应便于互相区别，可分别为：图1、表2、公式(5)等。