



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیووتر

شبکه های عصبی و یادگیری عمیق

Mini Project #2

پرهاشم زیلوچیان مقدم	نام و نام خانوادگی
810198304	شماره دانشجویی
1399/03/20	تاریخ ارسال گزارش

فهرست گزارش سوالات (لطفاً پس از تکمیل گزارش، این فهرست را به روز کنید.)

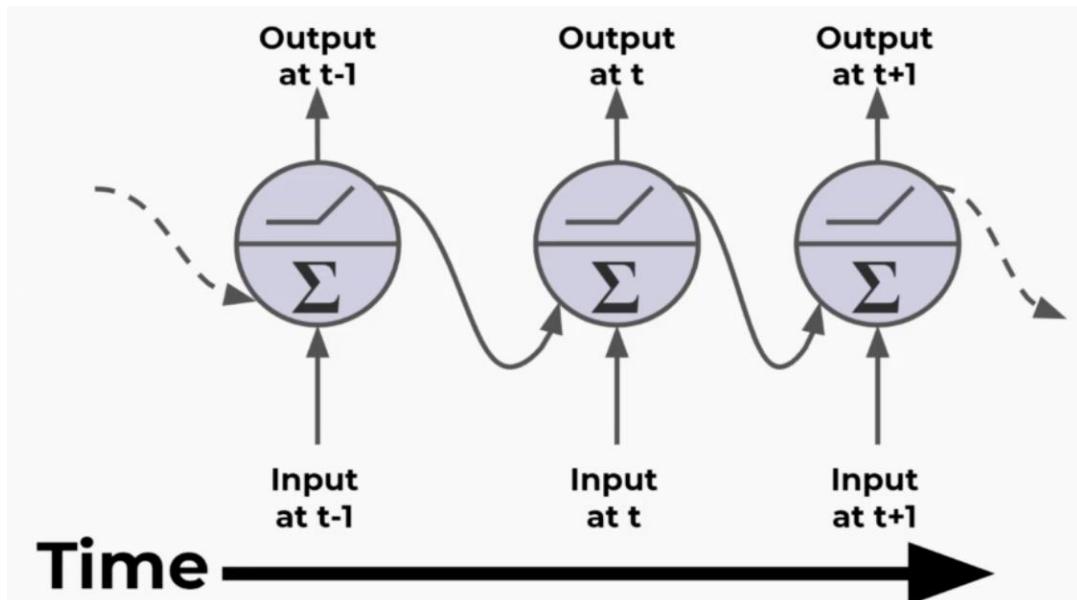
سوال ۱ – قسمت اول (طراحی شبکه‌های عصبی) 3

سوال ۲ – نقصان دادگان 220

سوال 1 - قسمت اول (طراحی شبکه‌های عصبی)

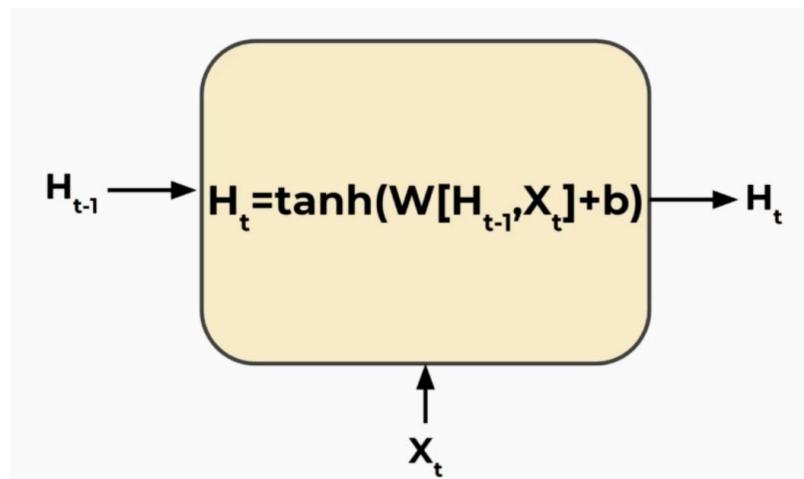
در این سوال از ما خواسته شده است که از شبکه‌های RNN برای پیش‌بینی آلودگی هوا در شهر Beijing چین استفاده کنیم. به این منظور از دیتابستی که به ما داده شده است استفاده می‌کنیم که شامل داده‌های مربوط به بازه زمانی 2010 تا 2015 این شهر می‌باشد.

این دیتابست داده شده به ما از نوع Time Series هست و ما می‌دانیم که شبکه‌های RNN در داده‌های Sequence‌ای عملکرد خوبی را دارند.



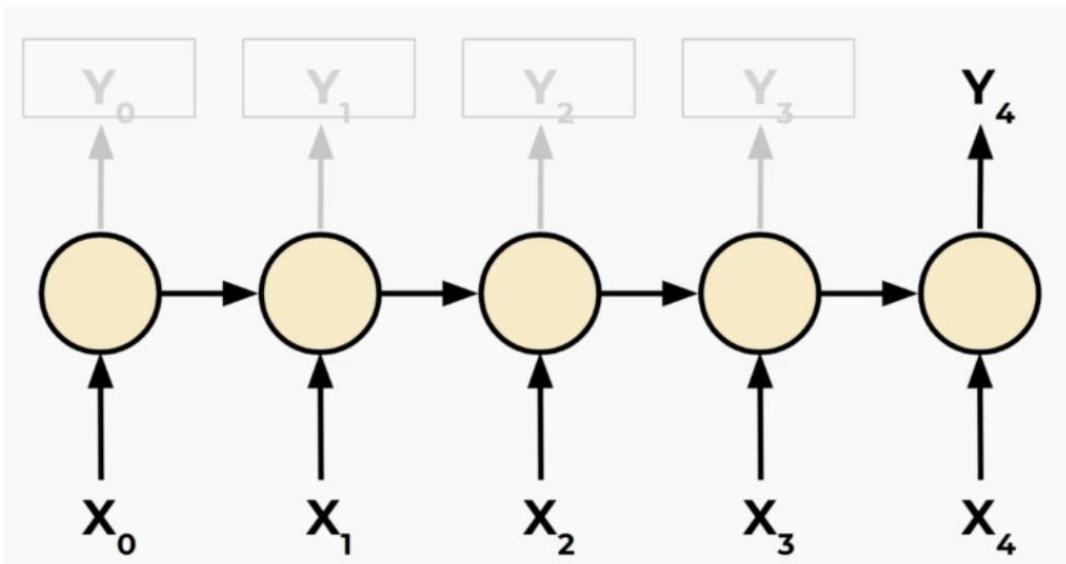
شکل 1: یک نمونه ساده از عملکرد RNN بر روی داده‌های Sequence‌ای

نرون‌هایی که در شکل بالا مشاهده می‌کنید در حقیقت نرون بازگشتی هستند.



شکل 2: نمونه یک نرون RNN

انواع مختلفی از RNN‌ها را داریم مانند One to Many و Many to One، Many to Many که در این سوال با توجه به این که از ما خواسته شده است که با توجه به ۱۱ دیتای قبلی مربوط به آلودگی هوا دیتای ۱۲ام را پیش‌بینی کنیم در نتیجه با حالت Many to One سر و کار داریم.



شکل ۳: حالت Many to One

پس از این موارد به سراغ بارگزاری دیتاست می‌رویم و این که بخواهیم روی آن عملیات‌های مربوطه را انجام دهیم.

```

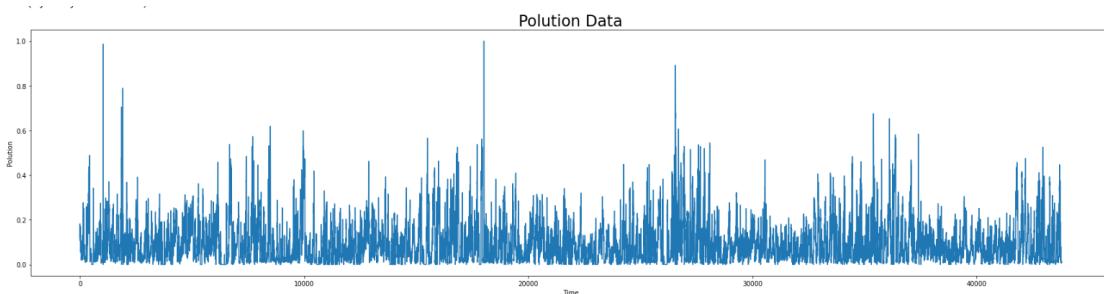
2 DATA_DIR = "Beijing-Pollution-DataSet/"
3
4 data = np.load(DATA_DIR + 'polution_dataSet.npy')

```

شکل ۴: دستورات بارگزاری دیتاست

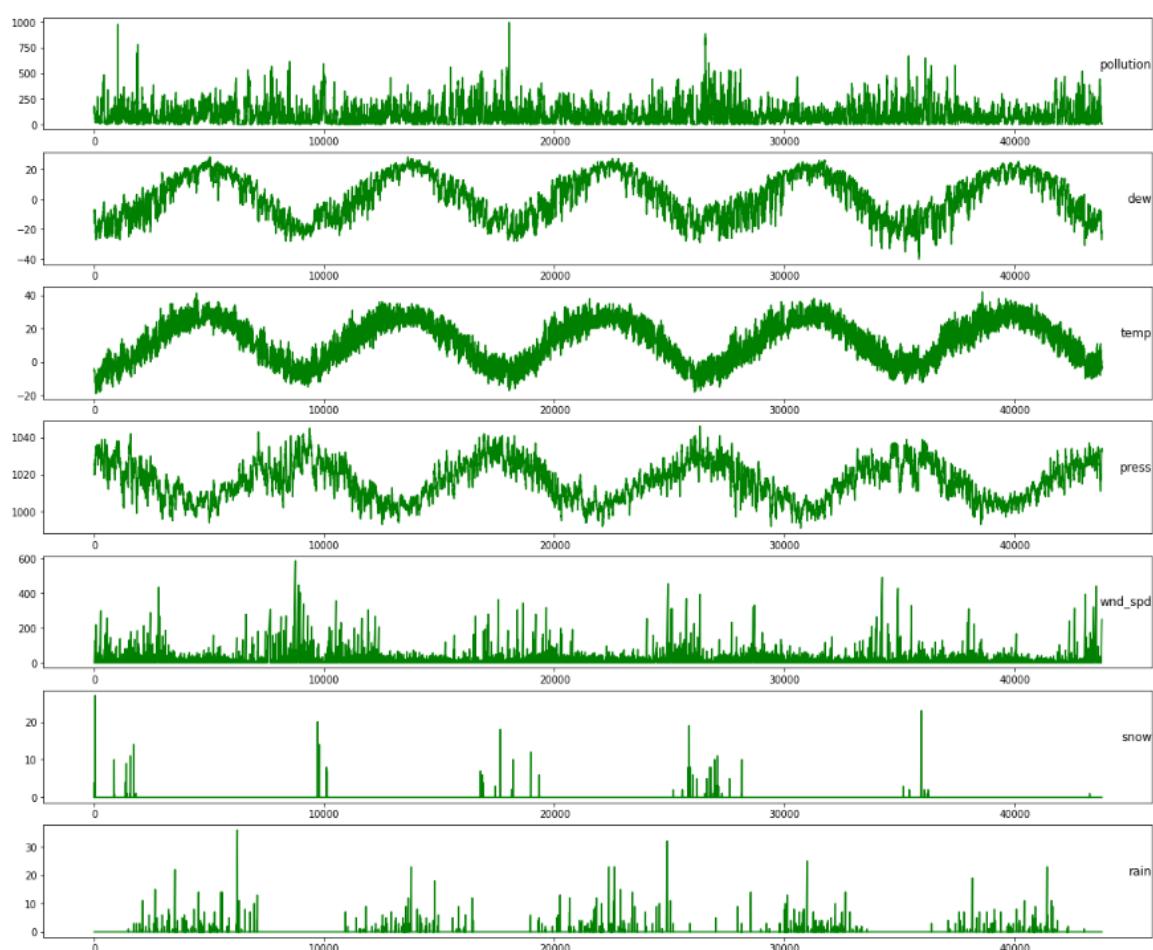
لازم به ذکر است که من دیتاستم را برای این که می‌خواستم که با گوگل Colab کار کنم در گیت بارگزاری کرده‌ام و از آنجا دیتاهایم را لود می‌کنم.

سپس من نمودار داده‌های مربوط به آلودگی که ستون اول داده‌ها است را رسم می‌کنم:



شکل ۵: نمودار مربوط به مقادیر آلودگی

همچنین در ادامه نموداری از تمامی داده‌های موجود را می‌توانید که مشاهده نمایید:



شکل ۶: نمودار مقادیر مختلف در کنار هم

همچنین لازم است به این موضوع اشاره کنم که داده‌های ما نرمال شده هستند و همگی بین ۰ و ۱ هستند و بنابراین نیازی به نرمال‌سازی مجدد آن‌ها نیست همچنین داده‌های Nan و نامربوط هم وجود ندارد پس می‌توانیم به سادگی به سراغ ادامه حل سوال برویم.

پس از این موضوع، شروع به ساخت شبکه و آموزش آن می‌کنیم.

❖ قسمت 1 و 2

در این دو قسمت از سوال از ما خواسته شده است که سه شبکه LSTM و GRU را طراحی کنم و سپس داده‌های آموزشی ام را به آن‌ها بدهم تا آموزش ببینند و سپس بر این اساس نمودارهای Loss مربوط به train و valid را رسم کنم. همچنین براساس مقادیر واقعی و پیش‌بینی شده نیز یک نمودار رسم کنم.

قسمت مهمی از این سوال تقسیم بندی داده‌ها است و به این منظور منتابع زیر را نوشتیم:

```
30 # split a multivariate sequence into samples
31 def split_sequences(sequences, n_steps=11, n_samples=12000, start_from=0):
32     X, y = list(), list()
33     for i in range(start_from, (start_from + n_samples)):
34         # find the end of this pattern
35         end_ix = i + n_steps
36         # check if we are beyond the dataset
37         # gather input and output parts of the pattern
38         seq_x = sequences[i:end_ix, :]
39         seq_y = sequences[end_ix, 0]
40         y.append(seq_y)
41         X.append(seq_x)
42
43     return array(X), array(y)
```

شکل 7: تابعی به منظور تقسیم بندی داده‌ها

کاری که تابع موجود در شکل بالا انجام می‌دهد این است که مقادیر را براساس پنجره زمانی که برای آن در نظر می‌گیریم که همان `n_steps` می‌باشد، می‌آید و داده‌ها را به این صورت جداسازی می‌کند که به تعداد پنجره‌زنی داده داریم که در اینجا 11 می‌باشد و سپس با توجه به این‌ها مقدار در کنار هر کدام نیز 8 ویژگی قرار دارد که برای سایر پارامترهای جوی هوا می‌باشد. پس از این اقدام به سراغ ساخت داده‌های valid و train می‌رویم.

```
1. n_timesteps = 11
2. dataset = data
3. train_X, train_y = split_sequences(dataset, n_timesteps, n_samples=15000, start_f
rom=0)
4. valid_X, valid_y = split_sequences(dataset, n_timesteps, n_samples=3000, start_fr
om=15000)
```

در قسمت بالا می‌توانید مشاهده کنید که از داده اول تا 15000 را به عنوان آموزش يا train و از 15000 تا 3000 تا بعدی آن یعنی 18000 را به عنوان valid در نظر گرفته‌ام.

در ادامه نیز من جدولی برای مقادیر داده‌های Train، Test و Valid را قرار می‌دهم.

جدول ۱: تعداد و بازه مربوط به مقادیر استفاده شده در سه مجموعه Valid، Train و Test

Set	Plenty	Period
Train	15000	1-15000
Valid	3000	15001-18000
Test	3000	18001-21000

پس از این مورد به سراغ ساعت مدل‌ها می‌روم که در ادامه کد مربوط به طراحی هر سه مدل RNN و GRU و LSTM را قرار می‌دهم.

ابتدا می‌توانید کد مربوط به RNN را مشاهده نمایید:

```
1. class RNN(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
3.                  , n_layers=1):
4.         super(RNN, self).__init__()
5.         self.n_features = n_features
6.         self.seq_len = seq_length
7.         self.n_output = n_output
8.
9.         self.n_hidden = n_hidden_layers # number of hidden states
10.        self.n_layers = n_layers # number of LSTM layers (stacked)
11.
12.        # define RNN with specified parameters
13.        # bath_first means that the first dim of the input and output will be the
batch_size
14.        self.rnn = nn.RNN(input_size=self.n_features,
15.                           hidden_size=self.n_hidden,
16.                           num_layers=self.n_layers,
17.                           batch_first=True)
18.
19.        # last, fully connected layer
20.        self.l_linear = torch.nn.Linear(self.n_hidden*self.seq_len, self.n_output
)
21.
22.    def forward(self, x, hidden):
23.        # hidden_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).req
uires_grad_()
24.        # cell_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requi
res_grad_()
25.        batch_size = x.size(0)
```

```

26.         rnn_out, hidden = self.rnn(x, hidden)
27.         # print(rnn_out.shape)
28.         rnn_out = rnn_out.contiguous().view(batch_size, -1)
29.
30.         # lstm_out(with batch_first = True) is
31.         # (batch_size,seq_len,num_directions * hidden_size)
32.         # for following linear layer we want to keep batch_size dimension and merge rest
33.         # .contiguous() -> solves tensor compatibility error
34.         # x = lstm_out.contiguous().view(batch_size, -1)
35.         out = self.l_linear(rnn_out)
36.
37.     return out, hidden

```

سپس در ادامه می‌توانید که کد مربوط به LSTM را مشاهده کنید:

```

1. class LSTM(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
3.                  , n_layers=1):
3.         super(MV_LSTM, self).__init__()
4.         self.n_features = n_features
5.         self.seq_len = seq_length
6.
7.         self.n_hidden = n_hidden_layers # number of hidden states
8.         self.n_layers = n_layers # number of LSTM layers (stacked)
9.         self.n_output = n_output
10.
11.        self.l_lstm = torch.nn.LSTM(input_size = n_features,
12.                                     hidden_size = self.n_hidden,
13.                                     num_layers = self.n_layers,
14.                                     batch_first = True)
15.        # according to pytorch docs LSTM output is
16.        # (batch_size, seq_len, num_directions * hidden_size)
17.        # when considering batch_first = True
18.        self.l_linear = torch.nn.Linear(self.n_hidden * self.seq_len, self.n_output)
19.
20.
21.    def forward(self, x):
22.        hidden_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requires_grad_()
23.        cell_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requires_grad_()
24.        self.hidden = (hidden_state.detach(), cell_state.detach())
25.
26.        batch_size, seq_len, _ = x.size()
27.
28.        lstm_out, self.hidden = self.l_lstm(x, self.hidden)
29.
30.        # lstm_out(with batch_first = True) is
31.        # (batch_size,seq_len,num_directions * hidden_size)
32.        # for following linear layer we want to keep batch_size dimension and merge rest
33.        # .contiguous() -> solves tensor compatibility error
34.        x = lstm_out.contiguous().view(batch_size, -1)
35.        # print("X shape :=> ", x.shape)
36.        # out = self.l_linear(lstm_out[:, -1, :])
37.        # print("Out Shape :=> ", lstm_out[:, -1, :].shape)
38.        out = self.l_linear(x)
39.
40.    return out

```

سپس در ادامه نیز می‌توانید که کد مربوط به GRU را مشاهده کنید:

```
1. class GRU(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
33, n_layers=1):
3.         super(GRU, self).__init__()
4.         self.n_features = n_features
5.         self.seq_len = seq_length
6.         self.n_output = n_output
7.
8.         self.n_hidden = n_hidden_layers # number of hidden states
9.         self.n_layers = n_layers # number of LSTM layers (stacked)
10.
11.        # define RNN with specified parameters
12.        # bath_first means that the first dim of the input and output will be the
batch_size
13.        self.rnn = nn.GRU(input_size=self.n_features,
14.                           hidden_size=self.n_hidden,
15.                           num_layers=self.n_layers,
16.                           batch_first=True)
17.
18.        # last, fully connected layer
19.        self.l_linear = torch.nn.Linear(self.n_hidden*self.seq_len, self.n_output
)
20.
21.
22.    def forward(self, x, hidden):
23.        # hidden_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).req
uires_grad_()
24.        # cell_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requi
res_grad_()
25.        batch_size = x.size(0)
26.
27.        rnn_out, hidden = self.rnn(x, hidden)
28.        rnn_out = rnn_out.contiguous().view(batch_size, -1)
29.
30.        # lstm_out(with batch_first = True) is
31.        # (batch_size,seq_len,num_directions * hidden_size)
32.        # for following linear layer we want to keep batch_size dimension and mer
ge rest
33.        # .contiguous() -> solves tensor compatibility error
34.        out = self.l_linear(rnn_out)
35.        return out, hidden
```

پس از این موارد اکنون به سراغ آموزش تک‌تک این مدل‌ها می‌رویم.

برای شروع و آزمون اول من به عنوان Optimizer از Adam استفاده کردم و همچنین برای سنجش میزان Loss نیز از MSE استفاده کردم:

همچنین برای یکسان بودن تمامی مدل‌ها من از Hyper Parameter های زیر استفاده می‌کنم:

جدول 2: مقادیر Hyper Parameter های استفاده شده در تمامی مدل‌ها

Hyper Parameter	Value
Input_size (num_features)	8
n_outputs	1
Num_layers	1
Hidden_size	233
Learning Rate	0.0003
Batch Size	200

همان‌طور که در جدول بالا می‌توانید مشاهده کنید، مقادیر تمامی Hyper Parameter ها برای سه مدل LSTM، GRU و RNN را یکسان در نظر می‌گیرم.

ابتدا از RNN شروع می‌کنیم:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 8: تنظیم و تعریف مدل RNN

سپس با استفاده از کدهای زیر اقدام به آموزش شبکه می‌کنیم:

```

1. import time
2. start_time = time.time()
3. hidden = None
4. hidden_test = None
5. epochs = 100
6. model.train()
7. batch_size = 200
8. running_loss_history = []
9. val_running_loss_history = []
10. for epoch in range(epochs):
11.     running_loss = 0.0
12.     val_running_loss = 0.0
13.     model.train()

```

```

14.     for b in range(0, len(train_X), batch_size):
15.         inpt = train_X[b:b+batch_size, :, :]
16.         target = train_y[b:b+batch_size]
17.
18.         # print("Input Shape :=> ", inpt.shape)
19.
20.         x_batch = torch.tensor(inpt, dtype=torch.float32)
21.         y_batch = torch.tensor(target, dtype=torch.float32)
22.
23.         output, hidden = model(x_batch, hidden)
24.
25.         hidden = hidden.data
26.         loss = criterion(output.view(-1), y_batch)
27.
28.         running_loss += loss.item()
29.
30.         loss.backward()
31.         optimizer.step()
32.         optimizer.zero_grad()
33.
34.     else:
35.         with torch.no_grad(): # it will temporarily set all the required grad flags to be false
36.             model.eval()
37.             for b in range(0, len(valid_X), batch_size):
38.                 inpt = valid_X[b:b+batch_size, :, :]
39.                 target = valid_y[b:b+batch_size]
40.
41.                 x_batch_test = torch.tensor(inpt, dtype=torch.float32)
42.                 y_batch_test = torch.tensor(target, dtype=torch.float32)
43.
44.                 # model.init_hidden(x_batch_test.size(0))
45.
46.                 output_test, hidden_test = model(x_batch_test, hidden_test)
47.
48.                 hidden_test = hidden_test.data
49.                 loss_valid = criterion(output_test.view(-1), y_batch_test)
50.
51.                 val_running_loss += loss_valid.item()
52.
53.             val_epoch_loss = val_running_loss / len(valid_X)
54.             val_running_loss_history.append(val_epoch_loss)
55.             epoch_loss = running_loss / len(train_X)
56.             running_loss_history.append(epoch_loss)
57.             print('step : ', epoch, ' Train loss : ', epoch_loss, ', Valid Loss : => ', val_epoch_loss)
58.             print("****->>>-----<<<-****")
59.
60.     total_time = time.time() - start_time
61.     print("=====")
62.     print("*****")
63.     print("The total Training Time is Equal with ==> : ", total_time)
64.     print("*****")
65.     print("=====")

```

سپس مقادیر Loss به دست آمده برای epoch 5 اول عبارت اند از:

```

step : 0 Train loss : 3.0320474342443048e-05 , Valid Loss : => 7.660914119333029e-05
***->>-----<<<-***  

step : 1 Train loss : 2.3332416727983704e-05 , Valid Loss : => 0.0001033981287231048
***->>-----<<<-***  

step : 2 Train loss : 2.604382321393738e-05 , Valid Loss : => 0.00010417895733068386
***->>-----<<<-***  

step : 3 Train loss : 2.3963819482984642e-05 , Valid Loss : => 5.716708418913186e-05
***->>-----<<<-***  

step : 4 Train loss : 1.6578543543194733e-05 , Valid Loss : => 5.8008673523242276e-05
***->>-----<<<-***  

step : 5 Train loss : 1.621348800060029e-05 , Valid Loss : => 4.479623772203922e-05
***->>-----<<<-***  


```

شکل 9: مقادیر 5 ایپاک اول

سپس مقادیر 5 ایپاک آخر را نیز می‌توانید در ادامه مشاهده نمایید:

```

step : 93 Train loss : 4.110078479667815e-06 , Valid Loss : => 5.199725555333619e-06
***->>-----<<<-***  

step : 94 Train loss : 4.174502585859349e-06 , Valid Loss : => 5.415236965442697e-06
***->>-----<<<-***  

step : 95 Train loss : 4.27423189006125e-06 , Valid Loss : => 5.561507811459402e-06
***->>-----<<<-***  

step : 96 Train loss : 4.260481764019156e-06 , Valid Loss : => 5.31798112206161e-06
***->>-----<<<-***  

step : 97 Train loss : 4.1969262432151786e-06 , Valid Loss : => 5.06400378071703e-06
***->>-----<<<-***  

step : 98 Train loss : 4.116539996660625e-06 , Valid Loss : => 5.2543139997093625e-06
***->>-----<<<-***  

step : 99 Train loss : 4.1935943280501915e-06 , Valid Loss : => 5.472144422431787e-06
***->>-----<<<-***  


```

شکل 10: مقادیر 5 ایپاک آخر آموزش

سپس در ادامه می‌توانید مقدار زمان آموزش را به ثانیه مشاهده نمایید:

```

=====
*****  

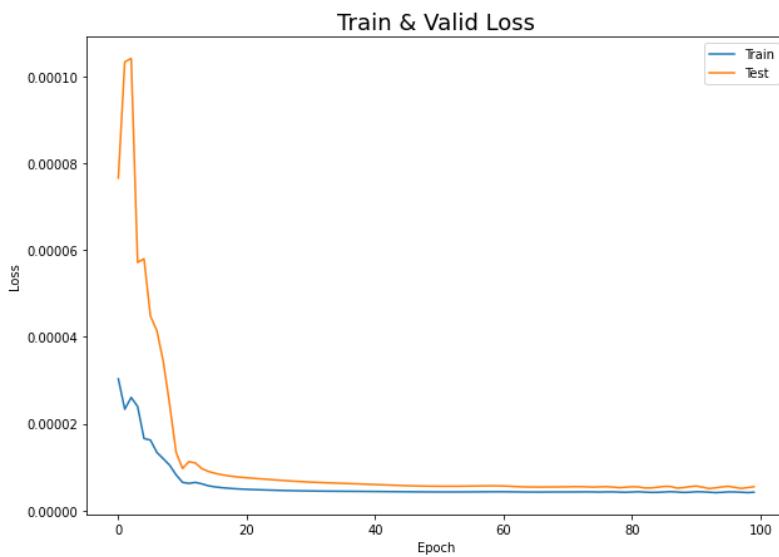
The total Training Time is Equal with ==> : 163.33652544021606 Sec.  

*****  

=====
```

شکل 11: مقدار زمان آموزش با RNN با .OPTIMIZER ADAM

سپس در ادامه می‌توانید نمودار مقادیر Loss را مشاهده کنید:



شکل 12: نمودار مربوط به Loss در دو مجموعه Train و Valid برای مدل RNN همراه با Optimizer Adam و MSE Loss

سپس شروع به رسم نمودار مربوط به مقادیر واقعی و پیش‌بینی می‌کنیم. ابتدا این نمودار را به ازای تمامی مقادیر Test رسم می‌کنیم که یعنی از 18000 تا 21000 که می‌توانید آن را در ادامه مشاهده کنید:

```

1 test_x, test_y = split_sequences(dataset, n_timesteps, n_samples=3000, start_from=18000)
2 model.eval()
3 test_x = torch.tensor(test_x, dtype=torch.float32)
4 res, hid = model(test_x, None)

```

شکل 13: کد مربوط به محاسبه مقادیر پیش‌بینی داده‌های Test

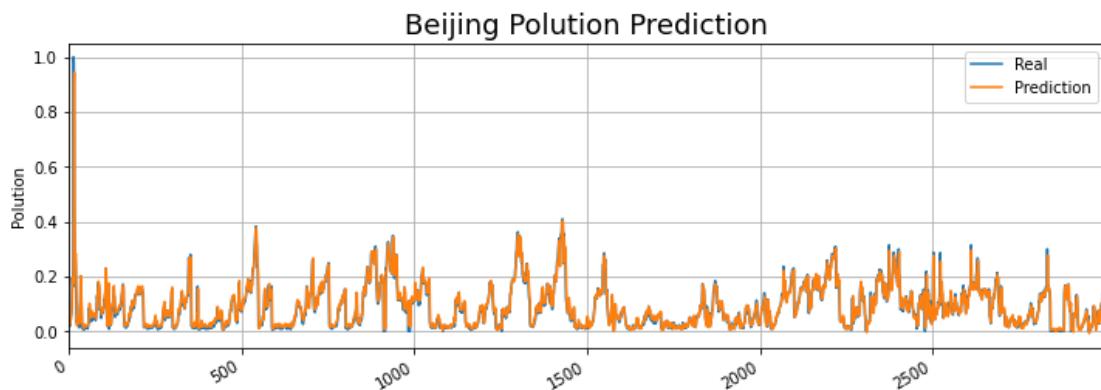
سپس در ادامه می‌توانید نمودار مربوطه را مشاهده کنید:

```

1 fig = plt.figure(figsize=(12, 4))
2 plt.title("Beijing Pollution Prediction", fontsize=18)
3 plt.ylabel('Pollution')
4 plt.grid(True)
5 plt.autoscale(axis='x', tight=True)
6 fig.autofmt_xdate()
7 plt.plot(test_y, label="Real")
8 plt.plot(res.detach().numpy(), label="Prediction")
9 plt.legend()
10 plt.show()

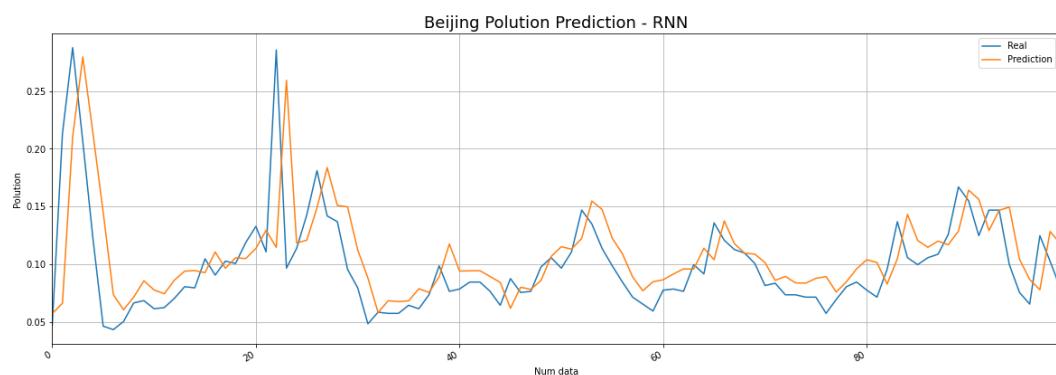
```

شکل 14: کد مربوط به رسم نمودار مربوطه



شکل 15: نمودار مقدار واقعی و پیش‌بینی برای حالت استفاده از مدل RNN و ADAM و MSE

برای این که بتوانید بهتر دقت آن را مشاهده کنید، آن را برای 100 داده رسم می‌کنم:



شکل 16: مقدار داده‌های واقعی و پیش‌بینی به ازای 100 داده برای حالت مدل RNN و ADAM و MSE

در ادامه نیز می‌توانید میزان Loss را برای داده‌های تست مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 4.706020466983319e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 17: میزان Loss به دست امده برای مدل RNN در حالت MSE و Adam

این نمودارهایی که در بالا رسم کردم مربوط به مقادیر ساعت‌های پیوسته بود و یعنی تمامی ساعت‌ها را به ازای 11 ساعت قبلی پیش‌بینی کردہ‌ام.

اما در ادامه نیز من نمودار و یک جدول از پیش‌بینی به ازای مقادیر فقط در ساعت‌های 12 و 24 را

قرار می‌دهم:

به این منظور من یک تابع جدید به نام split_sequence12 را نوشتم که این کار را برای من انجام می‌دهد و این تابع را می‌توانید در ادامه مشاهده کنید:

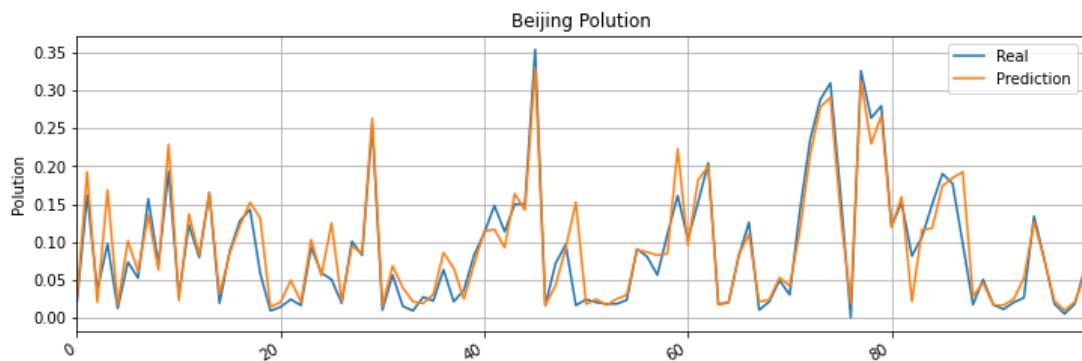
```

1 # split a multivariate sequence into samples
2 def split_sequences12(sequences, n_steps, n_samples=12000, start_from=0):
3     X, y = list(), list()
4     j = 0
5     for i in range(start_from, (start_from + n_samples)):
6         # find the end of this pattern
7         end_ix = j*12 + n_steps + start_from
8         # check if we are beyond the dataset
9         # gather input and output parts of the pattern
10        j = j + 1
11        seq_x = sequences[end_ix-11:end_ix, :]
12        seq_y = sequences[end_ix, 0]
13        y.append(seq_y)
14        X.append(seq_x)
15    return array(X), array(y)

```

شکل 18: تابع به منظور در نظر گرفتن stride=12 برای ساخت دیتاست

سپس نمودار به ازای 100 مجموعه 12 تایی از داده‌های تست به صورت زیر می‌شود:



شکل 19: نمودار مربوط به 100 مجموعه 12 تایی از داده‌های تست

همچنین من در ادامه تعدادی از داده‌ها را به عنوان نمونه نشان می‌دهم (در کنار مقدار واقعی مقدار پیش‌بینی شده نیز قرار داده شده است):

» نسخه کامل این داده‌ها را می‌توانید که در فایل predict_every12Hour_RNN_ADAM_MSE.csv مشاهده کنید که در کنار فایل‌های آپلود شده قرار دارد.

	Real Values	Predicted Values
0	0.021127	0.042602
1	0.161972	0.201825
2	0.036217	0.036238
3	0.097586	0.178540
4	0.012072	0.032184
5	0.073441	0.114631
6	0.052314	0.070277
7	0.156942	0.146373
8	0.070423	0.077607
9	0.193159	0.239618
10	0.029175	0.040697
11	0.122736	0.150965
12	0.079477	0.096206
13	0.164990	0.174189
14	0.019115	0.044323
15	0.087525	0.101851
16	0.127767	0.135327
17	0.142857	0.163645
18	0.058350	0.137024
19	0.009054	0.027926
20	0.014085	0.035381

شکل 20: داده‌های Test پیش‌بینی شده و مقدار واقعی آن‌ها در حالت مدل RNN و MSE و ADAM

در ادامه سپس به بررسی حالت LSTM می‌پردازیم:

► بررسی LSTM در حالت MSE و ADAM

در ادامه می‌توانید که کد مربوط به مدل LSTM را مشاهده کنید:

```
1. class LSTM(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
33, n_layers=1):
```

```

3.     super(LSTM, self).__init__()
4.     self.n_features = n_features
5.     self.seq_len = seq_length
6.
7.     self.n_hidden = n_hidden_layers # number of hidden states
8.     self.n_layers = n_layers # number of LSTM layers (stacked)
9.     self.n_output = n_output
10.
11.    self.l_lstm = torch.nn.LSTM(input_size = n_features,
12.                                hidden_size = self.n_hidden,
13.                                num_layers = self.n_layers,
14.                                batch_first = True)
15.    # according to pytorch docs LSTM output is
16.    # (batch_size, seq_len, num_directions * hidden_size)
17.    # when considering batch_first = True
18.    self.l_linear = torch.nn.Linear(self.n_hidden * self.seq_len, self.n_output)
19.
20.
21.    def forward(self, x):
22.        hidden_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requires_grad_()
23.        cell_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requires_grad_()
24.        self.hidden = (hidden_state.detach(), cell_state.detach())
25.
26.        batch_size, seq_len, _ = x.size()
27.
28.        lstm_out, self.hidden = self.l_lstm(x, self.hidden)
29.
30.        # lstm_out(with batch_first = True) is
31.        # (batch_size,seq_len,num_directions * hidden_size)
32.        # for following linear layer we want to keep batch_size dimension and merge rest
33.        # .contiguous() -> solves tensor compatibility error
34.        x = lstm_out.contiguous().view(batch_size, -1)
35.        # print("X shape :=> ", x.shape)
36.        # out = self.l_linear(lstm_out[:, -1, :])
37.        # print("Out Shape :=> ", lstm_out[:, -1, :].shape)
38.        out = self.l_linear(x)
39.
        return out

```

سپس به تعریف تابع مربوط به Loss و Optimizer می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 21: تعریف مدل و Loss و تابع optimizer برای LSTM

پس از این موضوع بر اساس کد زیر اقدام به آموزش این شبکه می‌کنیم:

```

1. import time
2. start_time = time.time()
3.
4. # train_X, train_y
5. epochs = 100
6. model.train()
7. batch_size = 200

```

```

8. running_loss_history = []
9. val_running_loss_history = []
10. for epoch in range(epochs):
11.     running_loss = 0.0
12.     val_running_loss = 0.0
13.     model.train()
14.     for b in range(0, len(train_X), batch_size):
15.         inpt = train_X[b:b+batch_size, :, :]
16.         target = train_y[b:b+batch_size]
17.
18.         # print("Input Shape :=> ", inpt.shape)
19.
20.         x_batch = torch.tensor(inpt, dtype=torch.float32)
21.         y_batch = torch.tensor(target, dtype=torch.float32)
22.
23.         output = model(x_batch)
24.         loss = criterion(output.view(-1), y_batch)
25.
26.         running_loss += loss.item()
27.
28.         loss.backward()
29.         optimizer.step()
30.         optimizer.zero_grad()
31.
32.     else:
33.
34.         with torch.no_grad(): # it will temporarily set all the required grad flags to be false
35.             model.eval()
36.             for b in range(0, len(test_X), batch_size):
37.                 inpt = test_X[b:b+batch_size, :, :]
38.                 target = test_y[b:b+batch_size]
39.
40.                 x_batch_test = torch.tensor(inpt, dtype=torch.float32)
41.                 y_batch_test = torch.tensor(target, dtype=torch.float32)
42.
43.                 # model.init_hidden(x_batch_test.size(0))
44.
45.                 output_test = model(x_batch_test)
46.                 loss_test = criterion(output_test.view(-1), y_batch_test)
47.
48.                 val_running_loss += loss_test.item()
49.
50.     val_epoch_loss = val_running_loss / len(test_X)
51.     val_running_loss_history.append(val_epoch_loss)
52.     epoch_loss = running_loss / len(train_X)
53.     running_loss_history.append(epoch_loss)
54.     print('step : ', epoch, ' Train loss : ', epoch_loss, ', Valid Loss : => ', val_epoch_loss)
55.     print("****->>>-----<<<-****")
56.
57. total_time = time.time() - start_time
58. print("=====")
59. print("*****")
60. print("The total Training Time is Equal with ==> : {0} Sec.".format(total_time))

61. print("*****")
62. print("=====")

```

پس از این در ادامه می‌توانید که مقدار مربوط به Loss 5 اپاک اول را مشاهده کنید:

```

step : 0 Train loss : 3.401486576379587e-05 , Valid Loss : => 8.084209232280651e-05
***->>>-----<<<-***  

step : 1 Train loss : 2.9394303588196635e-05 , Valid Loss : => 0.00010162711919595798
***->>>-----<<<-***  

step : 2 Train loss : 3.4934918714376785e-05 , Valid Loss : => 8.676674806823334e-05
***->>>-----<<<-***  

step : 3 Train loss : 2.6448412123136224e-05 , Valid Loss : => 4.439612322797378e-05
***->>>-----<<<-***  

step : 4 Train loss : 1.6675084945745766e-05 , Valid Loss : => 3.52428094483912e-05
***->>>-----<<<-***  

step : 5 Train loss : 1.3896585422723243e-05 , Valid Loss : => 3.05624451333036e-05
***->>>-----<<<-***
```

شکل 22: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل LSTM

سپس در ادامه می‌توانید که مقادیر Loss را برای 5 ایپاک آخر مشاهده کنید:

```

***->>>-----<<<-***  

step : 93 Train loss : 3.961719358630944e-06 , Valid Loss : => 5.617984754887099e-06
***->>>-----<<<-***  

step : 94 Train loss : 3.957874840610505e-06 , Valid Loss : => 5.595787651448821e-06
***->>>-----<<<-***  

step : 95 Train loss : 3.9424581361042025e-06 , Valid Loss : => 5.556256869264568e-06
***->>>-----<<<-***  

step : 96 Train loss : 3.921036586689297e-06 , Valid Loss : => 5.549601715756581e-06
***->>>-----<<<-***  

step : 97 Train loss : 3.90870171734908e-06 , Valid Loss : => 5.586298861696074e-06
***->>>-----<<<-***  

step : 98 Train loss : 3.912303473528785e-06 , Valid Loss : => 5.6333737447857855e-06
***->>>-----<<<-***  

step : 99 Train loss : 3.923137904106018e-06 , Valid Loss : => 5.671373248333112e-06
***->>>-----<<<-***
```

شکل 23: مقادیر Loss داده‌های آموزشی و valid برای 5 ایپاک آخر مدل LSTM

سپس در ادامه می‌توانید مدت زمان آموزش این شبکه را برای 100 ایپاک را مشاهده نمایید:

```

=====
*****  

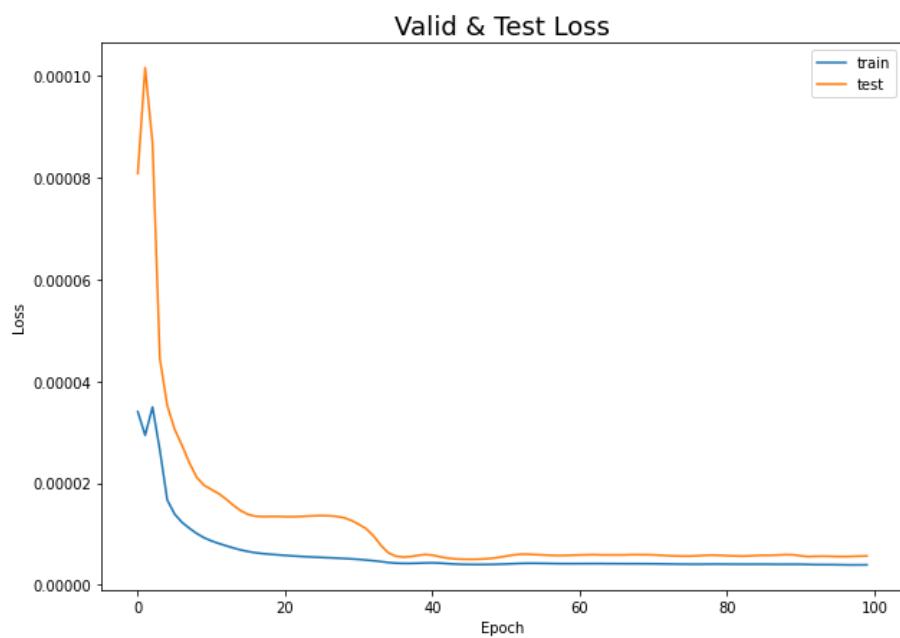
The total Training Time is Equal with ==> : 672.2394609451294 Sec.  

*****  

=====
```

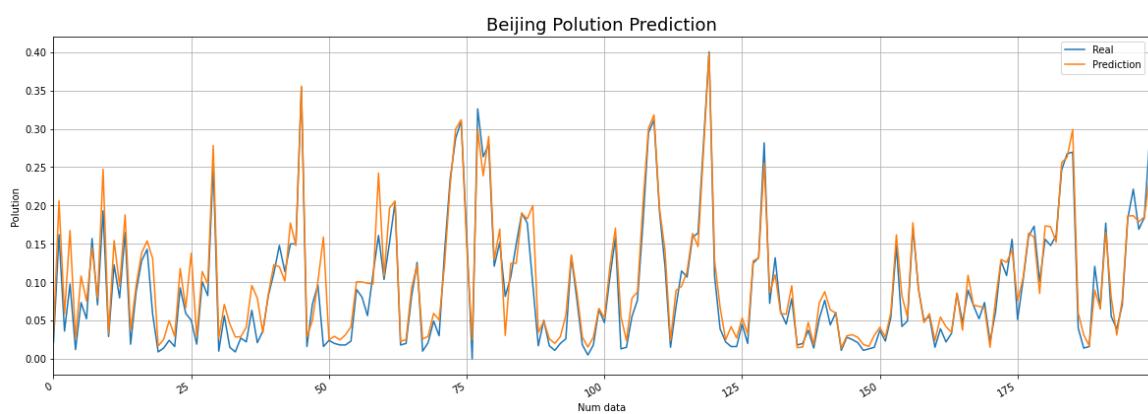
شکل 24: مقدار زمان آموزش برای مدل LSTM در حالت MSE و ADAM

سپس در ادامه می‌توانید نمودار مربوط به Loss در دو حالت train و test را مشاهده کنید:



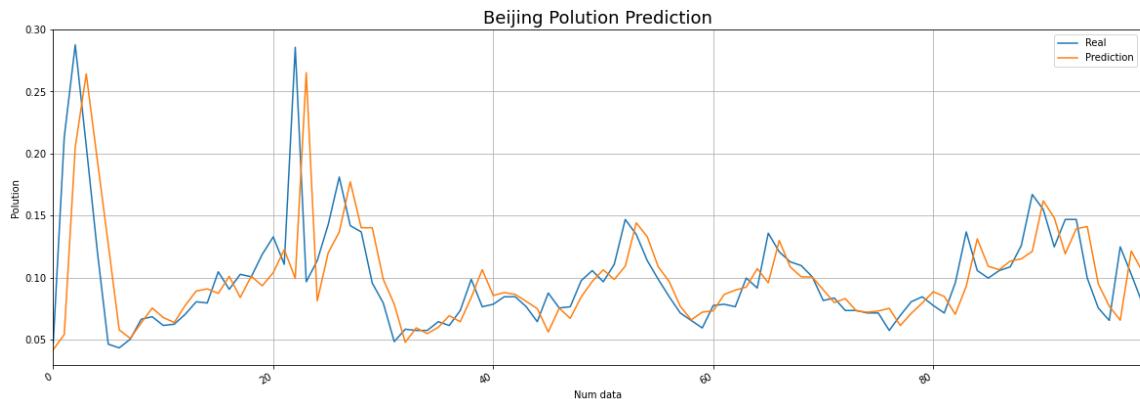
شکل 25: نمودار مربوط به Loss برای مدل LSTM در حالت MSE و ADAM

سپس در ادامه می‌توانید نمودار مربوط به میزان آلودگی واقعی و مقدار پیش‌بینی را مشاهده نمایید (برای کل مجموعه داده‌های تست):



شکل 26: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM برای حالت MSE و ADAM

سپس در ادامه برای مقایسه بهتر این نمودار را برای تعداد 100 عدد از داده‌ها رسم کردیم:



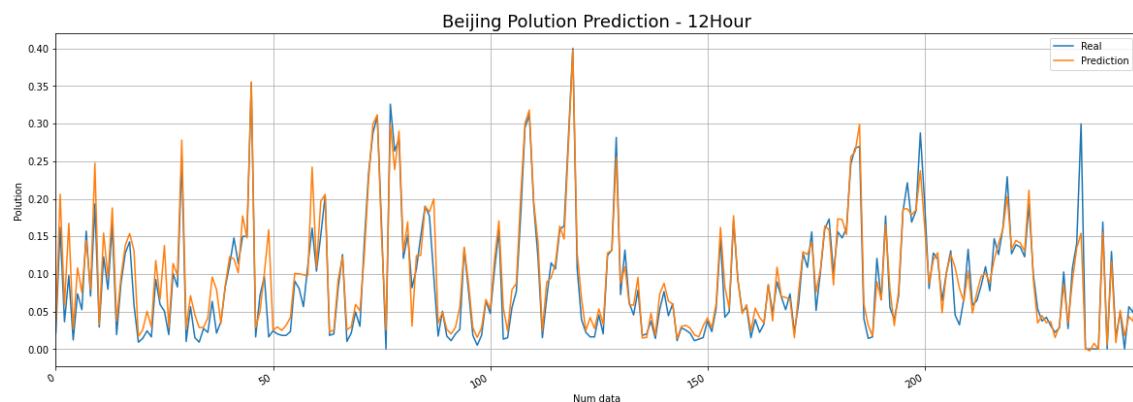
شکل 27: مقدار واقعی و پیش‌بینی شده برای تعداد 100 عدد از داده‌های تست

همچنین در ادامه نیز می‌توانید مقدار Loss مربوط به تمامی داده‌های Test را نیز مشاهده کنید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.854833362818075e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

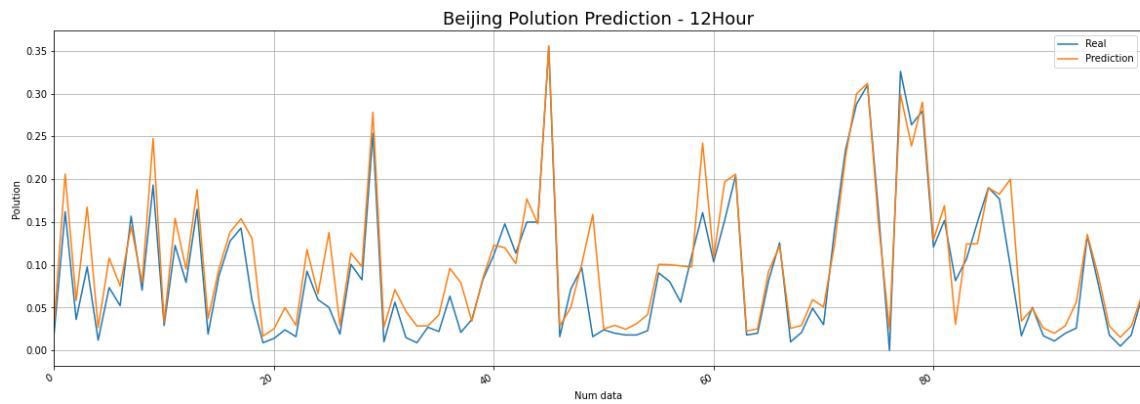
شکل 28: Loss بحسب آمده برای 3000 داده تست در مدل LSTM در حالت Adam و MSE

نمودارهای فوق مربوط به کلیه داده‌های تست بود و تمامی ساعت‌ها را شامل می‌شد. اما در ادامه من نمودار مربوط به پیش‌بینی ساعت‌های 12 و 24 را می‌آورم:



شکل 29: مقادیر واقعی و پیش‌بینی شده برای حالت ساعت‌های 12 و 24 کل داده‌های تست در مدل LSTM و حالت‌های ADAM و MSE

برای مشاهده دقیق‌تر می‌توانید در ادامه نمودار را به ازای تعداد کمتری از داده‌ها را مشاهده نمایید:



شکل 30: مقادیر واقعی و پیش‌بینی شده برای حالت ساعت‌های 12 و 24 تعدادی از داده‌های تست در مدل LSTM و حالت‌های MSE و ADAM

همچنین در ادامه می‌توانید برای مقایسه دقیق‌تر مقدار پیش‌بینی شده و واقعی موجود در دو ساعت 24 و 12 را نیز مشاهده نمایید:

	Real Values	Predicted Values
0	0.021127	0.034032
1	0.161972	0.206051
2	0.036217	0.058286
3	0.097586	0.167377
4	0.012072	0.026767
5	0.073441	0.108042
6	0.052314	0.075391
7	0.156942	0.144855
8	0.070423	0.079334
9	0.193159	0.247432
10	0.029175	0.033177
11	0.122736	0.154358
12	0.079477	0.094965
13	0.164990	0.187821
14	0.019115	0.037325
15	0.087525	0.095966
16	0.127767	0.138267
17	0.142857	0.153806
18	0.058350	0.130936
19	0.009054	0.016772
20	0.014085	0.025388
21	0.024145	0.050002
22	0.016097	0.029389

شکل 31: مقادیر واقعی و پیش‌بینی شده 22 داده در مدل LSTM در حالت ADAM و MSE

➤ می‌توانید فایل اکسل Predict_every12Hour_LSTM_ADAM_MSE.csv را به منظور مشاهده کلیه مقادیر واقعی و پیش‌بینی شده را مشاهده نمایید.

اکنون به بررسی مدل GRU می‌پردازیم:

► بررسی مدل GRU در حالت MSE و ADAM

برای این منظور ابتدا به طراحی مدل مربوطه می‌پردازیم:

```
1. class GRU(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
3.                  , n_layers=1):
4.         super(GRU, self).__init__()
5.         self.n_features = n_features
6.         self.seq_len = seq_length
7.         self.n_output = n_output
8.         self.n_hidden = n_hidden_layers # number of hidden states
9.         self.n_layers = n_layers # number of LSTM layers (stacked)
10.
11.        # define RNN with specified parameters
12.        # bath_first means that the first dim of the input and output will be the
13.          batch_size
14.          self.rnn = nn.GRU(input_size=self.n_features,
15.                             hidden_size=self.n_hidden,
16.                             num_layers=self.n_layers,
17.                             batch_first=True)
18.
19.        # last, fully connected layer
20.        self.l_linear = torch.nn.Linear(self.n_hidden*self.seq_len, self.n_output
21. )
22.
23.    def forward(self, x, hidden):
24.        # hidden_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).req
25.          uires_grad_()
26.          # cell_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requi
27.          res_grad_()
28.          batch_size = x.size(0)
29.
30.        rnn_out, hidden = self.rnn(x, hidden)
31.        rnn_out = rnn_out.contiguous().view(batch_size, -1)
32.
33.        # lstm_out(with batch_first = True) is
34.        # (batch_size,seq_len,num_directions * hidden_size)
35.        # for following linear layer we want to keep batch_size dimension and mer
36.          ge rest
37.          # .contiguous() -> solves tensor compatibility error
38.          out = self.l_linear(rnn_out)
39.          return out, hidden
```

سپس اقدام به تعیین Optimizer و تابع Loss مربوطه می‌کنم:

```

1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 32: تعریف مدل، Optimizer وتابع Loss برای GRU

سپس در ادامه می‌توانید نتیجه مربوط به مقادیر Loss را مشاهده نمایید:

در ادامه مقادیر مربوط به مقادیر 5 Loss اول را مشاهده نمایید:

```

step : 0 Train loss : 3.0134647440475723e-05 , Valid Loss : => 8.476531986768047e-05
***->>-----<<<-***  

step : 1 Train loss : 2.4077695154119282e-05 , Valid Loss : => 0.00010443098144605756
***->>-----<<<-***  

step : 2 Train loss : 2.632855320504556e-05 , Valid Loss : => 6.715554976835847e-05
***->>-----<<<-***  

step : 3 Train loss : 2.1120297877738874e-05 , Valid Loss : => 7.565707744409641e-05
***->>-----<<<-***  

step : 4 Train loss : 2.1184106264263393e-05 , Valid Loss : => 7.584338154022892e-05
***->>-----<<<-***  

step : 5 Train loss : 2.0203966142920156e-05 , Valid Loss : => 6.29526941726605e-05
***->>-----<<<-***  


```

شکل 33: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل GRU

سپس در ادامه نیز می‌توانید مقادیر مربوط به 5 آخر را مشاهده نمایید:

```

***->>-----<<<-***  

step : 93 Train loss : 4.054669407196343e-06 , Valid Loss : => 6.062935664279697e-06
***->>-----<<<-***  

step : 94 Train loss : 4.068766746786423e-06 , Valid Loss : => 6.088757644950723e-06
***->>-----<<<-***  

step : 95 Train loss : 4.060314762561272e-06 , Valid Loss : => 5.871779110748321e-06
***->>-----<<<-***  

step : 96 Train loss : 4.001589275624913e-06 , Valid Loss : => 5.653657504202177e-06
***->>-----<<<-***  

step : 97 Train loss : 3.960161799720178e-06 , Valid Loss : => 5.625945908832364e-06
***->>-----<<<-***  

step : 98 Train loss : 3.964523431689789e-06 , Valid Loss : => 5.7235529772394026e-06
***->>-----<<<-***  

step : 99 Train loss : 4.001937529149776e-06 , Valid Loss : => 5.875035116332583e-06
***->>-----<<<-***  


```

شکل 34: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل GRU

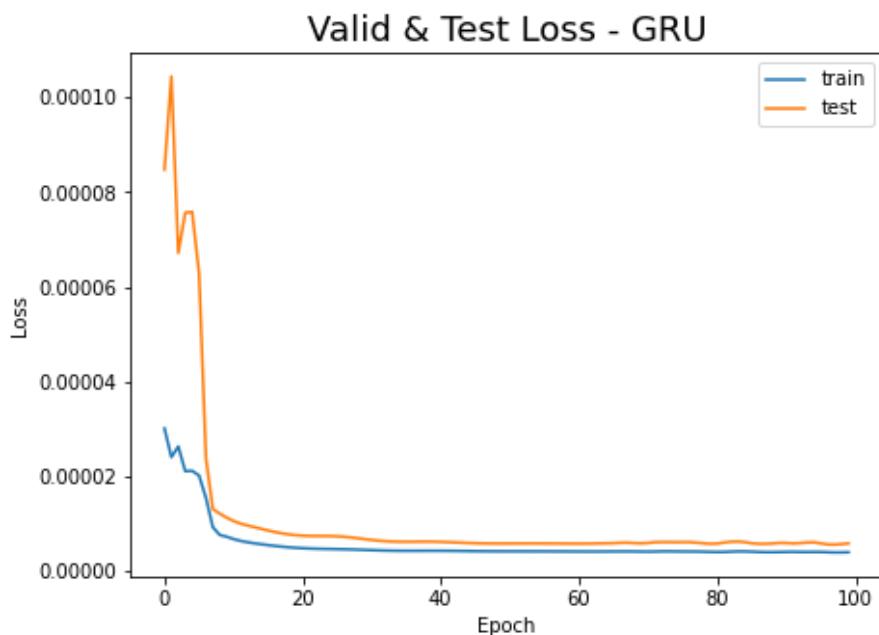
همچنین مدت زمان لازم برای آموزش این شبکه مطابق زیر است:

```

=====
*****
The total Training Time is Equal with ==> : 513.3587107658386 Sec.
*****
=====
```

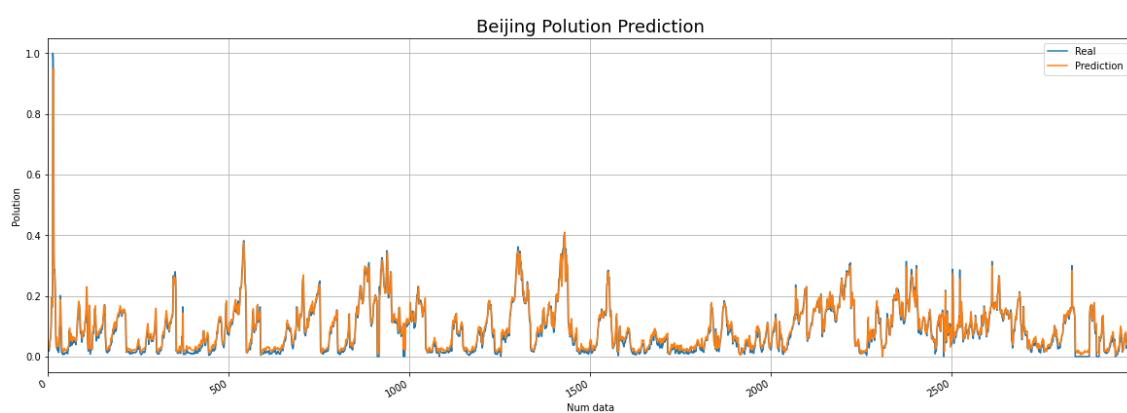
شکل 35: مدت زمان لازم برای آموزش شبکه GRU در حالت MSE و ADAM

سپس در ادامه نیز می‌توانید نمودار مقادیر Loss را برای داده‌های train و valid مشاهده نمایید:



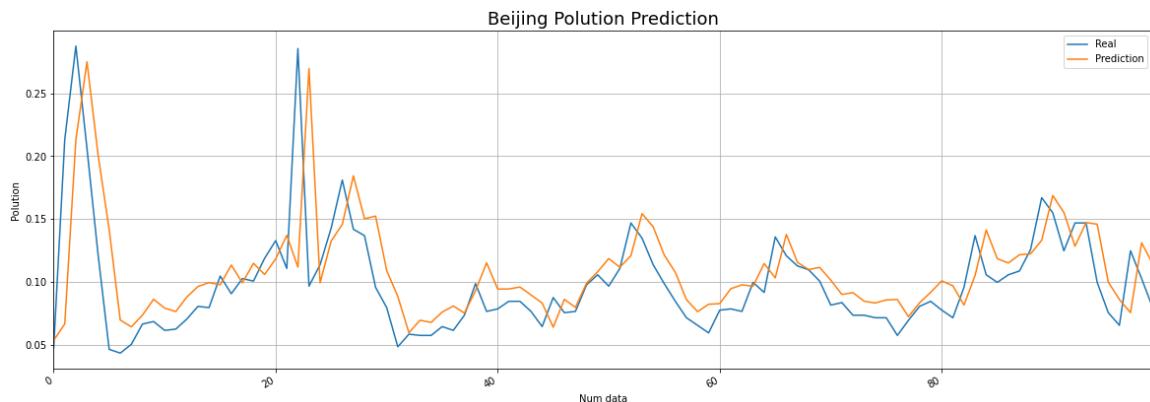
شکل 36: نمودار مربوط به Loss برای مدل GRU در حالت MSE و ADAM

در ادامه می‌توانید نمودار مربوط به مقدار واقعی و پیش‌بینی شده را برای مدل GRU مشاهده نمایید
(حالت کلیه داده‌های تست):



شکل 37: نمودار حالت پیش‌بینی و واقعی برای تمامی داده‌های تست مدل GRU در حالت MSE و ADAM

سپس برای این که بتوانیم بررسی دقیق‌تری از نحوه عملکرد این مدل داشته باشم، نمودار را به ازای 100 عدد از داده‌های تست رسم می‌کنم:



شکل 38: نمودار مربوط به مقدار واقعی و مقدار پیش‌بینی شده برای حالت GRU و MSE مدل ADAM برای 100 داده تست

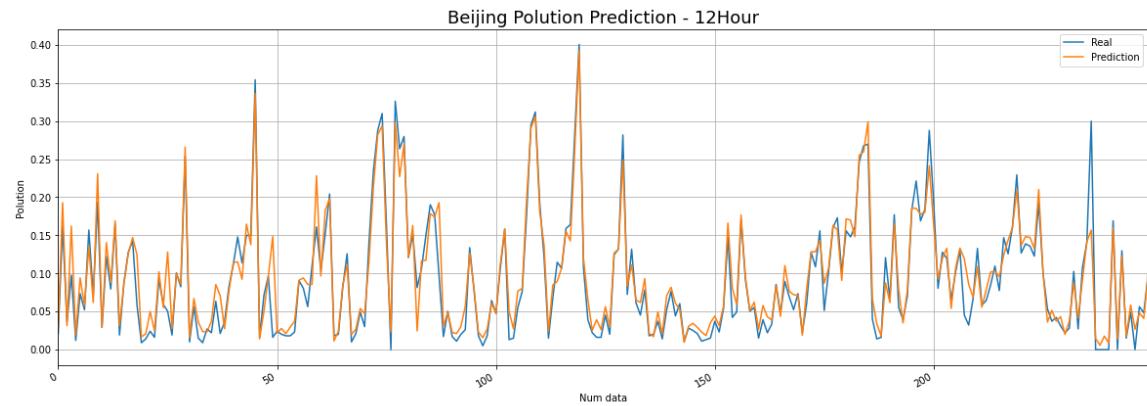
در ادامه نیز می‌توانید میزان Loss به دست آمده برای تمامی داده‌های تست را نیز مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 5.327515706691581e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 39: میزان Loss به دست آمده برای داده‌های تست در مدل GRU و حالت Adam و MSE

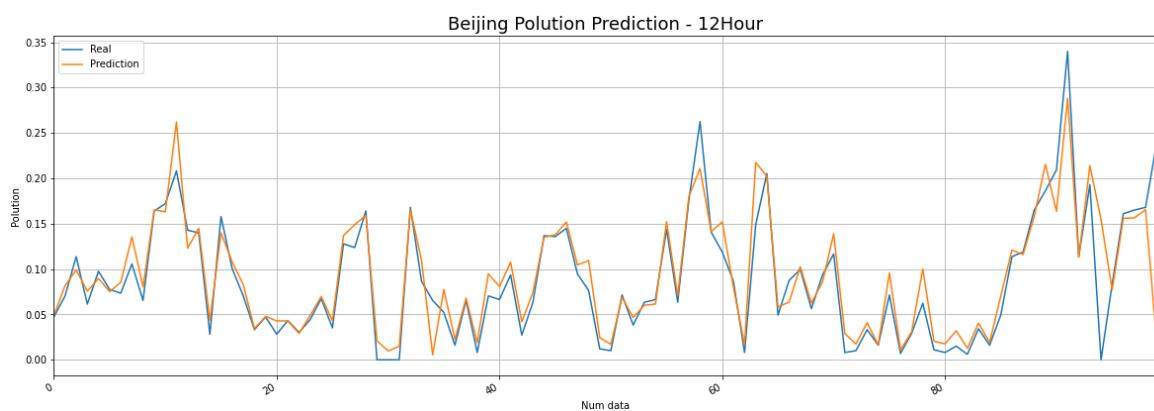
نمودارهایی که در بالا رسم شد برای حالت پیوسته و تمامی ساعت‌ها در داده‌های تست بودند. اما در ادامه من نمودار مربوط به فقط آلودگی‌های واقعی و پیش‌بینی شده در دو ساعت 12 و 24 را رسم می‌کنم: همان‌طور که در قسمت‌های قبل نیز در این مورد توضیح داده برای این تقسیم بندی از تابع `split_sequences12` استفاده کرده‌ام.

حال در ادامه می‌توانید نتایج آن را مشاهده نمایید:



شکل 40: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای MSE و ADAM برای تمامی داده‌های تست

سپس برای این که بتوانیم نگاه دقیق‌تری داشته باشیم و از نزدیک‌تر مشاهده نماییم، به ازای 100 داده تست این نمودار را رسم می‌کنیم:



شکل 41: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای MSE و ADAM برای 100 داده تست

همچنین در ادامه نیز می‌توانید این مقادیر واقعی و پیش‌بینی شده را در جدول زیر مشاهده نمایید (به ازای ساعت‌های 12 و 24):

- ✓ فایل کامل تمامی داده‌های تست به ازای ساعت‌های 12 و 24 را نیز می‌توانید در فایل [Predict_every12Hour_GRU_ADAM_MSE.csv](#) مشاهده نمایید.

	Real Values	Predicted Values
0	0.021127	0.032127
1	0.161972	0.204377
2	0.036217	0.045863
3	0.097586	0.170647
4	0.012072	0.023759
5	0.073441	0.100517
6	0.052314	0.064089
7	0.156942	0.143355
8	0.070423	0.074274
9	0.193159	0.243301
10	0.029175	0.036707
11	0.122736	0.149724
12	0.079477	0.094353
13	0.164990	0.178968
14	0.019115	0.038420
15	0.087525	0.091214
16	0.127767	0.136738
17	0.142857	0.160365
18	0.058350	0.137035
19	0.009054	0.020644
20	0.014085	0.024044
21	0.024145	0.052765

شکل 42: تعدادی از مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 داده‌های تست

اکنون پس از ساخت و آموزش هر سه مدل باید به بیان تفاوت‌های آن‌ها بپردازیم.

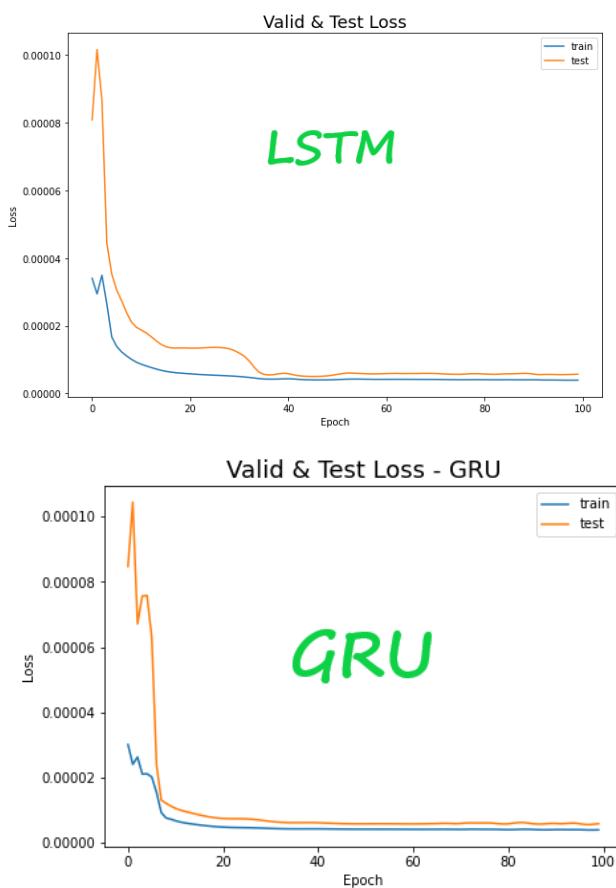
✓ مقایسه نمودار مقادیر پیش‌بینی و واقعی:

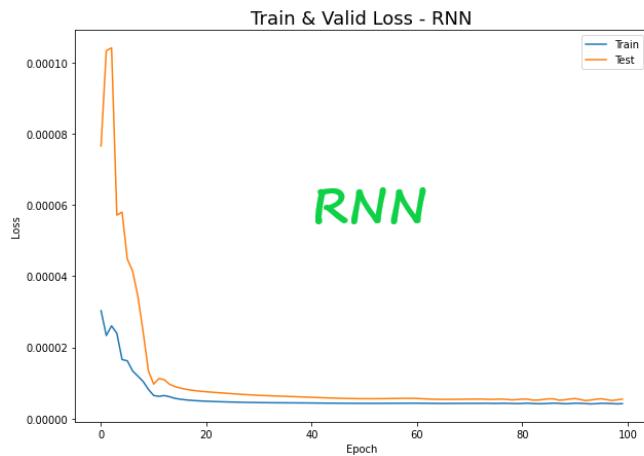
با مقایسه این نمودارها در سه حالت GRU، LSTM و RNN به این نتیجه می‌رسیم که اختلاف میان دو حالت واقعی و پیش‌بینی شده در حالت RNN از بقیه بیشتر است و از این نظر کمترین دقیقت را

دارد و نمودار مربوط به LSTM و GRU بسیار به یکدیگر نزدیک هستند و فقط کمی نمودار GRU بهتر عمل کرده است که با دقت بسیار زیاد این موضوع مشخص می‌شود.

✓ مقایسه نمودار Loss

با مقایسه نمودارهای Loss مربوط به این نتیجه می‌رسیم که نمودار Loss مربوط به GRU کمی با شبیب بهتری کاهش پیدا کرده است اما تقریباً تمامی نمودارهای مربوط به Loss در سه حالت عملکرد خوبی دارند.





شکل 43: مقایسه نمودارهای Loss مربوط به سه مدل GRU و LSTM و RNN در حالت MSE و ADAM

در شکل بالا نیز می‌توانید که این موضوع را به خوبی مشاهده کنید.

✓ مقایسه مقادیر Loss

با مقایسه مقادیر Loss آخر که در توضیحات هر شبکه آورده‌ام به این نتیجه می‌رسیم که این مقادیر برای شبکه GRU کمترین هستند و سپس برای LSTM و در آخر نیز RNN البته لازم به ذکر است که RNN مقدار یکی دوتا ایپاک آخر شاید کمتر باشد اما به دلیل نوسانات زیاد که پیشتر به آن اشاره کردم در مجموع GRU عملکرد بهتری دارد.

✓ مقایسه مقادیر Loss روی داده‌های تست:

با بررسی مقادیر Loss مربوط به داده‌های تست مطابق جدول زیر:

جدول 3. مقایسه مقادیر Loss در سه حالت

Model	Test Loss
RNN	4 . 706020466983319e-06
LSTM	4 . 854833362818075e-06
GRU	5 . 327515706691581e-06

با مقایسه مقادیر Loss به دست آمده در سه حالت به این نتیجه می‌رسیم که تقریباً سه مقدار Loss با هم برابر هستند و فقط با اختلاف جزئی Loss مربوط به RNN از بقیه کمتر است و سپس LSTM و در آخر نیز بیشتری مقدار را GRU دارد.

✓ مقایسه سرعت و زمان اجرا:

در ادامه من سرعت (زمان آموزش) هر سه شبکه را در حالت GRU و ADAM را قرار می‌دهم:

جدول 4: مقایسه سرعت آموزش سه شبکه در حالت ADAM و MSE

Network	Train Time
RNN	163.336 (s)
GRU	513.358 (s)
LSTM	688.313 (s)

❖ قسمت 3 ❖

در این قسمت از ما خواسته شده است که شبکه‌های طراحی شده در قسمت قبل یعنی RNN، GRU و LSTM را به ازای توابع LOSS و همچنین Optimizer‌های متفاوت طراحی کنیم.

در کل همه مواردی که از ما خواسته است ترکیب آن‌ها $2 \times 3 \times 3 = 18$ می‌شود که ما در قسمت قبل 3 عدد از آن‌ها را رسم کرده‌ایم و درست کرده‌ایم و بنابراین 15 تای دیگر آن‌ها باقی می‌ماند.

- لازم به ذکر است که برای بررسی این سوال من همه Hyper Parameter‌های دیگر مانند لايه‌های مخفی، نرخ یادگیری و... را طبق جدول 2 ثابت در نظر گرفته‌ام.

- همچنین خوب است که به این نکته نیز اشاره کنم که خود مدل طراحی شده و توابع و کدهای مربوط به Train و Test کردن مدل مشابه حالت‌های قبل است و بنابراین از تکرار آن‌ها خودداری می‌کنم.

در ادامه به بررسی تک‌تک حالت‌ها می‌پردازم:

► 4 - مدل RNN در حالت MAE و ADAM

در این حالت ابتدا تعریفتابع Loss و Optimizer آن را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 44: تعریف مدل، Optimizer وتابع Loss برای RNN

```
step : 0 Train loss : 0.00012969728310902913 , Valid Loss : => 0.0001897998588780562
***->>>-----<<<-***
step : 1 Train loss : 0.00013151160267492134 , Valid Loss : => 0.0001898176670074463
***->>>-----<<<-***
step : 2 Train loss : 0.00010955857622126738 , Valid Loss : => 0.00014071905240416527
***->>>-----<<<-***
step : 3 Train loss : 9.932535824676354e-05 , Valid Loss : => 0.0002566217432419459
***->>>-----<<<-***
step : 4 Train loss : 0.00010670759119093419 , Valid Loss : => 0.00010525722180803617
***->>>-----<<<-***
step : 5 Train loss : 7.35315028578043e-05 , Valid Loss : => 0.00012994851420323054
***->>>-----<<<-***
```

شکل 45: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل RNN

```
***->>>-----<<<-***
step : 93 Train loss : 3.878715721269449e-05 , Valid Loss : => 3.5977158695459366e-05
***->>>-----<<<-***
step : 94 Train loss : 3.59136742229263e-05 , Valid Loss : => 5.4682916030287745e-05
***->>>-----<<<-***
step : 95 Train loss : 4.0436020183066526e-05 , Valid Loss : => 3.576596826314926e-05
***->>>-----<<<-***
step : 96 Train loss : 3.752044488986333e-05 , Valid Loss : => 5.281322014828523e-05
***->>>-----<<<-***
step : 97 Train loss : 4.103778054316839e-05 , Valid Loss : => 3.7262892971436185e-05
***->>>-----<<<-***
step : 98 Train loss : 3.812354759623607e-05 , Valid Loss : => 5.079396814107895e-05
***->>>-----<<<-***
step : 99 Train loss : 3.975860755890608e-05 , Valid Loss : => 3.527899272739887e-05
***->>>-----<<<-***
```

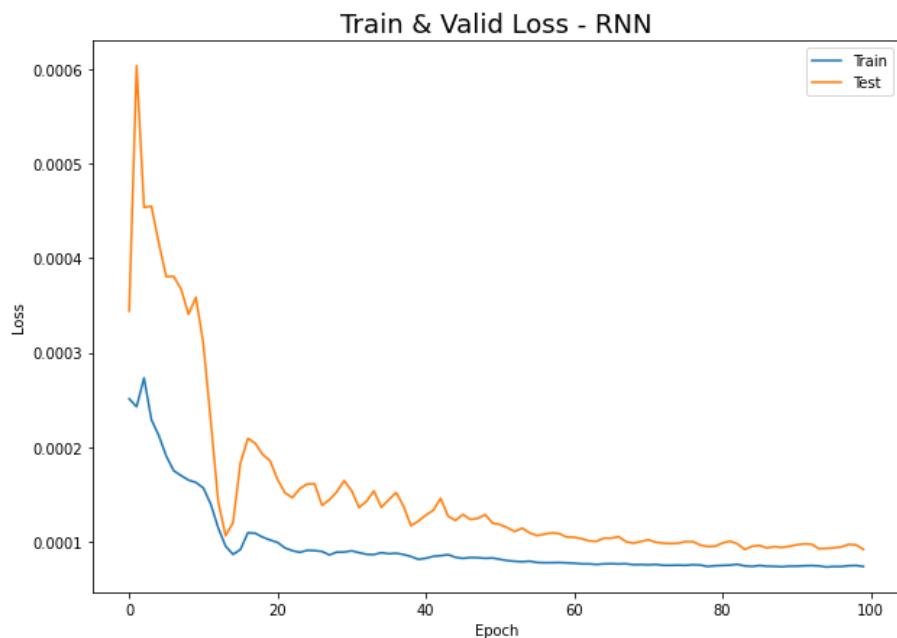
شکل 46: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل RNN

سپس در ادامه نیز می‌توانید زمان اجرای آموزش این مدل را نیز مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 161.41144227981567 Sec.
*****
=====
```

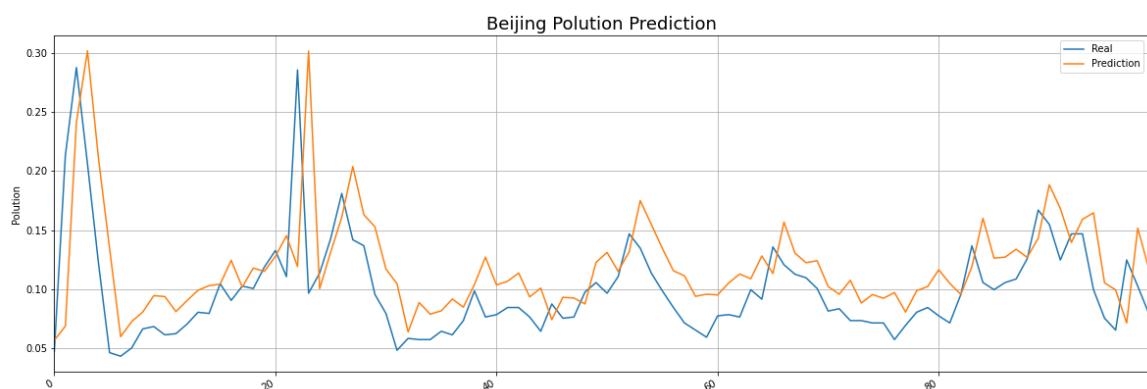
شکل 47: زمان آموزش مربوط به مدل RNN و حالت MAE و ADAM

سپس در ادامه می‌توانید نمودار Loss را در داده‌های آموزشی و valid را مشاهده نمایید:



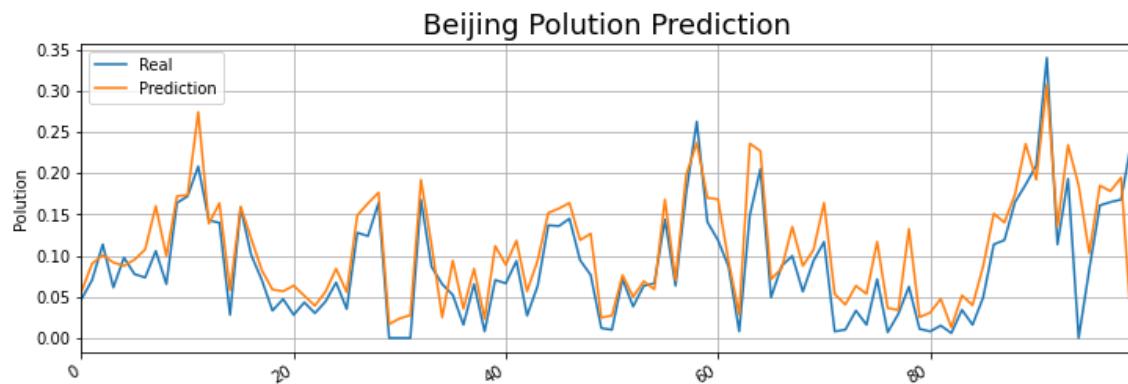
شکل 48: نمودار مربوط به Loss برای مدل RNN در حالت MAE و ADAM

سپس در ادامه می‌توانید که نمودار مقادیر واقعی و پیش‌بینی را به ازای 100 عدد از داده‌ها به صورت ساعت‌های پیوسته مشاهده نمایید:



شکل 49: نمودار مربوط به مقدار واقعی و مقدار پیش‌بینی شده برای حالت MAE و ADAM برای 100 داده تست

در ادامه نیز می‌توانید این نمودار را برای حالتی که داده‌های برای ساعت‌های 12 و 24 را می‌خواهیم پیش‌بینی کنیم را مشاهده کنید:



شکل ۵۰: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های ۱۲ و ۲۴ در مدل RNN به ازای MAE و ADAM برای ۱۰۰ داده تست

سپس در ادامه نیز می‌توانید که جدول مربوط به تعدادی از این پیش‌بینی‌ها برای ساعت‌های ۱۲ و ۲۴ را مشاهده نمایید:

	Real Values	Predicted Values
0	0.021127	0.013531
1	0.161972	0.184002
2	0.036217	0.011693
3	0.097586	0.152886
4	0.012072	0.016144
5	0.073441	0.106764
6	0.052314	0.071675
7	0.156942	0.137850
8	0.070423	0.067809
9	0.193159	0.249171
10	0.029175	0.030542
11	0.122736	0.136221
12	0.079477	0.089147
13	0.164990	0.177095
14	0.019115	0.015634
15	0.087525	0.085954
16	0.127767	0.125827
17	0.142857	0.147720
18	0.058350	0.125336
19	0.009054	0.001728
20	0.014085	0.011964
21	0.024145	0.029316

شکل 51: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

سپس در ادامه نیز می‌توانید مقدار Loss محاسبه شده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 5.9799636093278725e-05
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 52: مقدار Loss به دست آمده برای داده‌های تست در مدل RNN و حالت MAE و ADAM

۵- مدل RNN در حالت MSE و AdaGrad ➤

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 53: تعریف مدل، Loss و تابع Optimizer برای RNN

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 3.451949867109458e-05 , Valid Loss : => 5.2610844916974506e-05
***->>>-----<<-***
step : 1 Train loss : 2.5161962257698177e-05 , Valid Loss : => 4.9613953878482184e-05
***->>>-----<<-***
step : 2 Train loss : 2.234493423408518e-05 , Valid Loss : => 4.6184802505498134e-05
***->>>-----<<-***
step : 3 Train loss : 2.0588835961340617e-05 , Valid Loss : => 4.237484093755484e-05
***->>>-----<<-***
step : 4 Train loss : 1.9231820564406614e-05 , Valid Loss : => 3.867904686679443e-05
***->>>-----<<-***
step : 5 Train loss : 1.810048824020972e-05 , Valid Loss : => 3.5393401825179655e-05
***->>>-----<<-***
```

شکل 54: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل RNN

```

***_>>>-----<<<-***  

step : 93 Train loss : 8.64947383136799e-06 , Valid Loss : => 1.4707170970117053e-05  

***_>>>-----<<<-***  

step : 94 Train loss : 8.626661315793171e-06 , Valid Loss : => 1.4665899837079148e-05  

***_>>>-----<<<-***  

step : 95 Train loss : 8.604207477765158e-06 , Valid Loss : => 1.4625172363594175e-05  

***_>>>-----<<<-***  

step : 96 Train loss : 8.582101797219366e-06 , Valid Loss : => 1.4584971553025146e-05  

***_>>>-----<<<-***  

step : 97 Train loss : 8.560333765732745e-06 , Valid Loss : => 1.45452832027028e-05  

***_>>>-----<<<-***  

step : 98 Train loss : 8.53889454350186e-06 , Valid Loss : => 1.4506093963670234e-05  

***_>>>-----<<<-***  

step : 99 Train loss : 8.517773965528856e-06 , Valid Loss : => 1.4467387149731318e-05  

***_>>>-----<<<-***
```

شکل 55: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل RNN

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```

=====
*****  

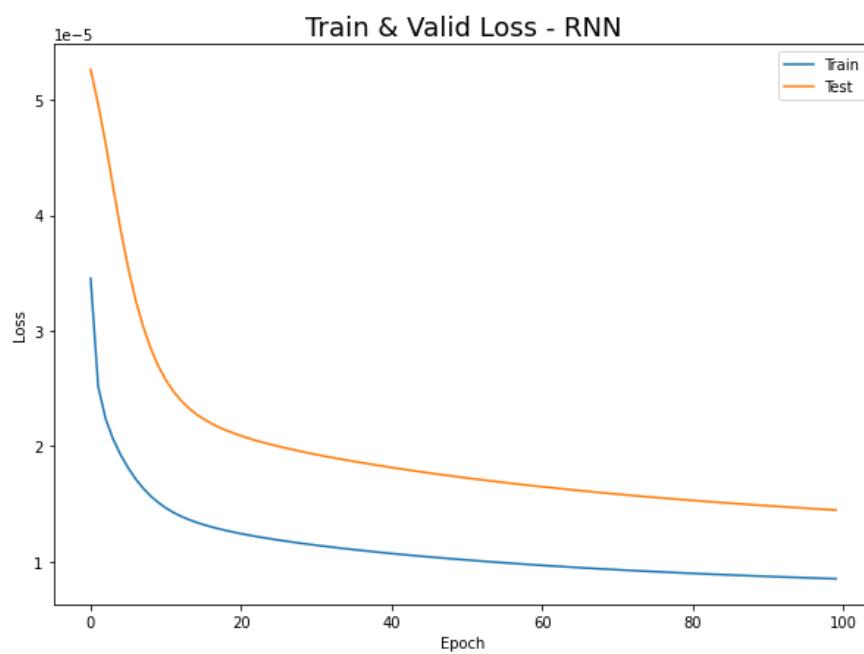
The total Training Time is Equal with ==> : 178.83883047103882 Sec.  

*****  

=====
```

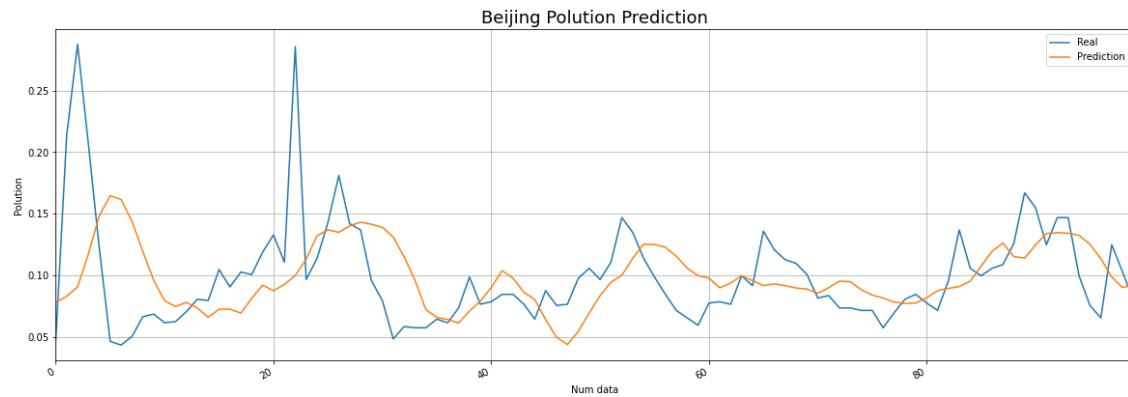
شکل 56: زمان آموزش مدل RNN در حالت AdaGrad و MSE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Train و Valid را قرار می‌دهم:



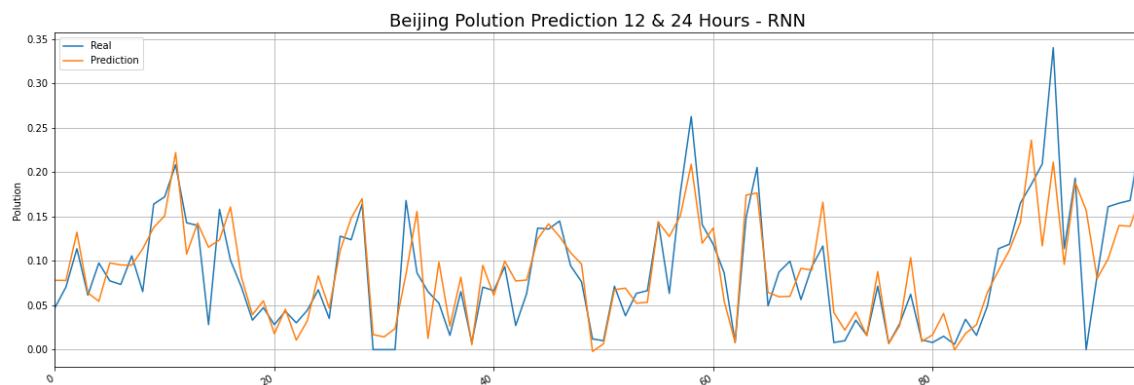
شکل 57: نمودار Loss مربوط به حالت Valid و Test در مدل RNN و حالت AdaGrad و MSE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 58: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل RNN و حالت MSE و AdaGrad

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 59: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل RNN به ازای MSE و AdaGrad برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
*** Test Loss :=>> 9.137879557480725e-06
>>>-----*****-----<<<
>>>-----<<<
#####
#####
```

شکل 60: مقدار Loss به دست آمده برای مدل RNN در حالت MSE و AdaGrad برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.078063
1	0.070423	0.077935
2	0.113682	0.132172
3	0.061368	0.064066
4	0.097586	0.054469
5	0.077465	0.097697
6	0.073441	0.095335
7	0.105634	0.095221
8	0.065392	0.113541
9	0.163984	0.137603
10	0.172032	0.150745
11	0.208249	0.221909
12	0.142857	0.107645
13	0.139839	0.142621
14	0.028169	0.115372
15	0.157948	0.123700
16	0.100604	0.160583
17	0.070423	0.081597
18	0.033199	0.039349
19	0.047284	0.054963
20	0.028169	0.017936
21	0.043260	0.045571
22	0.030181	0.010721

شکل 61: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 6 - مدل RNN در حالت MAE و AdaGrad

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 62: تعریف مدل، Optimizer و تابع Loss برای RNN

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 0.0002779823778818051 , Valid Loss : => 0.00036049910510579746
***->>>-----<<<-***  
step : 1 Train loss : 0.0002344992900888125 , Valid Loss : => 0.00035305453340212504
***->>>-----<<<-***  
step : 2 Train loss : 0.00022211307634909948 , Valid Loss : => 0.00034276389330625535
***->>>-----<<<-***  
step : 3 Train loss : 0.00021325068635245163 , Valid Loss : => 0.00032775470117727914
***->>>-----<<<-***  
step : 4 Train loss : 0.0002053158786147833 , Valid Loss : => 0.0003116802324851354
***->>>-----<<<-***  
step : 5 Train loss : 0.00019805853366851808 , Valid Loss : => 0.0002955681309103966
***->>>-----<<<-***
```

شکل 63: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل RNN

```
***->>>-----<<<-***  
step : 93 Train loss : 0.00011721486176053682 , Valid Loss : => 0.00015257593741019567
***->>>-----<<<-***  
step : 94 Train loss : 0.00011701564236233631 , Valid Loss : => 0.00015220380077759425
***->>>-----<<<-***  
step : 95 Train loss : 0.00011681350202610096 , Valid Loss : => 0.00015182136309643588
***->>>-----<<<-***  
step : 96 Train loss : 0.00011661613906423251 , Valid Loss : => 0.0001516059475640456
***->>>-----<<<-***  
step : 97 Train loss : 0.00011642539935807387 , Valid Loss : => 0.00015131462489565215
***->>>-----<<<-***  
step : 98 Train loss : 0.00011623012317965428 , Valid Loss : => 0.00015102067155142626
***->>>-----<<<-***  
step : 99 Train loss : 0.00011605972368270158 , Valid Loss : => 0.0001507290005683899
***->>>-----<<<-***
```

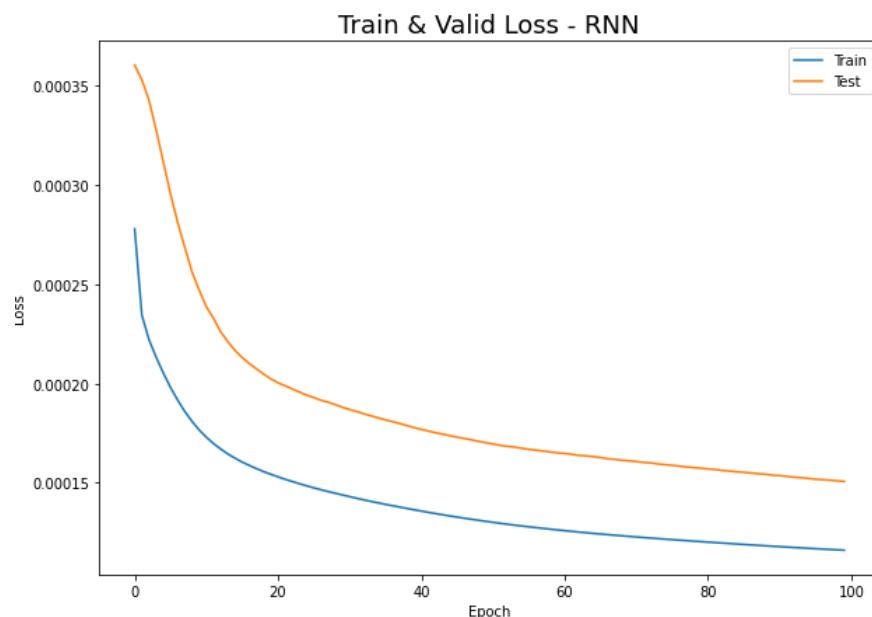
شکل 64: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل RNN

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 163.64319777488708 Sec.
*****
=====
```

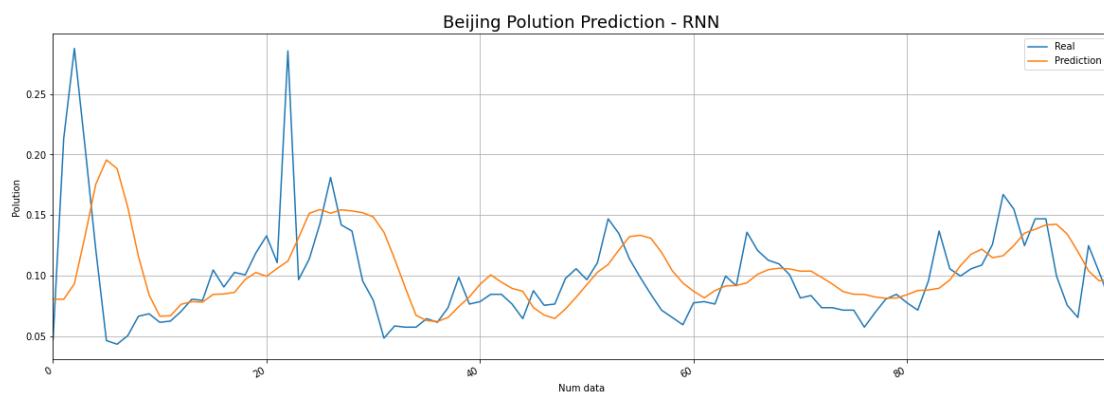
شکل 65: زمان آموزش مدل RNN در حالت AdaGrad و MAE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Train و Valid را قرار می‌دهم:



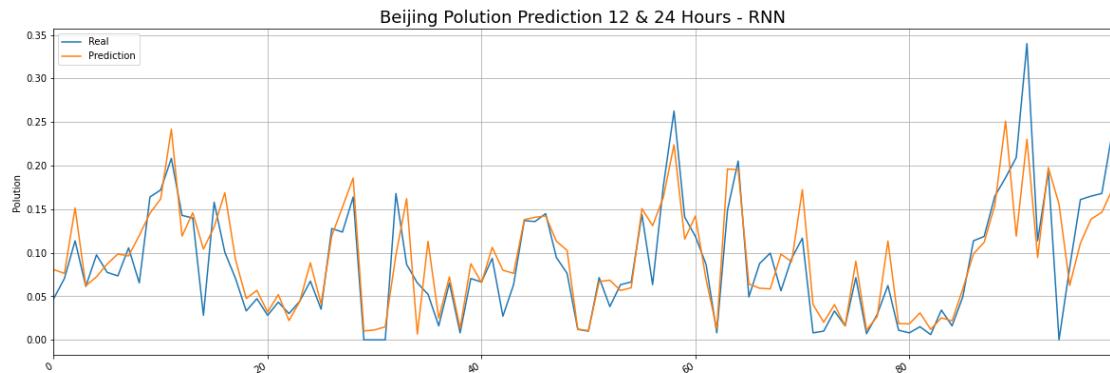
شکل 66: نمودار Loss مربوط به حالت Valid و Test در مدل RNN و حالت AdaGrad و MAE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 67: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل RNN و حالت AdaGrad و MAE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 68: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل RNN به ازای MAE و AdaGrad برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.00011676602003475031
>>>-----*****-----<<<
>>>-----<<<
#####
#
```

شکل 69: مقدار Loss به دست آمده برای مدل RNN در حالت AdaGrad و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.080630
1	0.070423	0.076298
2	0.113682	0.151452
3	0.061368	0.062032
4	0.097586	0.072434
5	0.077465	0.087073
6	0.073441	0.098512
7	0.105634	0.096325
8	0.065392	0.119785
9	0.163984	0.145215
10	0.172032	0.161729
11	0.208249	0.241787
12	0.142857	0.119040
13	0.139839	0.145859
14	0.028169	0.104216
15	0.157948	0.129465
16	0.100604	0.168863
17	0.070423	0.092047
18	0.033199	0.047428
19	0.047284	0.056800
20	0.028169	0.031836
21	0.043260	0.051946
22	0.030181	0.022138

شکل 70: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

MSE و RMSprop در حالت RNN - 7 ➤

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 71: تعریف مدل، Loss و تابع Optimizer برای RNN

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 5.672690744977445e-05 , Valid Loss : => 4.557873161199192e-05
***->>>-----<<<-***  
step : 1 Train loss : 1.8268974765669554e-05 , Valid Loss : => 3.242410874615113e-05
***->>>-----<<<-***  
step : 2 Train loss : 1.4062165756088991e-05 , Valid Loss : => 2.2270036628469824e-05
***->>>-----<<<-***  
step : 3 Train loss : 1.2055692131010195e-05 , Valid Loss : => 1.771205838304013e-05
***->>>-----<<<-***  
step : 4 Train loss : 1.0685449272083739e-05 , Valid Loss : => 1.4725571187833944e-05
***->>>-----<<<-***  
step : 5 Train loss : 9.594818794478972e-06 , Valid Loss : => 1.2573036248795687e-05
***->>>-----<<<-***
```

شکل 72: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل RNN

```
***->>>-----<<<-***  
step : 93 Train loss : 4.139065943309106e-06 , Valid Loss : => 4.961059428751469e-06
***->>>-----<<<-***  
step : 94 Train loss : 4.145111834319929e-06 , Valid Loss : => 4.966081692449128e-06
***->>>-----<<<-***  
step : 95 Train loss : 4.139611532445997e-06 , Valid Loss : => 4.957699391525238e-06
***->>>-----<<<-***  
step : 96 Train loss : 4.138477270801862e-06 , Valid Loss : => 4.957048697785164e-06
***->>>-----<<<-***  
step : 97 Train loss : 4.124734265496954e-06 , Valid Loss : => 4.95138699382854e-06
***->>>-----<<<-***  
step : 98 Train loss : 4.122676045517437e-06 , Valid Loss : => 4.954226033684487e-06
***->>>-----<<<-***  
step : 99 Train loss : 4.1237723033797615e-06 , Valid Loss : => 4.948425630573183e-06
***->>>-----<<<-***
```

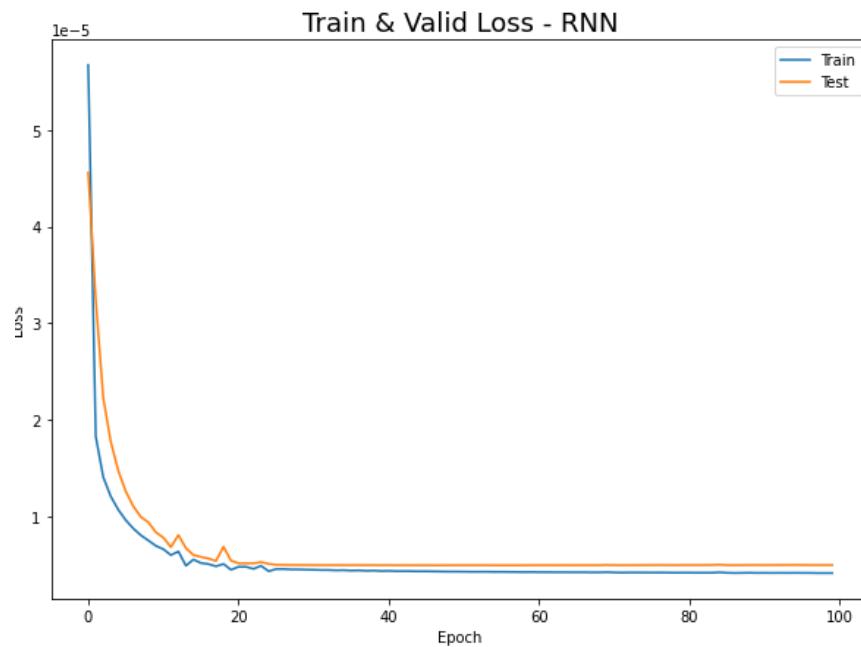
شکل 73: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل RNN

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 176.53414273262024 Sec.
*****
=====
```

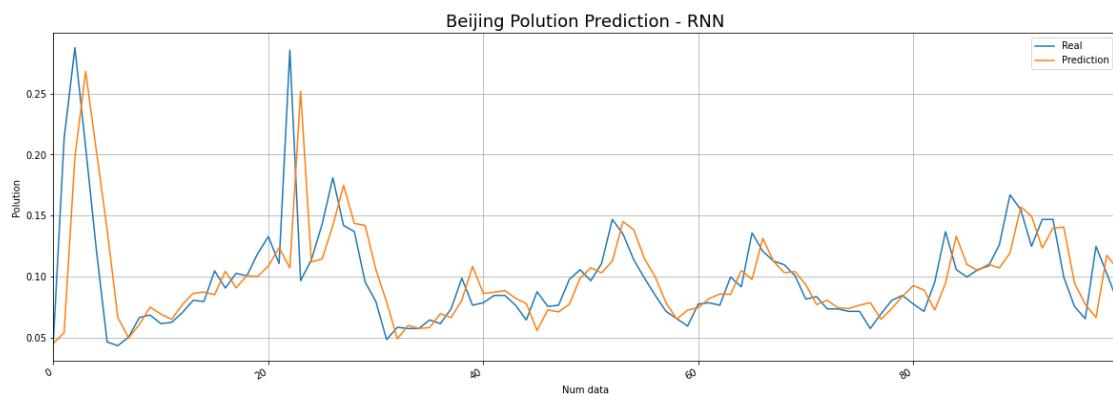
شکل 74: زمان آموزش مدل RNN در حالت RMSprop و MSE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



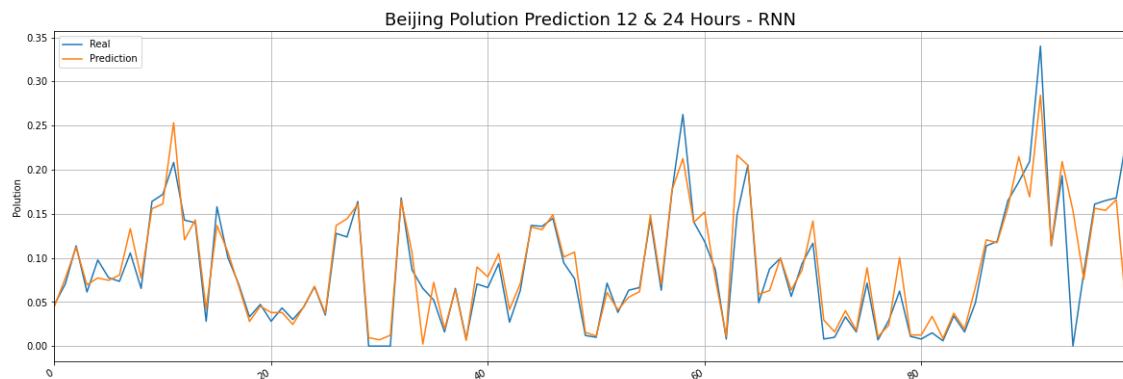
شکل 75: نمودار Loss مربوط به حالت Valid و حالت Test در مدل RNN و RMSprop و MSE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 76: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل RNN و RMSprop و MSE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 77: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل RNN به ازای MSE و RMSprop برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.030287971545476e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 78: مقدار Loss به دست آمده برای مدل RNN در حالت MSE و RMSprop برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.045080
1	0.070423	0.077002
2	0.113682	0.111921
3	0.061368	0.069490
4	0.097586	0.077286
5	0.077465	0.074578
6	0.073441	0.080485
7	0.105634	0.133134
8	0.065392	0.077187
9	0.163984	0.155697
10	0.172032	0.161201
11	0.208249	0.253275
12	0.142857	0.120481
13	0.139839	0.143051
14	0.028169	0.041966
15	0.157948	0.136879
16	0.100604	0.106942
17	0.070423	0.068363
18	0.033199	0.028030
19	0.047284	0.045257
20	0.028169	0.038046
21	0.043260	0.038204
22	0.030181	0.024561

شکل 79: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

۸ - مدل RNN در حالت MAE و RMSprop ➤

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)

```

شکل 80: تعریف مدل، Loss و تابع Optimizer برای RNN

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```

step : 1 Train loss : 0.00021089223970969517 , Valid Loss : => 0.00022876766013602416
***->>>-----<<<-***  

step : 2 Train loss : 0.00017355747148394584 , Valid Loss : => 0.00018288093619048596
***->>>-----<<<-***  

step : 3 Train loss : 0.00015090624305109183 , Valid Loss : => 0.00015172208473086357
***->>>-----<<<-***  

step : 4 Train loss : 0.00013686749655753374 , Valid Loss : => 0.00013474349739650885
***->>>-----<<<-***  

step : 5 Train loss : 0.00012669291055450837 , Valid Loss : => 0.00012031986936926841
***->>>-----<<<-***  


```

شکل 81: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل RNN

```

***->>>-----<<<-***  

step : 93 Train loss : 7.106371245657404e-05 , Valid Loss : => 8.405264249692361e-05
***->>>-----<<<-***  

step : 94 Train loss : 7.093205197403828e-05 , Valid Loss : => 8.130407364418109e-05
***->>>-----<<<-***  

step : 95 Train loss : 7.187162435924013e-05 , Valid Loss : => 9.134285990148783e-05
***->>>-----<<<-***  

step : 96 Train loss : 7.130323890596628e-05 , Valid Loss : => 7.99770684291919e-05
***->>>-----<<<-***  

step : 97 Train loss : 7.13921400718391e-05 , Valid Loss : => 9.156376961618662e-05
***->>>-----<<<-***  

step : 98 Train loss : 7.151534225170812e-05 , Valid Loss : => 7.994026225060224e-05
***->>>-----<<<-***  

step : 99 Train loss : 7.065622148414452e-05 , Valid Loss : => 8.447386355449757e-05
***->>>-----<<<-***  


```

شکل 82: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل RNN

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```

=====
*****  

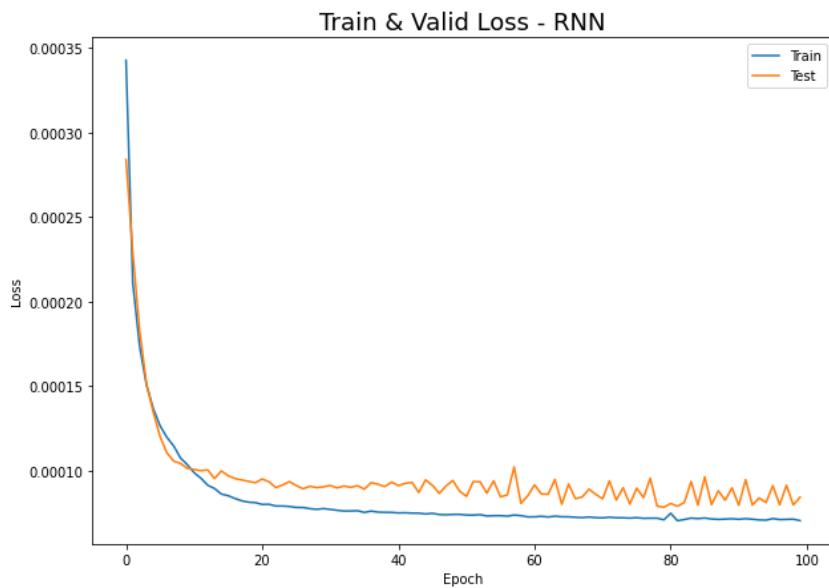
The total Training Time is Equal with ==> : 178.9830629825592 Sec.  

*****  

=====
```

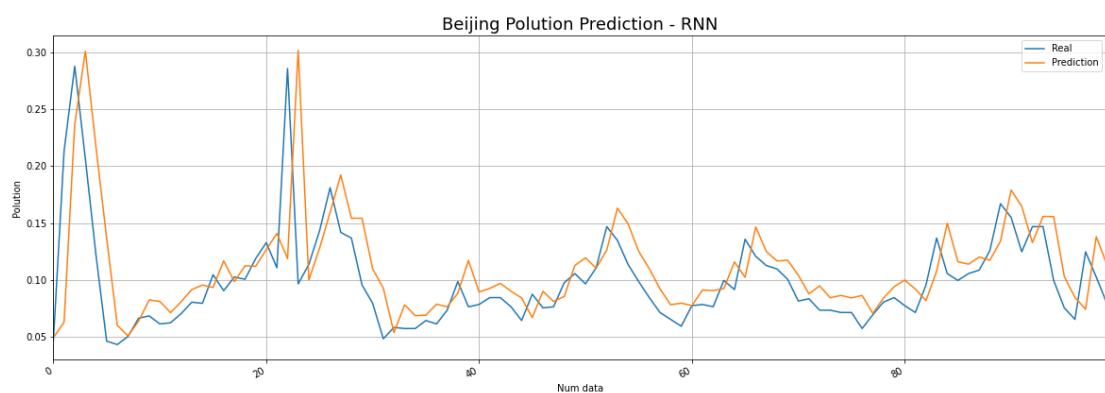
شکل 83: زمان آموزش مدل RNN در حالت MAE و RMSprop در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Train و Valid را قرار می‌دهم:



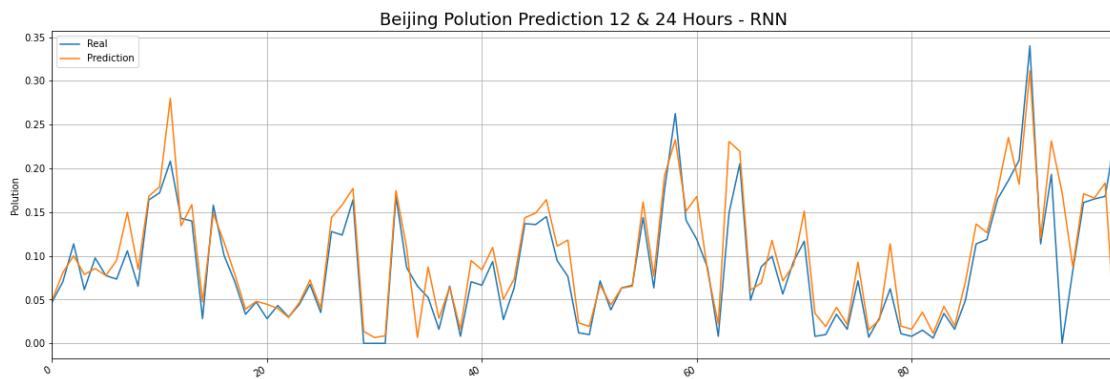
شکل 84: نمودار Loss مربوط به حالت Valid و Test در مدل RNN و حالت MAE و RMSprop

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 85: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل RNN و حالت MAE و RMSprop

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 86: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل RNN به ازای RMSprop و MAE برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 7.268758537247778e-05
>>>-----*****-----<<<
>>>-----<<<
#####
#####
```

شکل 87: مقدار Loss به دست آمده برای مدل RNN در حالت RMSprop و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.049360
1	0.070423	0.080881
2	0.113682	0.100014
3	0.061368	0.078730
4	0.097586	0.085510
5	0.077465	0.077516
6	0.073441	0.094830
7	0.105634	0.149808
8	0.065392	0.084626
9	0.163984	0.168132
10	0.172032	0.178882
11	0.208249	0.280070
12	0.142857	0.134266
13	0.139839	0.158413
14	0.028169	0.047389
15	0.157948	0.148556
16	0.100604	0.115631
17	0.070423	0.078922
18	0.033199	0.039023
19	0.047284	0.048011
20	0.028169	0.044644
21	0.043260	0.039933
22	0.030181	0.029590

شکل 8: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 9 - مدل LSTM در حالت MAE و ADAM

در این حالت در ابتدا تعریف تابع Loss و Optimizer مربوطه را قرار می‌دهم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 89: تعریف مدل، Loss و Optimizer برای LSTM

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```

step : 0 Train loss : 0.00027704122165838875 , Valid Loss : => 0.000449042355020841
***->>>-----<<<-***>
step : 1 Train loss : 0.00025477446566025416 , Valid Loss : => 0.0005258952528238297
***->>>-----<<<-***>
step : 2 Train loss : 0.00026649090896050135 , Valid Loss : => 0.0004878516358633836
***->>>-----<<<-***>
step : 3 Train loss : 0.00026181539048751194 , Valid Loss : => 0.0004824184129635493
***->>>-----<<<-***>
step : 4 Train loss : 0.0002443061782668034 , Valid Loss : => 0.00039543235550324124
***->>>-----<<<-***>
step : 5 Train loss : 0.00020639932428797087 , Valid Loss : => 0.0003113922029733658
***->>>-----<<<-***>

```

شکل 90: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل LSTM

```

***->>>-----<<<-***>
step : 93 Train loss : 7.33949141887327e-05 , Valid Loss : => 9.755355802675089e-05
***->>>-----<<<-***>
step : 94 Train loss : 7.34765335607032e-05 , Valid Loss : => 9.620948353161415e-05
***->>>-----<<<-***>
step : 95 Train loss : 7.322319603214661e-05 , Valid Loss : => 9.599540041138728e-05
***->>>-----<<<-***>
step : 96 Train loss : 7.318675288309654e-05 , Valid Loss : => 9.665625045696894e-05
***->>>-----<<<-***>
step : 97 Train loss : 7.329193651676178e-05 , Valid Loss : => 9.617557159314553e-05
***->>>-----<<<-***>
step : 98 Train loss : 7.321183973302444e-05 , Valid Loss : => 9.561609290540219e-05
***->>>-----<<<-***>
step : 99 Train loss : 7.316476292908192e-05 , Valid Loss : => 9.653084073215723e-05
***->>>-----<<<-***>

```

شکل 91: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل LSTM

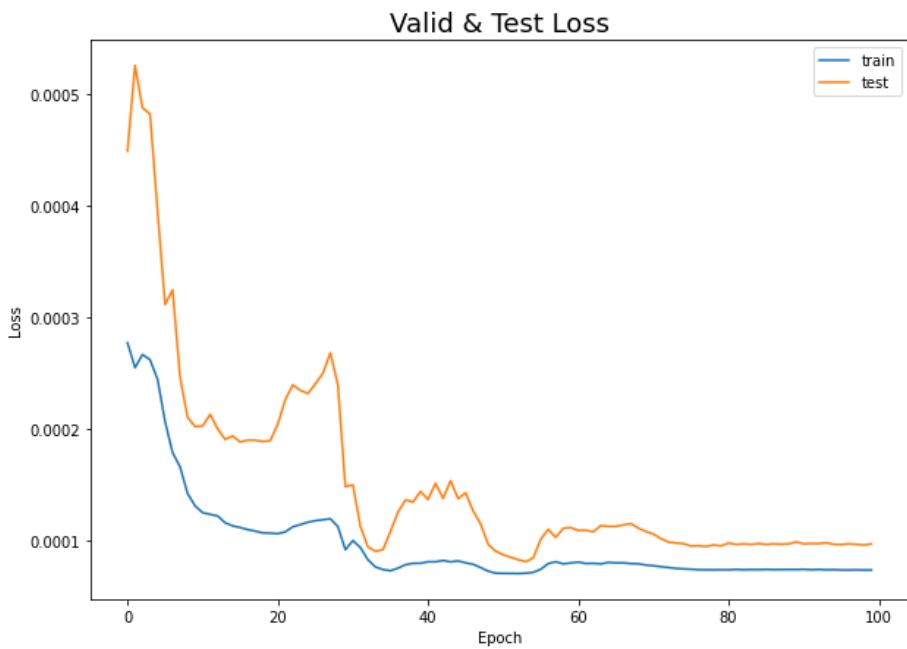
سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 678.7958512306213 Sec.
*****
=====
```

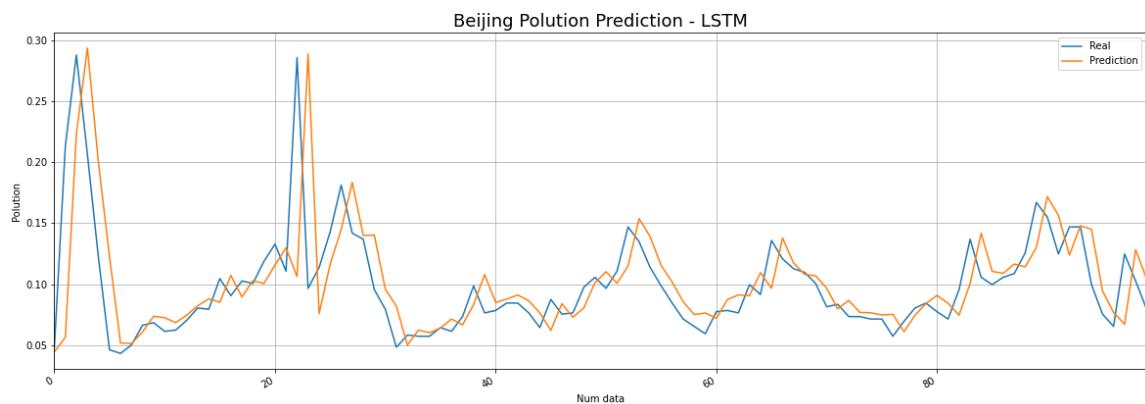
شکل 92: زمان آموزش مدل LSTM در حالت MAE و ADAM در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Train و Valid را قرار می‌دهم:



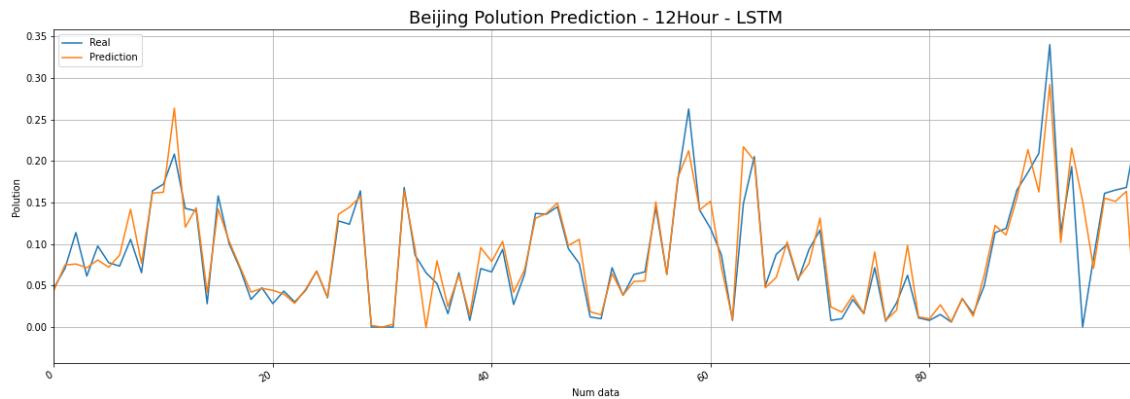
شکل 93: نمودار Loss مربوط به حالت Valid و Test در مدل LSTM و حالت MAE و ADAM

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 94: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل LSTM و حالت MAE و ADAM

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل ۹۵: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های ۱۲ و ۲۴ در مدل LSTM به ازای ADAM و MAE برای ۱۰۰ داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 9.078168341269095e-05
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل ۹۶: مقدار Loss به دست آمده برای مدل LSTM در حالت Adam و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های ۱۲ و ۲۴ را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.044428
1	0.070423	0.074683
2	0.113682	0.075874
3	0.061368	0.071433
4	0.097586	0.080716
5	0.077465	0.072002
6	0.073441	0.086699
7	0.105634	0.141742
8	0.065392	0.076830
9	0.163984	0.161300
10	0.172032	0.162274
11	0.208249	0.263485
12	0.142857	0.120316
13	0.139839	0.143560
14	0.028169	0.041599
15	0.157948	0.142082
16	0.100604	0.103632
17	0.070423	0.072496
18	0.033199	0.042033
19	0.047284	0.046611
20	0.028169	0.044182
21	0.043260	0.039759
22	0.030181	0.028585

شکل ۹۷: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های ۱۲ و ۲۴

► ۱۰ - مدل LSTM در حالت MSE و AdaGrad

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 98: تعریف مدل، Loss و Optimizer برای LSTM

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```

step : 0 Train loss : 3.8364940990383426e-05 , Valid Loss : => 6.599482047992449e-05
***->>>-----<<<-***>
step : 1 Train loss : 3.517691114296516e-05 , Valid Loss : => 6.260063328469793e-05
***->>>-----<<<-***>
step : 2 Train loss : 3.374268711389353e-05 , Valid Loss : => 6.0043433603520196e-05
***->>>-----<<<-***>
step : 3 Train loss : 3.259101683118691e-05 , Valid Loss : => 5.788928712718189e-05
***->>>-----<<<-***>
step : 4 Train loss : 3.158861263655126e-05 , Valid Loss : => 5.5976903764531014e-05
***->>>-----<<<-***>
step : 5 Train loss : 3.068287964755048e-05 , Valid Loss : => 5.422968363078932e-05
***->>>-----<<<-***>

```

شکل 99: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل LSTM

```

***->>>-----<<<-***>
step : 93 Train loss : 1.377577685828631e-05 , Valid Loss : => 2.3149797537674507e-05
***->>>-----<<<-***>
step : 94 Train loss : 1.3746340665966272e-05 , Valid Loss : => 2.3103095203017194e-05
***->>>-----<<<-***>
step : 95 Train loss : 1.3717266804693887e-05 , Valid Loss : => 2.305677874634663e-05
***->>>-----<<<-***>
step : 96 Train loss : 1.368854381920149e-05 , Valid Loss : => 2.301083606046935e-05
***->>>-----<<<-***>
step : 97 Train loss : 1.3660159222005556e-05 , Valid Loss : => 2.296525853065153e-05
***->>>-----<<<-***>
step : 98 Train loss : 1.3632101833354682e-05 , Valid Loss : => 2.2920035912344852e-05
***->>>-----<<<-***>
step : 99 Train loss : 1.3604360655881464e-05 , Valid Loss : => 2.2875160754968724e-05
***->>>-----<<<-***>

```

شکل 100: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل LSTM

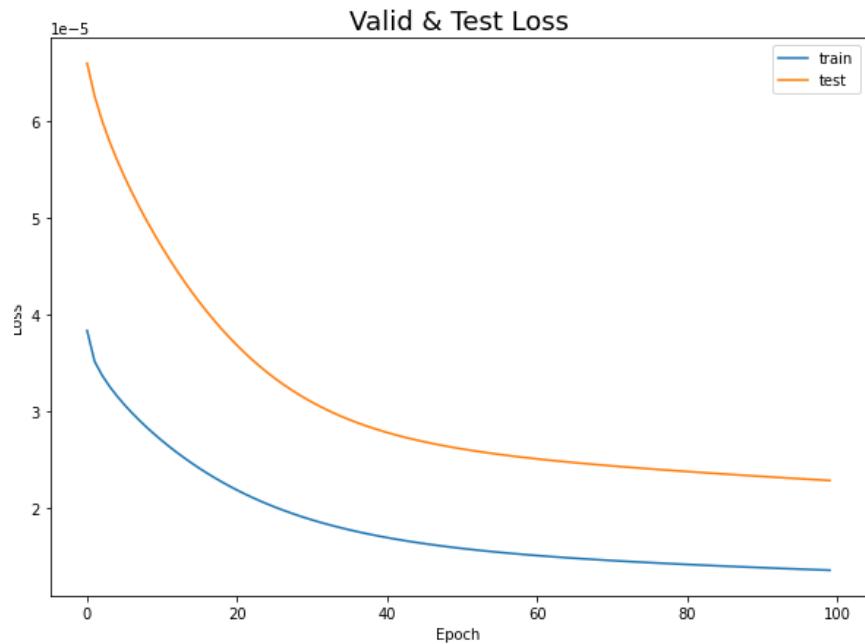
سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 768.7854142189026 Sec.
*****
=====
```

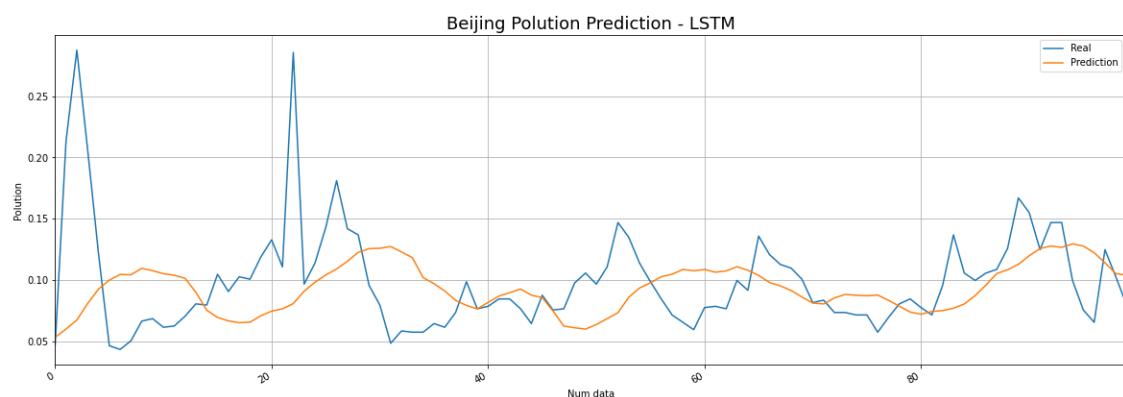
شکل 101: زمان آموزش مدل LSTM در حالت AdaGrad و MSE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Train و Valid را قرار می‌دهم:



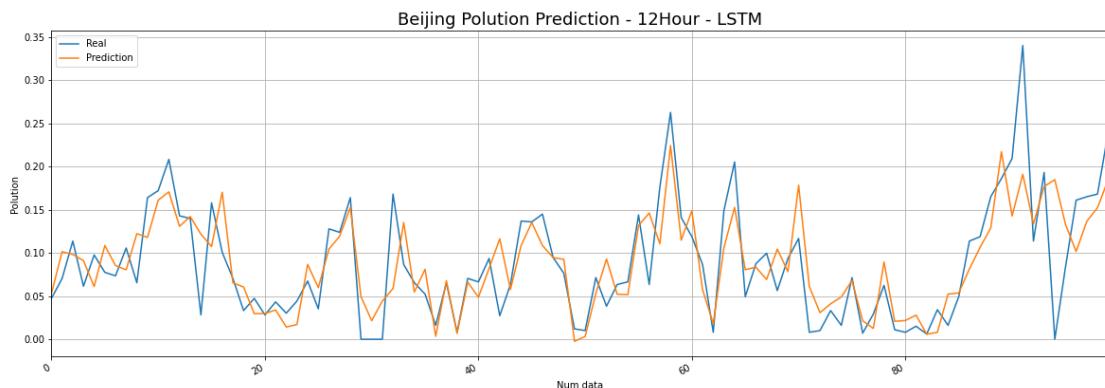
شکل 102: نمودار Loss مربوط به حالت Valid و Test در مدل LSTM و حالت AdaGrad و MSE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالت که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 103: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل LSTM و حالت AdaGrad و MSE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 104: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل LSTM به ازای MSE و AdaGrad برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 1.339279623546948e-05
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 105: مقدار Loss به دست آمده برای مدل LSTM در حالت MSE و AdaGrad برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.053101
1	0.070423	0.101334
2	0.113682	0.098151
3	0.061368	0.090975
4	0.097586	0.060953
5	0.077465	0.108536
6	0.073441	0.085325
7	0.105634	0.080389
8	0.065392	0.122242
9	0.163984	0.117861
10	0.172032	0.161006
11	0.208249	0.170631
12	0.142857	0.130748
13	0.139839	0.142105
14	0.028169	0.121655
15	0.157948	0.107362
16	0.100604	0.170199
17	0.070423	0.065306
18	0.033199	0.060517
19	0.047284	0.029637
20	0.028169	0.029795
21	0.043260	0.033898
22	0.030181	0.014169

شکل 106: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 11 - مدل LSTM در حالت MAE و AdaGrad

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 107: تعریف مدل، Optimizer و تابع Loss برای LSTM

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 0.0002976400457322597 , Valid Loss : => 0.0004169174035390218
***->>>-----<<<-***  
step : 1 Train loss : 0.0002782383703937133 , Valid Loss : => 0.00041061794385313985
***->>>-----<<<-***  
step : 2 Train loss : 0.00026982923485338687 , Valid Loss : => 0.00040322375297546385
***->>>-----<<<-***  
step : 3 Train loss : 0.00026370725234349567 , Valid Loss : => 0.00039589580645163854
***->>>-----<<<-***  
step : 4 Train loss : 0.000258789029593269 , Valid Loss : => 0.00038842249661684035
***->>>-----<<<-***  
step : 5 Train loss : 0.0002544618227829536 , Valid Loss : => 0.0003809774654606978
***->>>-----<<<-***
```

شکل 108: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل LSTM

```
***->>>-----<<<-***  
step : 93 Train loss : 0.00015852038525044918 , Valid Loss : => 0.00020833348172406355
***->>>-----<<<-***  
step : 94 Train loss : 0.0001582728100940585 , Valid Loss : => 0.00020794469552735487
***->>>-----<<<-***  
step : 95 Train loss : 0.00015803043699512878 , Valid Loss : => 0.00020754017060001692
***->>>-----<<<-***  
step : 96 Train loss : 0.00015778444285194078 , Valid Loss : => 0.00020721604861319065
***->>>-----<<<-***  
step : 97 Train loss : 0.00015754763750980296 , Valid Loss : => 0.00020680716882149377
***->>>-----<<<-***  
step : 98 Train loss : 0.0001573084284241001 , Valid Loss : => 0.00020643371529877185
***->>>-----<<<-***  
step : 99 Train loss : 0.00015707765519618988 , Valid Loss : => 0.0002061026884863774
***->>>-----<<<-***
```

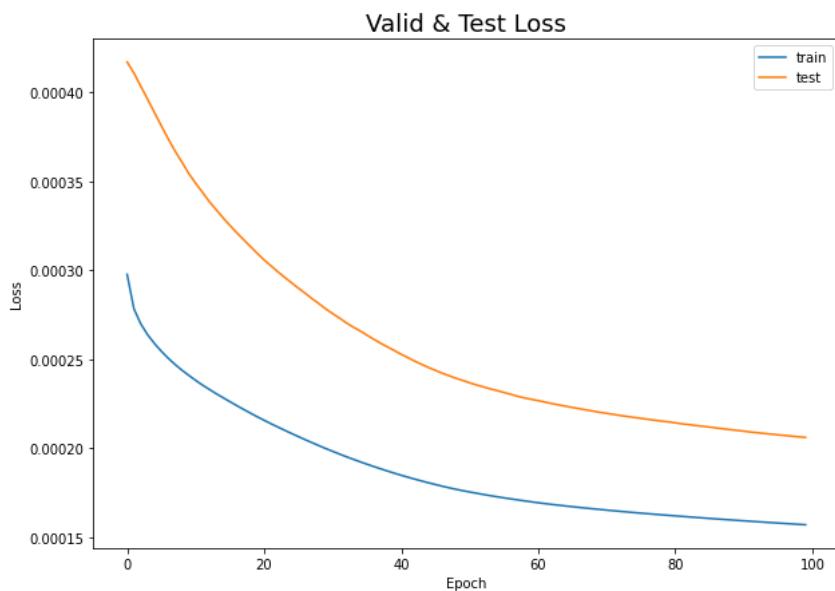
شکل 109: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل LSTM

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 652.3857469558716 Sec.  
*****  
=====
```

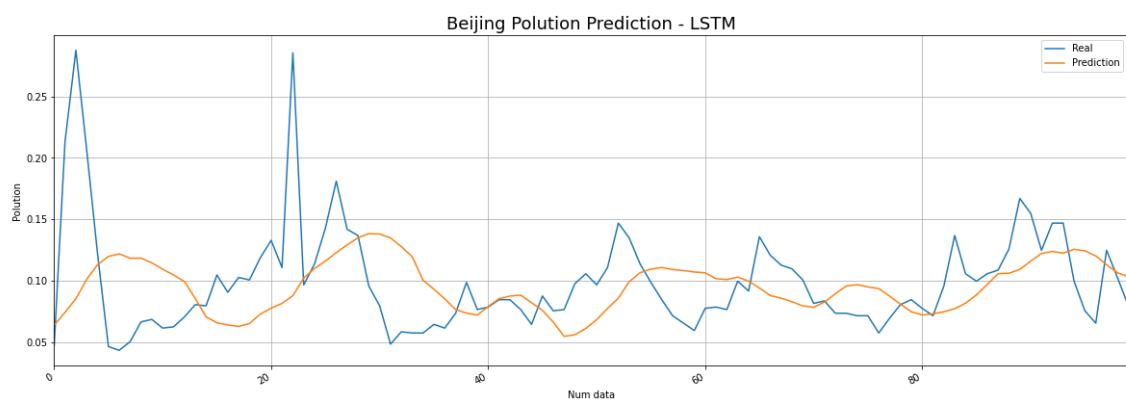
شکل 110: زمان آموزش مدل LSTM در حالت AdaGrad و MAE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



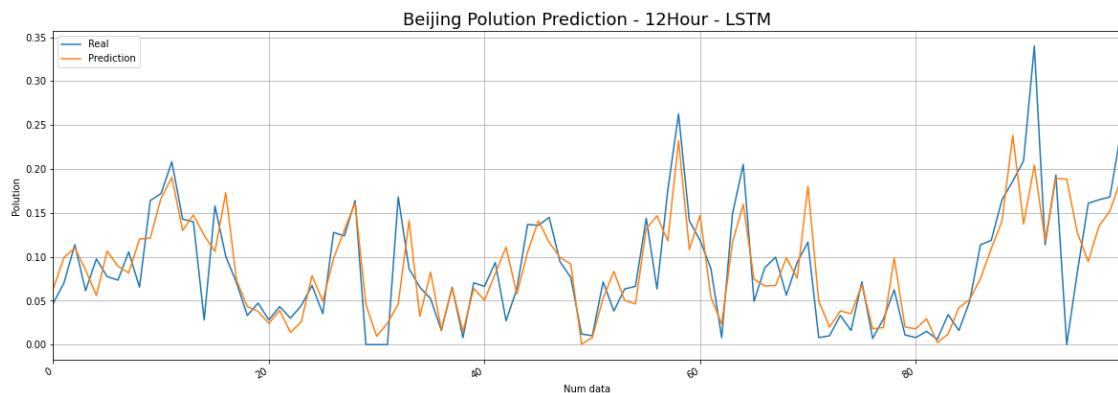
شکل 111: نمودار Loss مربوط به حالت Valid و Test در مدل LSTM و MAE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالت که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 112: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل LSTM و MAE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 113: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل LSTM به ازای AdaGrad و MAE برای داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.00015516493003815414
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 114: مقدار Loss به دست آمده برای مدل LSTM در حالت AdaGrad و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.063773
1	0.070423	0.099358
2	0.113682	0.109989
3	0.061368	0.085122
4	0.097586	0.055951
5	0.077465	0.106411
6	0.073441	0.089331
7	0.105634	0.081633
8	0.065392	0.120171
9	0.163984	0.121318
10	0.172032	0.166717
11	0.208249	0.190333
12	0.142857	0.130042
13	0.139839	0.147625
14	0.028169	0.124423
15	0.157948	0.106028
16	0.100604	0.172940
17	0.070423	0.073407
18	0.033199	0.043278
19	0.047284	0.037853
20	0.028169	0.024278
21	0.043260	0.038980
22	0.030181	0.014066

شکل 115: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 12 - مدل LSTM در حالت MSE و RMSprop

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)

```

شکل 116: تعریف مدل، Loss و تابع Optimizer برای LSTM

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```

step : 0 Train loss : 3.417752895426626e-05 , Valid Loss : => 5.290285140896837e-05
***->>>-----<<<-***  

step : 1 Train loss : 1.952959647945439e-05 , Valid Loss : => 3.77776159044976034e-05
***->>>-----<<<-***  

step : 2 Train loss : 1.5369250470151504e-05 , Valid Loss : => 2.8376678392911952e-05
***->>>-----<<<-***  

step : 3 Train loss : 1.3177460078926136e-05 , Valid Loss : => 2.3271646932698785e-05
***->>>-----<<<-***  

step : 4 Train loss : 1.1643657053355128e-05 , Valid Loss : => 2.0221844237918657e-05
***->>>-----<<<-***  

step : 5 Train loss : 1.0463022819021717e-05 , Valid Loss : => 1.8157828405189016e-05
***->>>-----<<<-***  


```

شکل 117: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل LSTM

```

***->>>-----<<<-***  

step : 93 Train loss : 3.788491471398932e-06 , Valid Loss : => 4.969964444171638e-06
***->>>-----<<<-***  

step : 94 Train loss : 3.783221045159735e-06 , Valid Loss : => 4.973596476096039e-06
***->>>-----<<<-***  

step : 95 Train loss : 3.7779860567146293e-06 , Valid Loss : => 4.977057173770542e-06
***->>>-----<<<-***  

step : 96 Train loss : 3.7727901702358697e-06 , Valid Loss : => 4.980286602706959e-06
***->>>-----<<<-***  

step : 97 Train loss : 3.7676362320780755e-06 , Valid Loss : => 4.9832330745023984e-06
***->>>-----<<<-***  

step : 98 Train loss : 3.76252717008659e-06 , Valid Loss : => 4.985863837646321e-06
***->>>-----<<<-***  

step : 99 Train loss : 3.7574629900821794e-06 , Valid Loss : => 4.988159770922114e-06
***->>>-----<<<-***  


```

شکل 118: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل LSTM

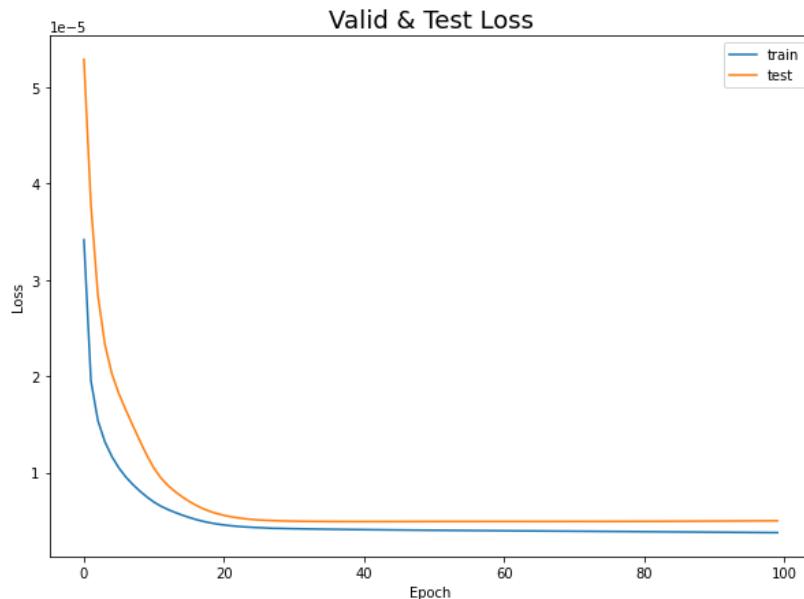
سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 708.1412284374237 Sec.
*****
=====
```

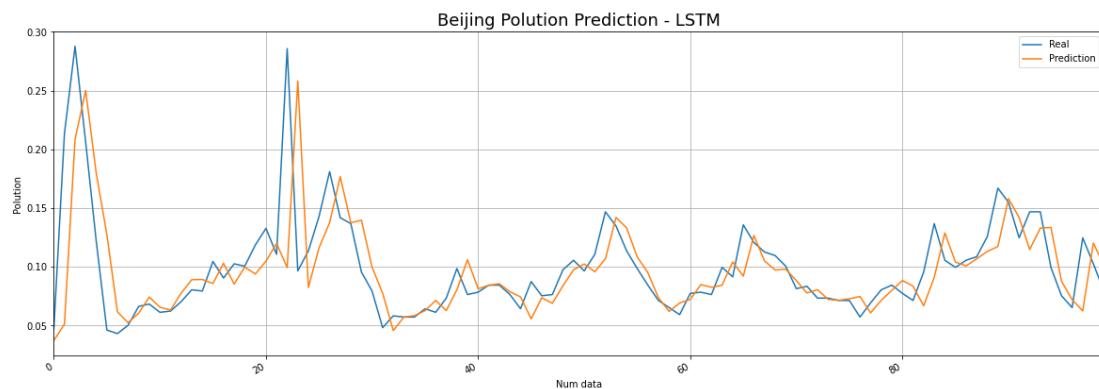
شکل 119: زمان آموزش مدل LSTM در حالت MSE و RMSprop در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



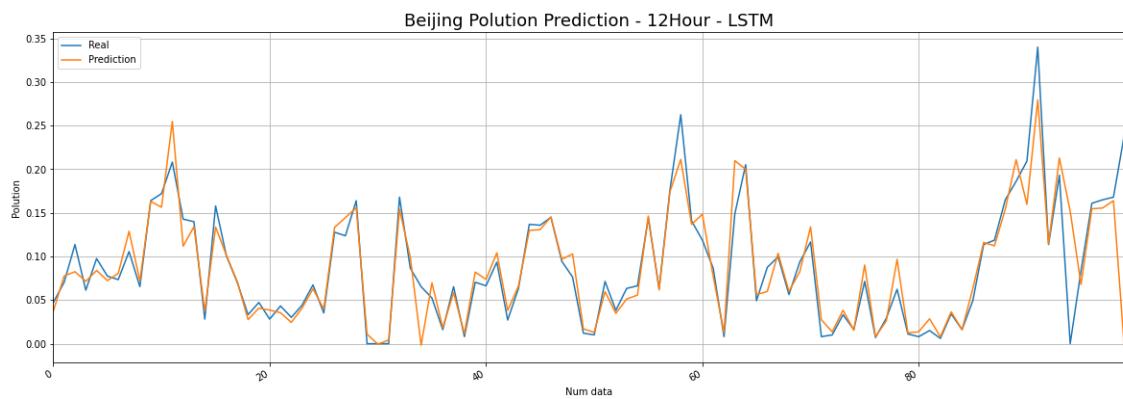
شکل 120: نمودار Loss مربوط به حالت Valid و Test در مدل LSTM و حالت RMSprop و MSE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 121: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل LSTM و حالت RMSprop و MSE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 122: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل LSTM به ازای RMSprop و MSE برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.013743474691486e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 123: مقدار Loss به دست آمده برای مدل LSTM در حالت RMSprop و MSE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.037327
1	0.070423	0.077641
2	0.113682	0.082530
3	0.061368	0.071525
4	0.097586	0.083930
5	0.077465	0.072360
6	0.073441	0.080604
7	0.105634	0.128899
8	0.065392	0.072068
9	0.163984	0.163129
10	0.172032	0.156432
11	0.208249	0.254842
12	0.142857	0.111793
13	0.139839	0.134169
14	0.028169	0.037311
15	0.157948	0.133701
16	0.100604	0.101418
17	0.070423	0.071468
18	0.033199	0.027585
19	0.047284	0.040735
20	0.028169	0.038562
21	0.043260	0.035450
22	0.030181	0.024423

شکل 124: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 13 - مدل LSTM در حالت MAE و RMSprop

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 125: تعریف مدل، Optimizer و تابع Loss برای LSTM

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 0.00028904371956984203 , Valid Loss : => 0.00032653038452068965
***->>>-----<<<-***  
step : 1 Train loss : 0.00021705336446563403 , Valid Loss : => 0.0002565800646940867
***->>>-----<<<-***  
step : 2 Train loss : 0.00017943174354732036 , Valid Loss : => 0.00021421559527516366
***->>>-----<<<-***  
step : 3 Train loss : 0.00015840888222058615 , Valid Loss : => 0.00018751587346196174
***->>>-----<<<-***  
step : 4 Train loss : 0.0001462653884043296 , Valid Loss : => 0.00017552017358442147
***->>>-----<<<-***  
step : 5 Train loss : 0.0001370898090923826 , Valid Loss : => 0.0001632505562156439
***->>>-----<<<-***
```

شکل 126: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل LSTM

```
***->>>-----<<<-***  
step : 93 Train loss : 6.782538093005617e-05 , Valid Loss : => 7.548602763563395e-05
***->>>-----<<<-***  
step : 94 Train loss : 6.784266621495287e-05 , Valid Loss : => 7.520698538670937e-05
***->>>-----<<<-***  
step : 95 Train loss : 6.794797924036781e-05 , Valid Loss : => 7.51367195819815e-05
***->>>-----<<<-***  
step : 96 Train loss : 6.791796584924063e-05 , Valid Loss : => 7.525161902109782e-05
***->>>-----<<<-***  
step : 97 Train loss : 6.764911180362105e-05 , Valid Loss : => 7.505783500770728e-05
***->>>-----<<<-***  
step : 98 Train loss : 6.792673065016667e-05 , Valid Loss : => 7.55347056935231e-05
***->>>-----<<<-***  
step : 99 Train loss : 6.775045605997244e-05 , Valid Loss : => 7.555673861255249e-05
***->>>-----<<<-***
```

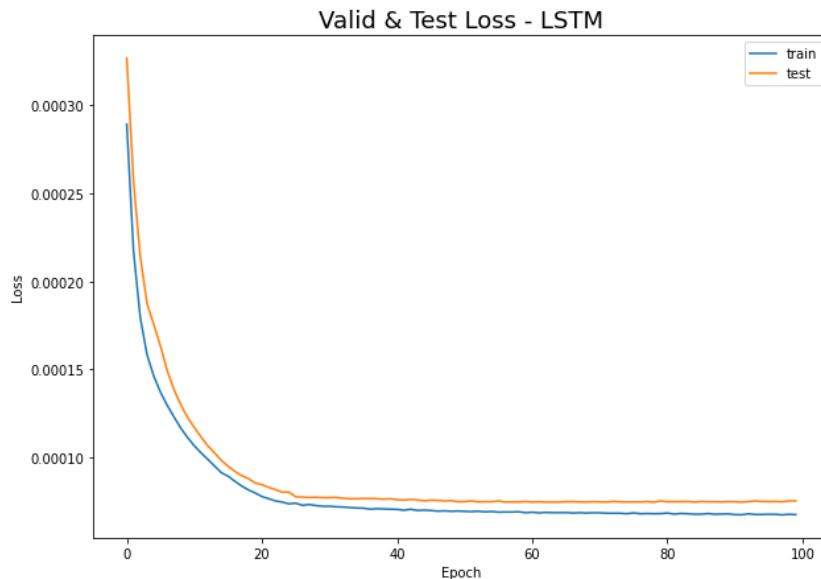
شکل 127: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل LSTM

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 711.8039515018463 Sec.
*****
=====
```

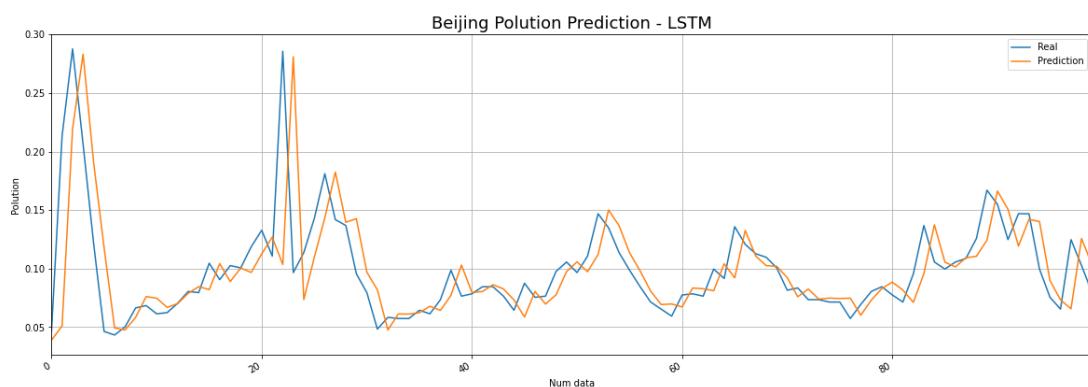
شکل 128: زمان آموزش مدل LSTM در حالت MAE و RMSprop در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



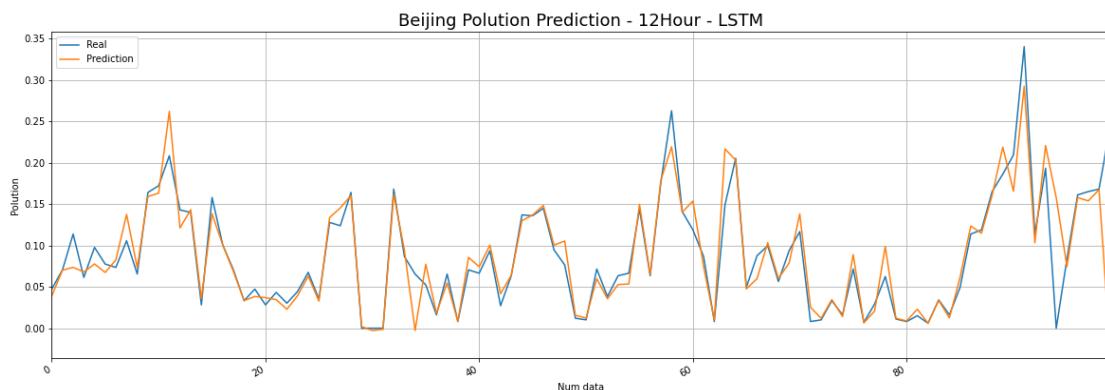
شکل 129: نمودار Loss مربوط به حالت Valid و RMSprop و حالت LSTM در مدل

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالت که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 130: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل LSTM و RMSprop و MAE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 131: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل LSTM به ازای RMSprop و MAE برای داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 6.509193141634266e-05
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 132: مقدار Loss به دست آمده برای مدل LSTM در حالت RMSprop و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.047284	0.038962
1	0.070423	0.070082
2	0.113682	0.073509
3	0.061368	0.067823
4	0.097586	0.077599
5	0.077465	0.067273
6	0.073441	0.082511
7	0.105634	0.137408
8	0.065392	0.073288
9	0.163984	0.158975
10	0.172032	0.163155
11	0.208249	0.261844
12	0.142857	0.121134
13	0.139839	0.143100
14	0.028169	0.035826
15	0.157948	0.138172
16	0.100604	0.100366
17	0.070423	0.068508
18	0.033199	0.033348
19	0.047284	0.038276
20	0.028169	0.036924
21	0.043260	0.034543

شکل 133: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 14 - مدل GRU در حالت MAE و ADAM

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```

1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 134: تعریف مدل، Loss و Optimizer برای GRU

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```

step : 0 Train loss : 0.00028951335189243157 , Valid Loss : => 0.000383052629729112
***->>-----<<<-***
step : 1 Train loss : 0.0002298495940864086 , Valid Loss : => 0.00042195195083816845
***->>-----<<<-***
step : 2 Train loss : 0.00022432622474928698 , Valid Loss : => 0.0004745387174189091
***->>-----<<<-***
step : 3 Train loss : 0.00023519927250842252 , Valid Loss : => 0.00045875442028045656
***->>-----<<<-***
step : 4 Train loss : 0.00021608626234034697 , Valid Loss : => 0.000497861764083306
***->>-----<<<-***
step : 5 Train loss : 0.00020743850705524285 , Valid Loss : => 0.00044281706462303796
***->>-----<<<-***

```

شکل 135: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل GRU

```

***->>-----<<<-***
step : 93 Train loss : 7.519913564125697e-05 , Valid Loss : => 0.00010556714236736298
***->>-----<<<-***
step : 94 Train loss : 7.548689305161436e-05 , Valid Loss : => 0.00010363160322109858
***->>-----<<<-***
step : 95 Train loss : 7.443634532392025e-05 , Valid Loss : => 0.00010262494379033645
***->>-----<<<-***
step : 96 Train loss : 7.387245831390222e-05 , Valid Loss : => 9.301021105299393e-05
***->>-----<<<-***
step : 97 Train loss : 7.262671043475469e-05 , Valid Loss : => 9.938695778449377e-05
***->>-----<<<-***
step : 98 Train loss : 7.407222567126155e-05 , Valid Loss : => 0.00010334907534221808
***->>-----<<<-***
step : 99 Train loss : 7.476664182419578e-05 , Valid Loss : => 0.00010474505492796501
***->>-----<<<-***

```

شکل 136: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل GRU

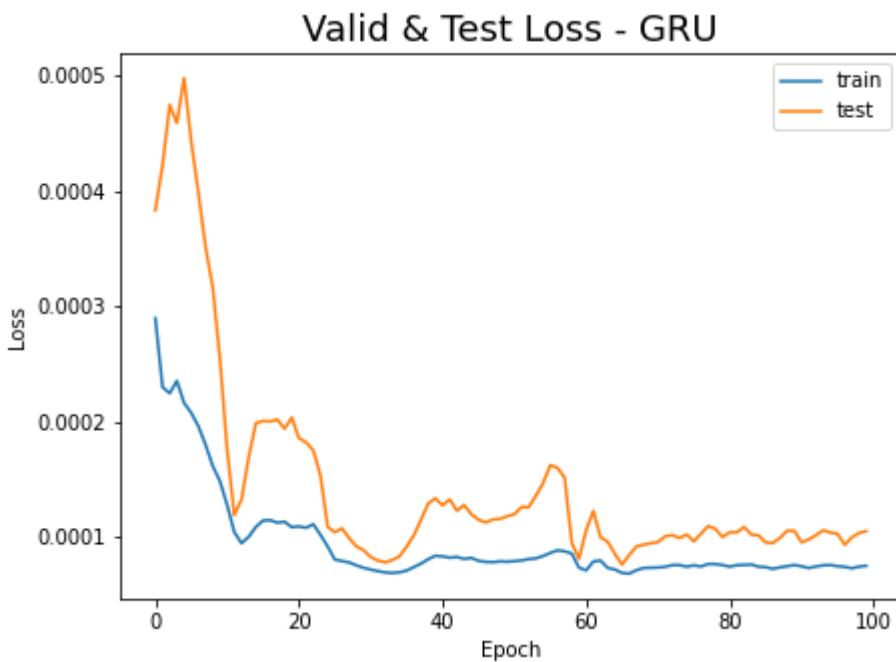
سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 554.418694972992 Sec.
*****
=====
```

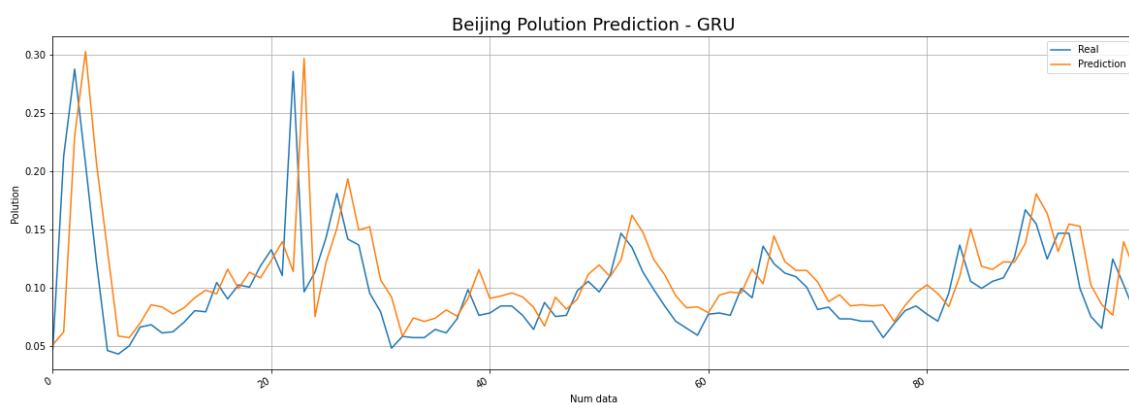
شکل 137: زمان آموزش مدل GRU در حالت ADAM و MAE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Train و Valid را قرار می‌دهم:



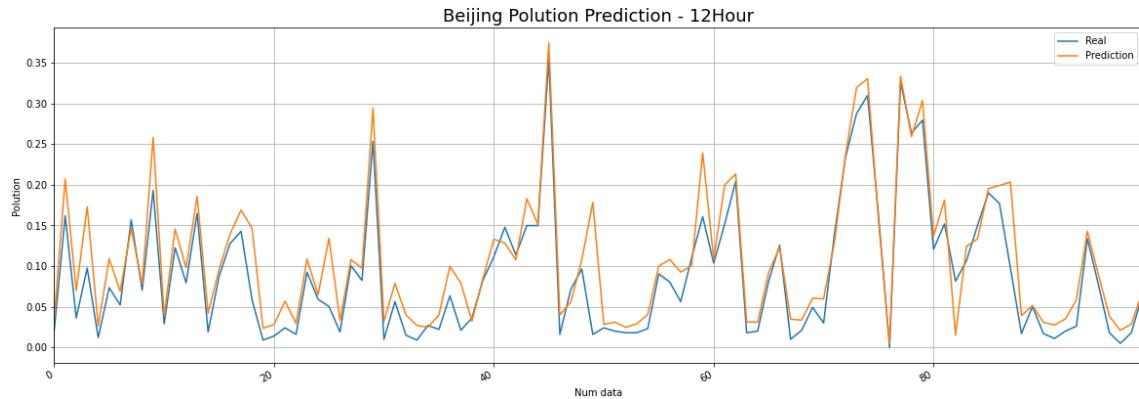
شکل 138: نمودار Loss مربوط به حالت Valid و حالت Test در مدل GRU و مدل ADAM

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 139: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 عدد تست در مدل GRU و مدل ADAM

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 140: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای ADAM و MAE برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 9.616751658419768e-05
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 141: مقدار Loss به دست آمده برای مدل GRU در حالت Adam و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.021127	0.036532
1	0.161972	0.206988
2	0.036217	0.069786
3	0.097586	0.172983
4	0.012072	0.026392
5	0.073441	0.109023
6	0.052314	0.069102
7	0.156942	0.146558
8	0.070423	0.077838
9	0.193159	0.258232
10	0.029175	0.038683
11	0.122736	0.145760
12	0.079477	0.097994
13	0.164990	0.185846
14	0.019115	0.042099
15	0.087525	0.096046
16	0.127767	0.139561
17	0.142857	0.168937
18	0.058350	0.146619
19	0.009054	0.023666
20	0.014085	0.028215
21	0.024145	0.056801
22	0.016097	0.029622

شکل 142: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 15 - مدل GRU و AdaGrad در حالت MSE

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 143: تعریف مدل، Optimizer و تابع Loss برای GRU

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 3.4064269981657464e-05 , Valid Loss : => 5.600056890398264e-05
***->>>-----<<<-***  
step : 1 Train loss : 2.8638637368567286e-05 , Valid Loss : => 5.31311157780389e-05
***->>>-----<<<-***  
step : 2 Train loss : 2.6459376335454484e-05 , Valid Loss : => 5.017225653864443e-05
***->>>-----<<<-***  
step : 3 Train loss : 2.4927856990446645e-05 , Valid Loss : => 4.729807586409152e-05
***->>>-----<<<-***  
step : 4 Train loss : 2.3709232332961012e-05 , Valid Loss : => 4.4662268133834004e-05
***->>>-----<<<-***  
step : 5 Train loss : 2.268789946101606e-05 , Valid Loss : => 4.230715326654414e-05
***->>>-----<<<-***
```

شکل 144: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 93 Train loss : 1.0636509799708922e-05 , Valid Loss : => 1.7629471801531813e-05
***->>>-----<<<-***  
step : 94 Train loss : 1.0606696122946838e-05 , Valid Loss : => 1.758217237268885e-05
***->>>-----<<<-***  
step : 95 Train loss : 1.0577226878376677e-05 , Valid Loss : => 1.753541388704131e-05
***->>>-----<<<-***  
step : 96 Train loss : 1.0548094619298355e-05 , Valid Loss : => 1.7489181365817785e-05
***->>>-----<<<-***  
step : 97 Train loss : 1.0519290135319655e-05 , Valid Loss : => 1.7443463555537165e-05
***->>>-----<<<-***  
step : 98 Train loss : 1.0490806740320598e-05 , Valid Loss : => 1.7398242567044994e-05
***->>>-----<<<-***  
step : 99 Train loss : 1.046263679357556e-05 , Valid Loss : => 1.735351230793943e-05
***->>>-----<<<-***
```

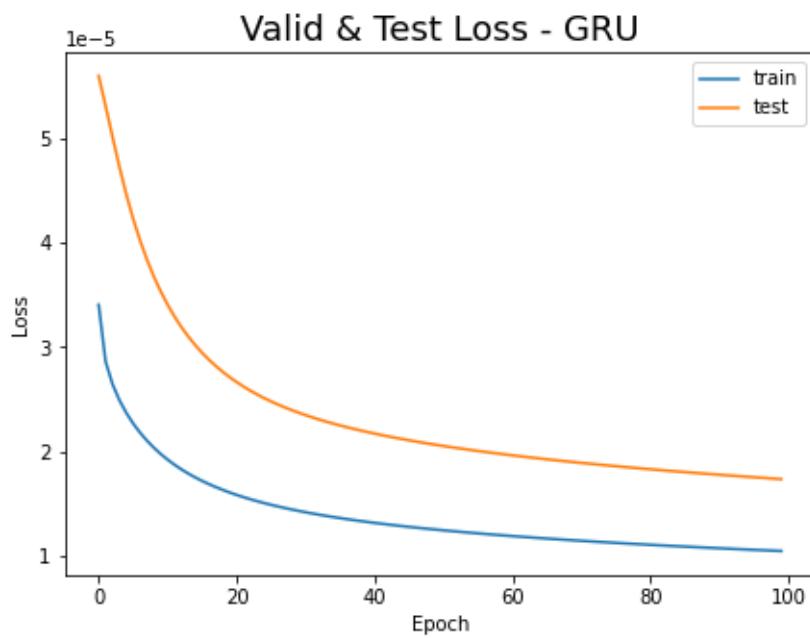
شکل 145: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل GRU

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 451.6257619857788 Sec.
*****  
=====
```

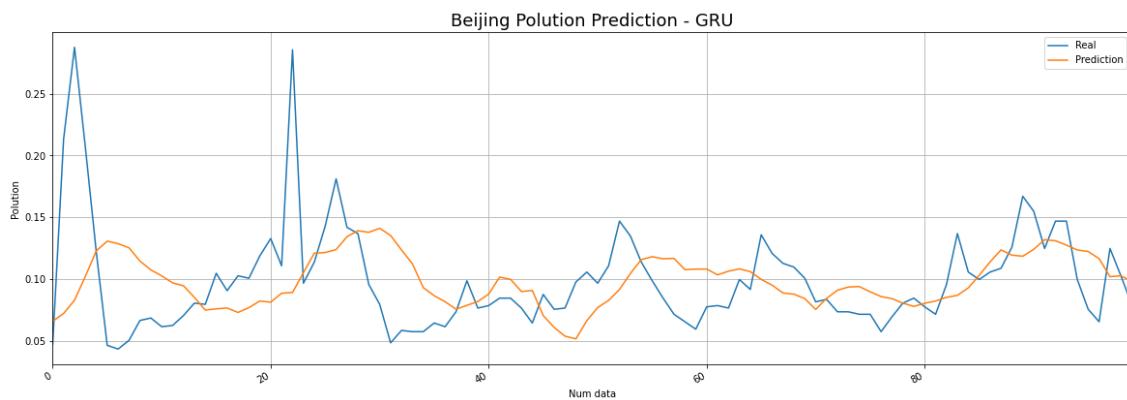
شکل 146: زمان آموزش مدل GRU و AdaGrad در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



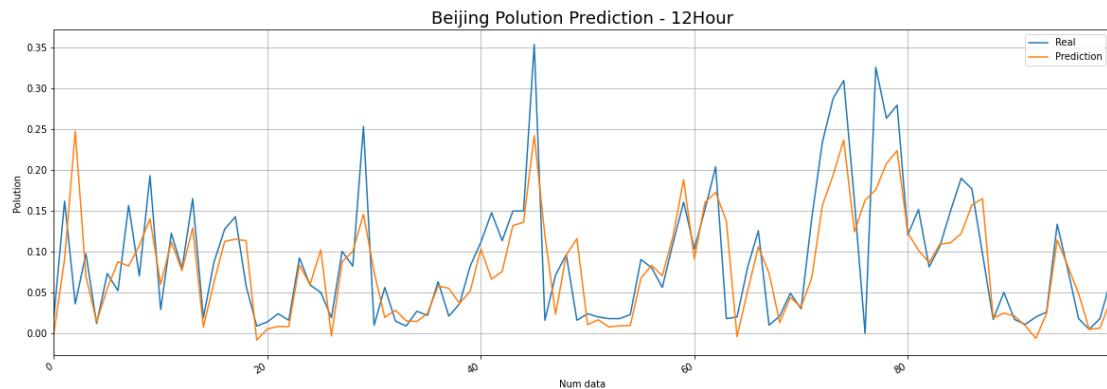
شکل ۱۴۷: نمودار Loss مربوط به حالت Valid و Test در مدل GRU و حالت AdaGrad

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل ۱۴۸: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل GRU و حالت AdaGrad

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 149: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای MSE برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 1.0772915799558784e-05
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 150: مقدار Loss به دست آمده برای مدل GRU در حالت MSE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.021127	0.000751
1	0.161972	0.091089
2	0.036217	0.247912
3	0.097586	0.069733
4	0.012072	0.014109
5	0.073441	0.055002
6	0.052314	0.087765
7	0.156942	0.082511
8	0.070423	0.107803
9	0.193159	0.140286
10	0.029175	0.060018
11	0.122736	0.112154
12	0.079477	0.076875
13	0.164990	0.129457
14	0.019115	0.007819
15	0.087525	0.063026
16	0.127767	0.112807
17	0.142857	0.115290
18	0.058350	0.113583
19	0.009054	-0.008251
20	0.014085	0.005463
21	0.024145	0.008633

شکل 151: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 16 - مدل GRU و MAE در حالت

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 152: تعریف مدل، Optimizer و تابع Loss برای GRU

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 0.0002775702913602193 , Valid Loss : => 0.0003770597167313099
***->>>-----<<<-***  
step : 1 Train loss : 0.0002500108043352763 , Valid Loss : => 0.0003704784922301769
***->>>-----<<<-***  
step : 2 Train loss : 0.00024043115749955177 , Valid Loss : => 0.0003627117263774077
***->>>-----<<<-***  
step : 3 Train loss : 0.00023397784990568957 , Valid Loss : => 0.0003524515752991041
***->>>-----<<<-***  
step : 4 Train loss : 0.00022857070614894231 , Valid Loss : => 0.00034225283935666087
***->>>-----<<<-***  
step : 5 Train loss : 0.00022384231872856617 , Valid Loss : => 0.0003319394091765086
***->>>-----<<<-***
```

شکل 153: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل GRU

```
=====  
***->>>-----<<<-***  
step : 93 Train loss : 0.00013420120763281982 , Valid Loss : => 0.00017513930425047875
***->>>-----<<<-***  
step : 94 Train loss : 0.00013393127620220185 , Valid Loss : => 0.00017472987435758114
***->>>-----<<<-***  
step : 95 Train loss : 0.00013366047342618307 , Valid Loss : => 0.00017437825041512648
***->>>-----<<<-***  
step : 96 Train loss : 0.0001334061769147714 , Valid Loss : => 0.0001740009505301714
***->>>-----<<<-***  
step : 97 Train loss : 0.00013315060113867124 , Valid Loss : => 0.00017365406453609468
***->>>-----<<<-***  
step : 98 Train loss : 0.00013289145343005658 , Valid Loss : => 0.0001732728723436594
***->>>-----<<<-***  
step : 99 Train loss : 0.00013262904323637484 , Valid Loss : => 0.00017289977086087067
***->>>-----<<<-***
```

شکل 154: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل GRU

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 449.21241521835327 Sec.
*****  
=====
```

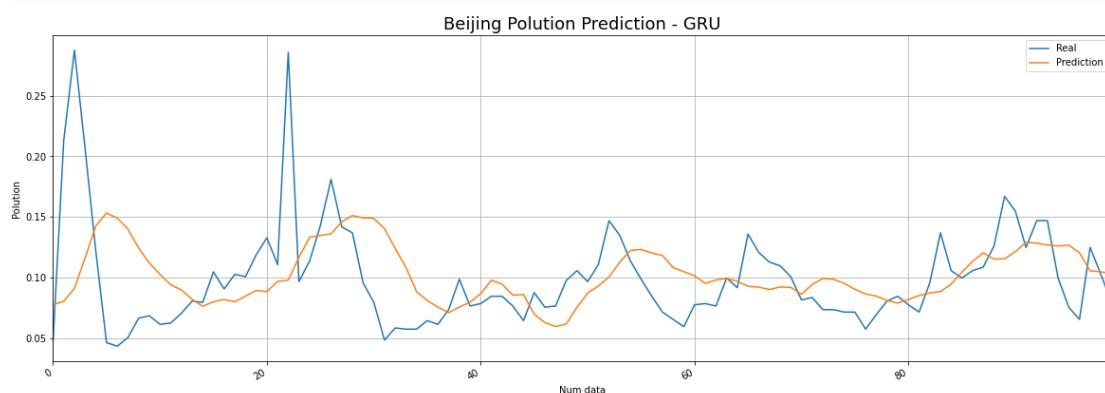
شکل 155: زمان آموزش مدل GRU در حالت AdaGrad و MAE در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



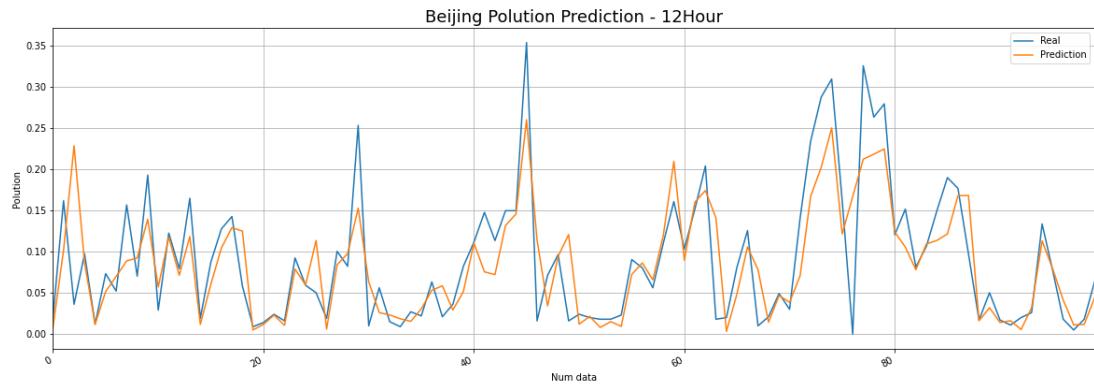
شکل 156: نمودار Loss مربوط به حالت Valid و Test در مدل GRU و حالت MAE و AdaGrad

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 157: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل GRU و حالت MAE و AdaGrad

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 158: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای $MAE = 100$ برای داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0001314339777454734
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 159: مقدار Loss به دست آمده برای مدل GRU در حالت $MAE = 100$ برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.021127	0.008690
1	0.161972	0.101859
2	0.036217	0.228842
3	0.097586	0.086579
4	0.012072	0.011954
5	0.073441	0.051708
6	0.052314	0.070037
7	0.156942	0.089047
8	0.070423	0.092497
9	0.193159	0.139526
10	0.029175	0.057380
11	0.122736	0.117815
12	0.079477	0.071524
13	0.164990	0.118604
14	0.019115	0.011844
15	0.087525	0.060561
16	0.127767	0.104670
17	0.142857	0.129320
18	0.058350	0.125198
19	0.009054	0.005055
20	0.014085	0.011845
21	0.024145	0.023099

شکل 160: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 17 - مدل GRU در حالت MSE و RMSprop

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 161: تعریف مدل، Optimizer و تابع Loss برای GRU

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 4.611558942512299e-05 , Valid Loss : => 4.234819269428651e-05
***->>>-----<<<-***  
step : 1 Train loss : 1.8628222479795415e-05 , Valid Loss : => 2.876194550966223e-05
***->>>-----<<<-***  
step : 2 Train loss : 1.4241383865009994e-05 , Valid Loss : => 2.0167886783989768e-05
***->>>-----<<<-***  
step : 3 Train loss : 1.2178376002702861e-05 , Valid Loss : => 1.6702203584524493e-05
***->>>-----<<<-***  
step : 4 Train loss : 1.0888016367486368e-05 , Valid Loss : => 1.5119585868281623e-05
***->>>-----<<<-***  
step : 5 Train loss : 9.82540895153458e-06 , Valid Loss : => 1.3719875365495681e-05
***->>>-----<<<-***
```

شکل 162: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 93 Train loss : 3.840923798270524e-06 , Valid Loss : => 4.924660461256281e-06
***->>>-----<<<-***  
step : 94 Train loss : 3.83665627547695e-06 , Valid Loss : => 4.927705759958674e-06
***->>>-----<<<-***  
step : 95 Train loss : 3.832333922036924e-06 , Valid Loss : => 4.930604404459397e-06
***->>>-----<<<-***  
step : 96 Train loss : 3.8279481280672674e-06 , Valid Loss : => 4.9333331213953594e-06
***->>>-----<<<-***  
step : 97 Train loss : 3.823496134524854e-06 , Valid Loss : => 4.935872023149083e-06
***->>>-----<<<-***  
step : 98 Train loss : 3.8189789251191545e-06 , Valid Loss : => 4.938208158516015e-06
***->>>-----<<<-***  
step : 99 Train loss : 3.814400609311027e-06 , Valid Loss : => 4.9403456990451865e-06
***->>>-----<<<-***
```

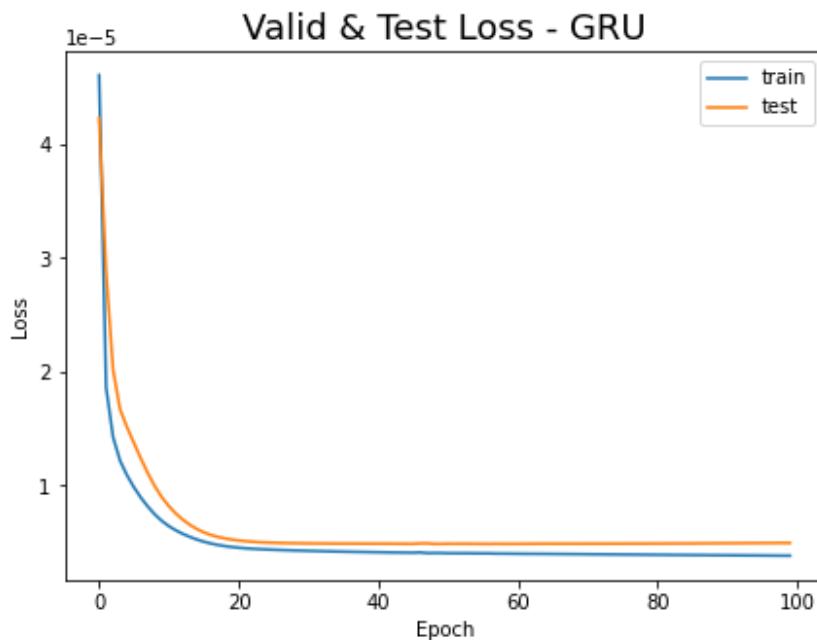
شکل 163: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل GRU

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 547.2024540901184 Sec.  
*****  
=====
```

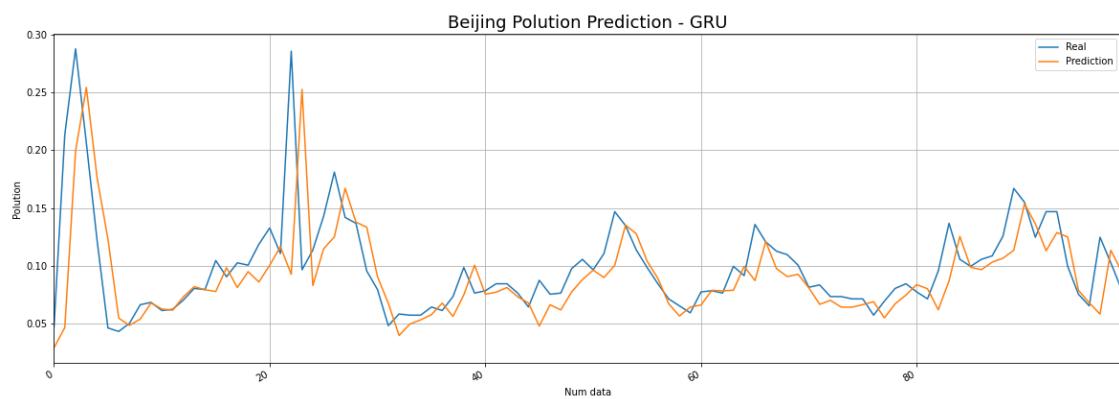
شکل 164: زمان آموزش مدل GRU در حالت MSE و RMSprop در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



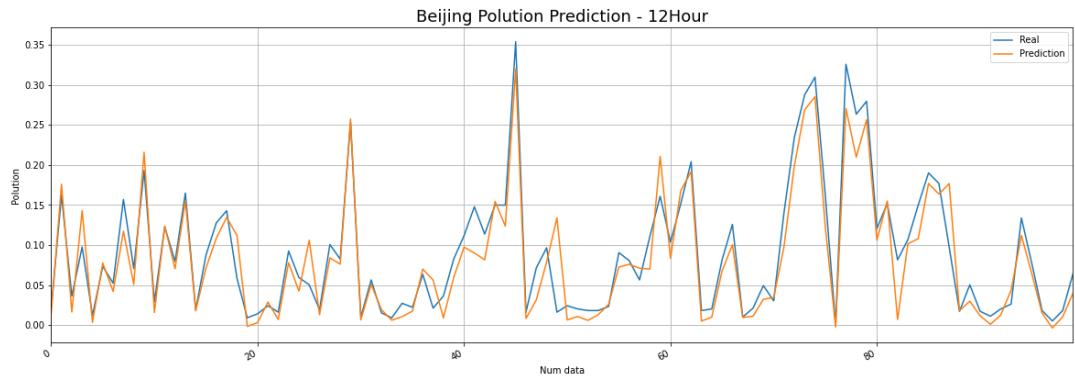
شکل ۱۶۵: نمودار Loss مربوط به حالت Valid و Test در مدل GRU و حالت RMSprop و MSE

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل ۱۶۶: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل GRU و حالت RMSprop و MSE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 167: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای MSE و RMSprop برای داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.209985510290911e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 168: مقدار Loss به دست آمده برای مدل GRU در حالت MSE و RMSprop برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.021127	0.014875
1	0.161972	0.175964
2	0.036217	0.016357
3	0.097586	0.142896
4	0.012072	0.003287
5	0.073441	0.077432
6	0.052314	0.041931
7	0.156942	0.117272
8	0.070423	0.050838
9	0.193159	0.216043
10	0.029175	0.015627
11	0.122736	0.123635
12	0.079477	0.070257
13	0.164990	0.155267
14	0.019115	0.017798
15	0.087525	0.071300
16	0.127767	0.108683
17	0.142857	0.134512
18	0.058350	0.112006
19	0.009054	-0.001666
20	0.014085	0.003046
21	0.024145	0.028800

شکل 169: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

► 18 - مدل GRU و RMSprop در حالت MAE

در این حالت در ابتدا تعریفتابع Loss و Optimizer مربوطه را قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 170: تعریف مدل، Optimizer و تابع Loss برای GRU

سپس در ادامه به بررسی Loss در 5 ایپاک اول و آخر آموزش شبکه می‌پردازیم:

```
step : 0 Train loss : 0.00032613365662594637 , Valid Loss : => 0.0002813616966207822
***->>>-----<<<-***  
step : 1 Train loss : 0.00020632097162306308 , Valid Loss : => 0.0002171630406131347
***->>>-----<<<-***  
step : 2 Train loss : 0.00016832587520281475 , Valid Loss : => 0.00018003576807677747
***->>>-----<<<-***  
step : 3 Train loss : 0.000151033270607392 , Valid Loss : => 0.00015873383979002635
***->>>-----<<<-***  
step : 4 Train loss : 0.0001392833258335789 , Valid Loss : => 0.00014299801674981912
***->>>-----<<<-***  
step : 5 Train loss : 0.0001279532788321376 , Valid Loss : => 0.00013045621166626612
***->>>-----<<<-***
```

شکل 171: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 93 Train loss : 6.858613062649966e-05 , Valid Loss : => 7.746787803868452e-05
***->>>-----<<<-***  
step : 94 Train loss : 6.839773723234733e-05 , Valid Loss : => 7.606785806516806e-05
***->>>-----<<<-***  
step : 95 Train loss : 6.896719078843792e-05 , Valid Loss : => 0.00011132127232849598
***->>>-----<<<-***  
step : 96 Train loss : 6.91125960710148e-05 , Valid Loss : => 7.638237935801347e-05
***->>>-----<<<-***  
step : 97 Train loss : 6.83754902643462e-05 , Valid Loss : => 7.632328756153583e-05
***->>>-----<<<-***  
step : 98 Train loss : 6.81923327036202e-05 , Valid Loss : => 7.825666014105081e-05
***->>>-----<<<-***  
step : 99 Train loss : 6.826096720372637e-05 , Valid Loss : => 7.64681069801251e-05
***->>>-----<<<-***
```

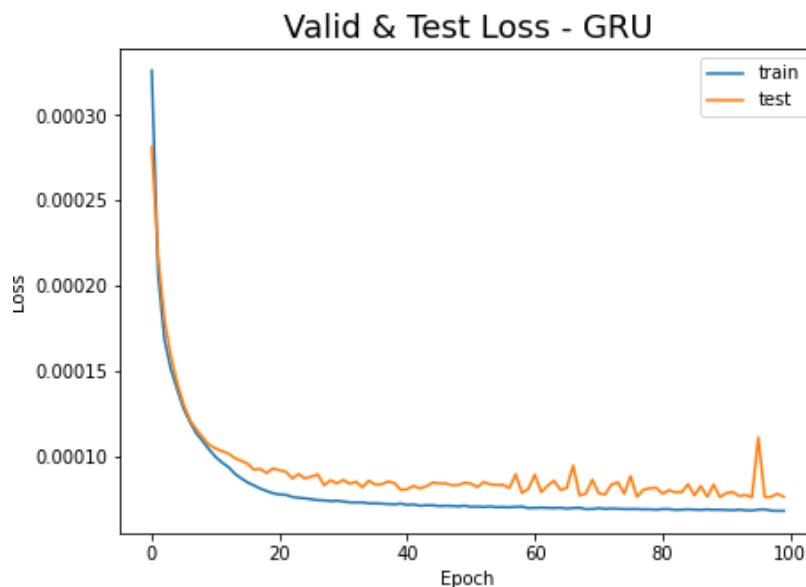
شکل 172: مقادیر Loss داده‌های آموزش و valid برای 5 ایپاک آخر مدل GRU

سپس در ادامه نیز می‌توانید زمان آموزش مربوط به این مدل را مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 608.3001618385315 Sec.  
*****  
=====
```

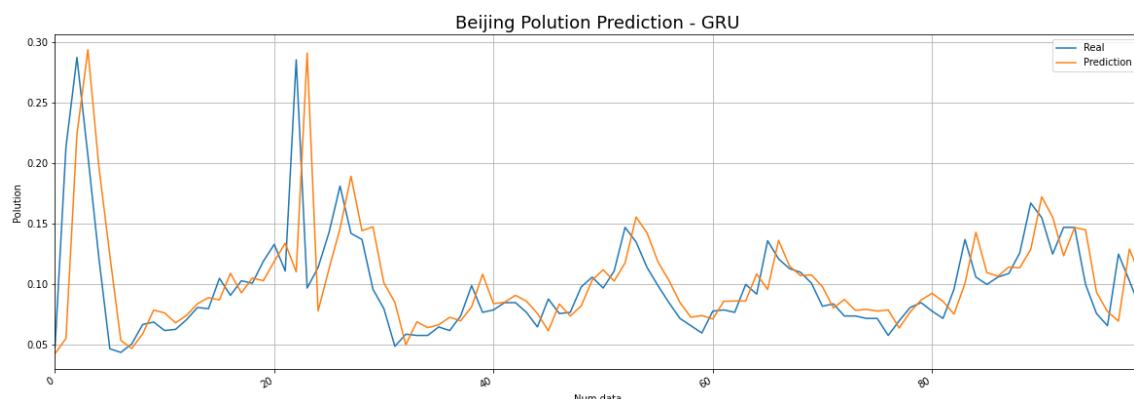
شکل 173: زمان آموزش مدل GRU در حالت MAE و RMSprop در 100 ایپاک

سپس در ادامه نیز نمودار مربوط به Loss در حالت Valid و Train را قرار می‌دهم:



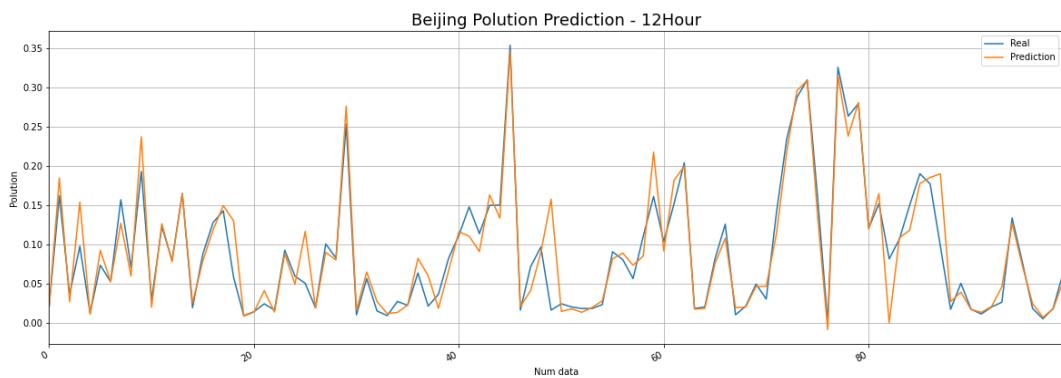
شکل 174: نمودار Loss مربوط به حالت Valid و Test در مدل GRU و حالت MAE و RMSprop

سپس در ادامه نمودار مقدار واقعی و پیش‌بینی شده را برای حالتی که تمامی ساعت‌ها را پیش‌بینی می‌کنیم را می‌توانید در ادامه برای 100 عدد از داده‌های تست مشاهده نمایید:



شکل 175: نمودار مقادیر واقعی و پیش‌بینی شده برای تمامی ساعت‌های 100 داده تست در مدل GRU و حالت MSE

سپس در ادامه این نمودار را برای حالتی که فقط داده‌ها را در دو ساعت 12 و 24 می‌خواهیم که پیش‌بینی کنیم را مشاهده می‌کنید:



شکل 176: نمودار مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24 در مدل GRU به ازای RMSprop و MAE برای 100 داده تست

سپس در ادامه نیز می‌توانید که مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 6.59434002203246e-05
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 177: مقدار Loss به دست آمده برای مدل GRU در حالت RMSprop و MAE برای داده‌های تست

همچنین در ادامه نیز می‌توانید تعدادی از پیش‌بینی‌های انجام شده برای حالت داده‌های ساعت‌های 12 و 24 را مشاهده نمایید و فایل کامل آن را نیز در کنار فایل‌های پروژه قرار داده‌ام:

	Real Values	Predicted Values
0	0.021127	0.020871
1	0.161972	0.184818
2	0.036217	0.026743
3	0.097586	0.153926
4	0.012072	0.010778
5	0.073441	0.092473
6	0.052314	0.052422
7	0.156942	0.126931
8	0.070423	0.059456
9	0.193159	0.237378
10	0.029175	0.019801
11	0.122736	0.126591
12	0.079477	0.077860
13	0.164990	0.164572
14	0.019115	0.025225
15	0.087525	0.078656
16	0.127767	0.118372
17	0.142857	0.149442
18	0.058350	0.130240
19	0.009054	0.008475
20	0.014085	0.013549
21	0.024145	0.041054

شکل 178: مقادیر واقعی و پیش‌بینی شده برای ساعت‌های 12 و 24

نتیجه‌گیری: با مقایسه تمامی موارد بالا به اضافه موارد قسمت A و B مشخص می‌شود که بیشترین زمان اجرا مربوط به شبکه LSTM است به ازای تمامی Optimizer‌ها و توابع Loss

✓ همچنین بهترین دقت را نیز شبکه LSTM ارائه می‌دهد و این شبکه در مقایسه و GRU و RNN بهترین دقت را دارد و موفق شده است که نمودار مقدار پیش‌بینی و واقعی بهتری را نیز به ما بدهد.

✓ همچنین پس از LSTM نیز GRU عملکرد بهتری دارد و در آخر نیز RNN.

✓ در زمینه Optimizer مورد استفاده بدترین عملکرد را AdaGrad دارد و دو حالت RMSprop و Adam بسته به شرایط عملکردهای خوبی داشتند هر دو فقط در حالت معمولًا ما نمودار Loss بهتری را به دست می‌آوردیم.

❖ قسمت D :

در این قسمت از ما خواسته شده است که تحلیل‌ها را به صورت ماهانه و هفتگی انجام دهیم. به این صورت که در حالت هفتگی داده‌های یک ساعت مشخص در 6 روز را برای پیش‌بینی روز 7 آم استفاده کنیم.

و همچنین در حالت ماهانه نیز به این صورت عمل می‌کنیم که یک ساعت و یک روز را به صورت Random انتخاب می‌کنیم و سپس این روز را در طول 3 هفته بررسی می‌کنیم و آن را مبنایی برای پیش‌بینی آن روز در هفته چهارم در نظر می‌گیریم.

مهم‌ترین تغییری که این سوال نسبت به قسمت‌های قبل دارد همین قسمت انتخاب داده‌ها است که من هر دو حالت را در ادامه قرار می‌دهم:

```

6 def select_week(sequences, n_samples=250):
7     X, y = list(), list()
8     rand_hour = randint(0, 24)
9     for i in range(0, n_samples):
10         start_ix = rand_hour + 168 * i # 168 : Week hours!
11         idxs = []
12         for j in range(0, 7):
13             if j <=5:
14                 idx = start_ix + (j * 24) # Add different days in week
15                 idxs.append(idx)
16             if j == 6: # Target
17                 idy = start_ix + (j * 24)
18                 seq_x = sequences[idxs, :]
19                 seq_y = sequences[idy, 0]
20                 y.append(seq_y)
21                 X.append(seq_x)
22
23     return X, y

```

شکل 179: تابع ساخت دیتاست برای حالت هفتگی

```

6 def select_month(sequences, n_samples=250):
7     X, y = list(), list()
8     rand_hour = randint(0, 24)
9     rand_day = randint(0, 7) ←
10    for i in range(0, n_samples):
11        start_ix = rand_hour + rand_day*24 + 672 * i # 168 : Week hours!
12        idxs = []
13        for j in range(0, 4):
14            if j <=2:
15                idx = start_ix + (j * 168) # Add different weeks
16                idxs.append(idx)
17            if j == 3: # Target
18                idy = start_ix + (j * 168)
19                seq_x = sequences[idxs, :]
20                seq_y = sequences[idy, 0]
21                y.append(seq_y)
22                X.append(seq_x)
23
24    return X, y

```

شکل 180: تابع ساخت دیتاست برای حالت ماهانه

مدل‌ها و... دقیقاً مانند حالت قبل است به همین دلیل من به آن‌ها اشاره نمی‌کنم.

به دلیل این که در این حالت دیتاهای ما به خاطر فاصله‌های زمانی زیاد خیلی کم می‌شود من بررسی کل داده‌ها را برای این بررسی در نظر می‌گیرم که تعداد 43799 هست.

در ادامه نیز می‌تواند حداکثر داده‌ها در هر کدام از حالت‌ها را مشاهده نمایید:

جدول ۵: تعداد داده‌ها در هر حالت با استفاده از کل داده‌ها

حالت	محاسبه	تعداد دیتاباها
هفتگی	$43799/(7*24*4)$	65
ماهانه	$43799/(7*24)$	260

ابتدا نتایج برای حالت ماهانه را می‌آورم:

حالت ماهانه:

در این حالت $Batch_Size=5$ هست. همچنین $Epoch=200$ هست.

در این حالت در مجموع 18 حالت را باید بررسی کنم که در ادامه آن‌ها را قرار می‌دهم.

► ۱- بررسی شبکه RNN در حالت **MSE** و **Adam**: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 181: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.0027145068533718586 , Valid Loss : => 0.005995448927084605
***->>>-----<<<-***>
step : 1 Train loss : 0.0024210677202790975 , Valid Loss : => 0.0029987676069140436
***->>>-----<<<-***>
step : 2 Train loss : 0.0016362265078350901 , Valid Loss : => 0.0019425715630253155
***->>>-----<<<-***>
step : 3 Train loss : 0.0015291853761300444 , Valid Loss : => 0.0018785338848829269
***->>>-----<<<-***>
step : 4 Train loss : 0.0013744240766391158 , Valid Loss : => 0.0019151617462436358
***->>>-----<<<-***>
step : 5 Train loss : 0.0013503111759200692 , Valid Loss : => 0.0019335030267635981
***->>>-----<<<-***>

```

شکل 182: آپاک اول مدل RNN

```

***->>>-----<<<-***>
step : 193 Train loss : 0.0005015740427188575 , Valid Loss : => 0.00445551003019015
***->>>-----<<<-***>
step : 194 Train loss : 0.0004952391982078553 , Valid Loss : => 0.004449400429924329
***->>>-----<<<-***>
step : 195 Train loss : 0.0005053711007349193 , Valid Loss : => 0.004515026820202669
***->>>-----<<<-***>
step : 196 Train loss : 0.0005026596551761031 , Valid Loss : => 0.004547466213504474
***->>>-----<<<-***>
step : 197 Train loss : 0.000492344789672643 , Valid Loss : => 0.004453796272476514
***->>>-----<<<-***>
step : 198 Train loss : 0.0005149036459624767 , Valid Loss : => 0.004529818954567114
***->>>-----<<<-***>
step : 199 Train loss : 0.0005385069735348224 , Valid Loss : => 0.00452942413588365
***->>>-----<<<-***>

```

شکل 183: آپاک آخر مدل RNN

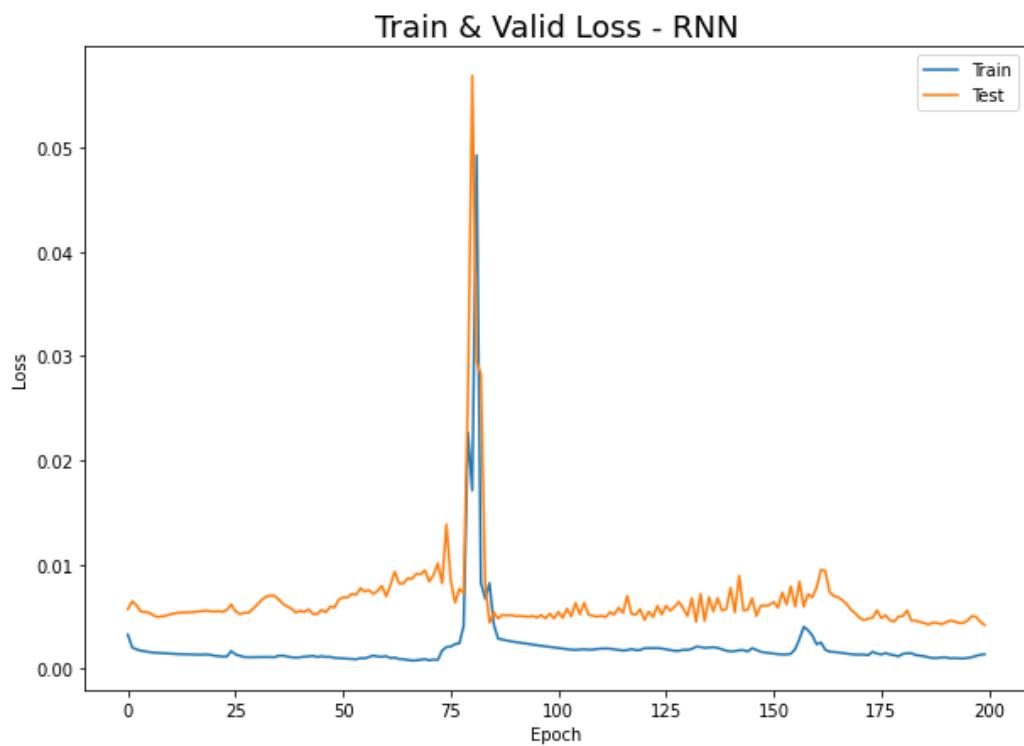
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 3.9094274044036865 Sec.
*****
=====
```

شکل 184: زمان اجرای آموزش در حالت مدل RNN و با MSE و Adam

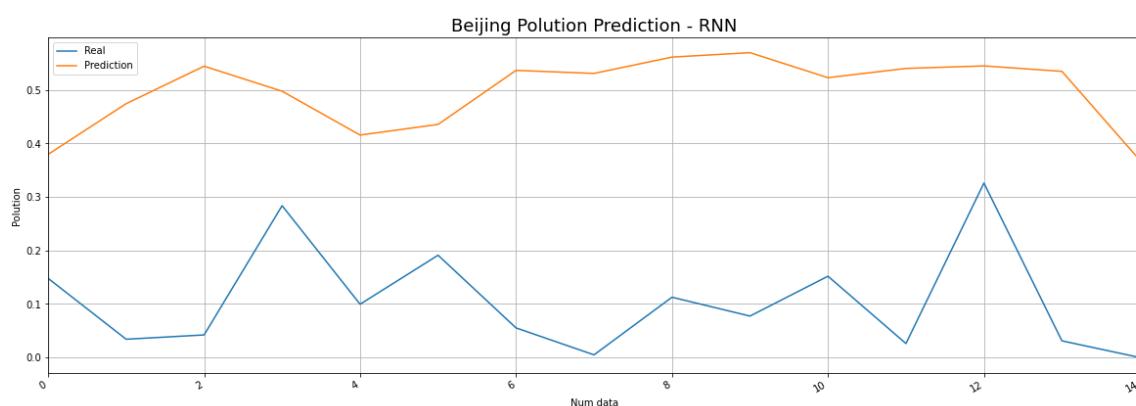
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 185: نمودار Loss در حالت مدل RNN با MSE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 186: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و Adam

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0046171765774488446
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 187: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم.

و از بررسی شکل مقدار واقعی و پیش‌بینی شده نیز می‌توانی متوجه شویم که شبکه توانسته است که روند زیاد و کم شدن مقدار آلودگی را یاد بگیرد اما خوب دقت کافی را در اختیار ندارد.

► 2- بررسی شبکه RNN در حالت MAE و Adam: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 188: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.018995155170559882 , Valid Loss : => 0.019876080751419067
***->>-----<<<-***
step : 1 Train loss : 0.017573208026587964 , Valid Loss : => 0.013702833652496338
***->>-----<<<-***
step : 2 Train loss : 0.017184275761246683 , Valid Loss : => 0.015229268123706181
***->>-----<<<-***
step : 3 Train loss : 0.014637033566832542 , Valid Loss : => 0.01384997417529424
***->>-----<<<-***
step : 4 Train loss : 0.01436897560954094 , Valid Loss : => 0.014078417172034582
***->>-----<<<-***
step : 5 Train loss : 0.01337546531111002 , Valid Loss : => 0.01417627657453219
***->>-----<<<-***
```

شکل 189: ایک اول مدل RNN

```

***->>-----<<<-***  

step : 193 Train loss : 0.01014621637761593 , Valid Loss : => 0.019183133790890376  

***->>-----<<<-***  

step : 194 Train loss : 0.010695481784641742 , Valid Loss : => 0.017800089716911317  

***->>-----<<<-***  

step : 195 Train loss : 0.009682839773595332 , Valid Loss : => 0.01844548831383387  

***->>-----<<<-***  

step : 196 Train loss : 0.00996481891721487 , Valid Loss : => 0.020855502287546793  

***->>-----<<<-***  

step : 197 Train loss : 0.010797698348760605 , Valid Loss : => 0.018391554554303486  

***->>-----<<<-***  

step : 198 Train loss : 0.010098442696034908 , Valid Loss : => 0.018259582420190174  

***->>-----<<<-***  

step : 199 Train loss : 0.010337962098419666 , Valid Loss : => 0.018145627280076345  

***->>-----<<<-***
```

شکل 190: ۵ ایپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

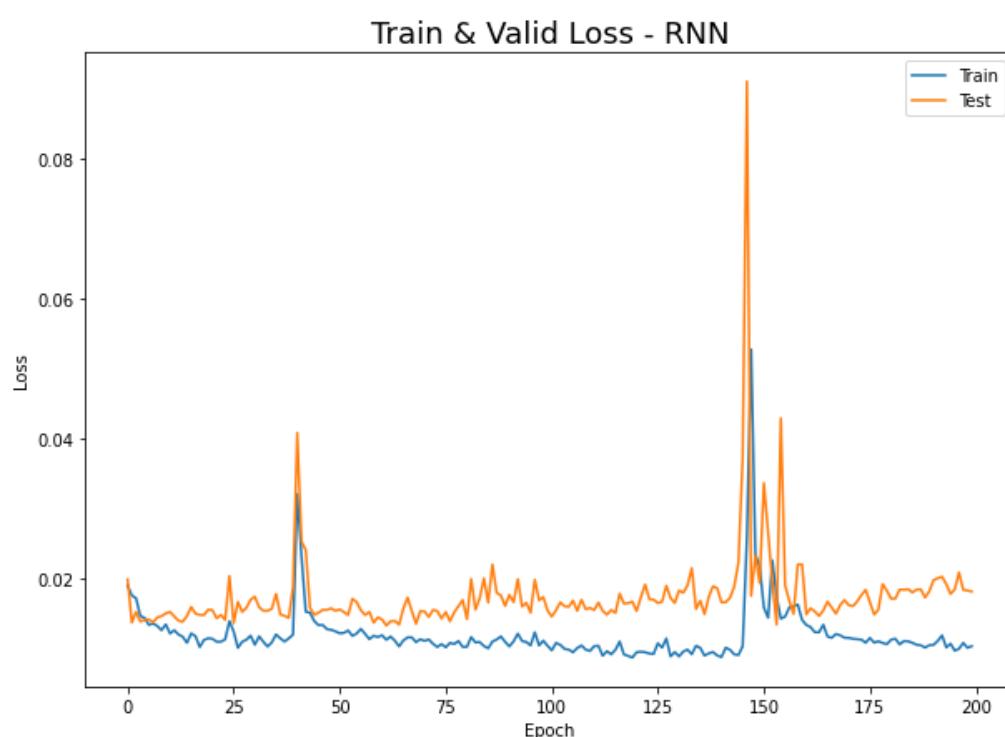
The total Training Time is Equal with ==> : 5.369305372238159 Sec.  

*****  

=====
```

شکل 191: زمان اجرای آموزش در حالت مدل RNN و با MAE و Adam

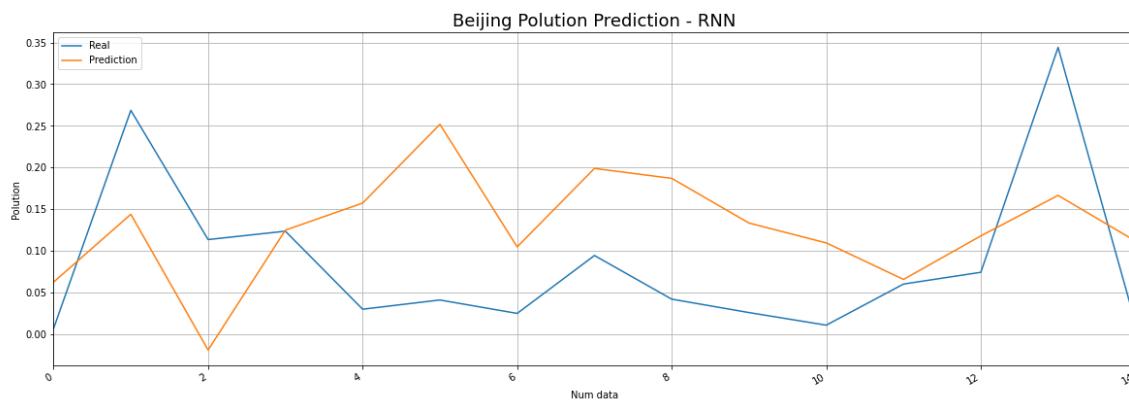
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 192: نمودار Loss در حالت مدل RNN با MAE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 193: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت Adam و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.01811915934085846
>>>-----*****-----<<<
>>>-----<<<
#####
#
```

شکل 194: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم.

و از بررسی شکل مقدار واقعی و پیش‌بینی شده نیز می‌توانی متوجه شویم که شبکه توانسته است که روند زیاد و کم شدن مقدار آلودگی را یاد بگیرد اما خوب دقت کافی را در اختیار ندارد.

► 3- بررسی شبکه RNN در حالت MSE و ADAGrad و حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)
```

شکل 195: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می پردازیم:

```
step : 0 Train loss : 0.002059369618073106 , Valid Loss : => 0.0016567523901661236  
***->>>-----<<<-***  
step : 1 Train loss : 0.0016436615632846952 , Valid Loss : => 0.0011269247469802698  
***->>>-----<<<-***  
step : 2 Train loss : 0.001507067703641951 , Valid Loss : => 0.0010587704678376515  
***->>>-----<<<-***  
step : 3 Train loss : 0.001451271460391581 , Valid Loss : => 0.0010326983251919349  
***->>>-----<<<-***  
step : 4 Train loss : 0.0014128131046891213 , Valid Loss : => 0.001020789270599683  
***->>>-----<<<-***  
step : 5 Train loss : 0.001382403769530356 , Valid Loss : => 0.0010150961577892303  
***->>>-----<<<-***
```

شکل ۱۹۶: آپاک اول مدل RNN

```
***->>>-----<<<-***  
step : 193 Train loss : 0.0006184611679054797 , Valid Loss : => 0.002181215149660905  
***->>>-----<<<-***  
step : 194 Train loss : 0.0006170529150404036 , Valid Loss : => 0.0021871703676879404  
***->>>-----<<<-***  
step : 195 Train loss : 0.0006156482477672398 , Valid Loss : => 0.002193141511331002  
***->>>-----<<<-***  
step : 196 Train loss : 0.0006142469984479249 , Valid Loss : => 0.0021991278665761155  
***->>>-----<<<-***  
step : 197 Train loss : 0.0006128487369278446 , Valid Loss : => 0.0022051312029361726  
***->>>-----<<<-***  
step : 198 Train loss : 0.0006114543834701181 , Valid Loss : => 0.002211150744309028  
***->>>-----<<<-***  
step : 199 Train loss : 0.0006100632378365844 , Valid Loss : => 0.0022171868942677973  
***->>>-----<<<-***
```

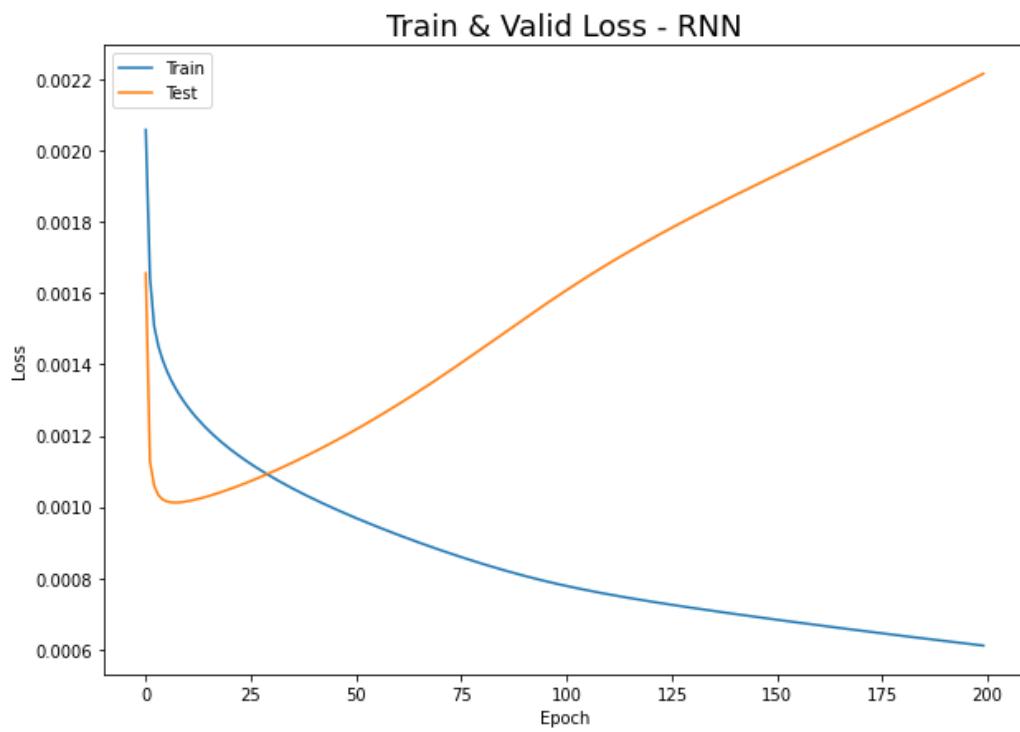
شکل ۱۹۷: آپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می توانید که در ادامه مشاهده نمایید:

```
=====  
*****  
The total Training Time is Equal with ==> : 3.6312246322631836 Sec.  
*****  
=====
```

شکل ۱۹۸: زمان اجرای آموزش در حالت مدل RNN و با MSE و AdaGrad

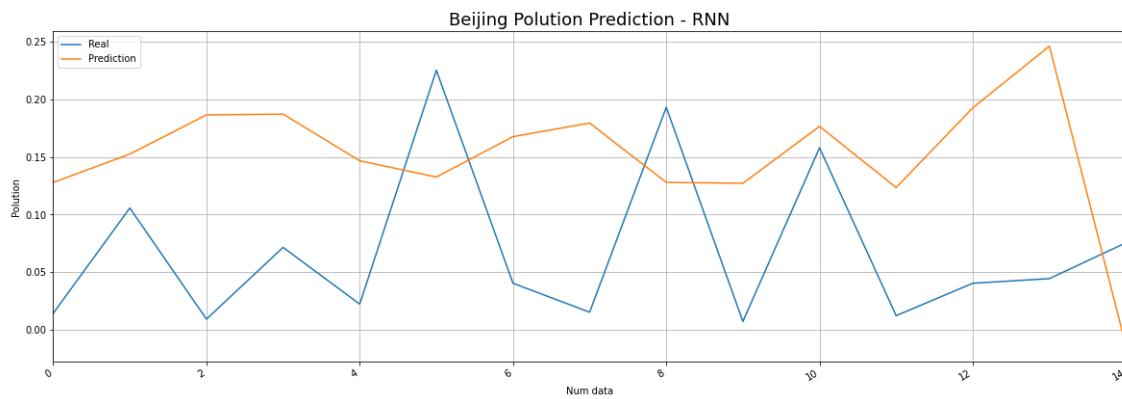
سپس در ادامه می توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 199: نمودار Loss در حالت مدل RNN با MSE و AdaGrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 200: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و AdaGrad

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 0.002217333360264699
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 201: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت حتی نتیجه از حالت‌های قبلی نیز بدتر شده که با Adam کار می‌کردیم و در این حالت حتی نتوانسته است که شبکه ما روندها و کلیت صعود و نزول‌ها را نیز یاد بگیرد.

همچنین نمودار Loss آن بسیار عملکرد بدی دارد و اصلاً به سمت همگرایی نرفته است.

► 4- بررسی شبکه RNN در حالت MAE و ADAGrad: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)

```

شکل 202: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.019460344687104225 , Valid Loss : => 0.015586857497692109
***->>>-----<<<-***>
step : 1 Train loss : 0.016449468731880187 , Valid Loss : => 0.015205195049444835
***->>>-----<<<-***>
step : 2 Train loss : 0.015782483965158463 , Valid Loss : => 0.015430008620023727
***->>>-----<<<-***>
step : 3 Train loss : 0.015244521424174309 , Valid Loss : => 0.015015538036823272
***->>>-----<<<-***>
step : 4 Train loss : 0.014432912394404411 , Valid Loss : => 0.014925585438807806
***->>>-----<<<-***>
step : 5 Train loss : 0.013893143497407437 , Valid Loss : => 0.01493879904349645
***->>>-----<<<-***>

```

شکل 203: ایپاک اول مدل RNN

```

***->>-----<<<-***  

step : 193 Train loss : 0.00966525698080659 , Valid Loss : => 0.01584487681587537  

***->>-----<<<-***  

step : 194 Train loss : 0.00960981797426939 , Valid Loss : => 0.01576998978853226  

***->>-----<<<-***  

step : 195 Train loss : 0.009420211594551801 , Valid Loss : => 0.0158395787080129  

***->>-----<<<-***  

step : 196 Train loss : 0.009661351088434458 , Valid Loss : => 0.015737877537806828  

***->>-----<<<-***  

step : 197 Train loss : 0.009449965320527554 , Valid Loss : => 0.015880963206291197  

***->>-----<<<-***  

step : 198 Train loss : 0.009314382262527943 , Valid Loss : => 0.01579939847191175  

***->>-----<<<-***  

step : 199 Train loss : 0.009432723708450795 , Valid Loss : => 0.015740745514631272  

***->>-----<<<-***
```

شکل 204: ایپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

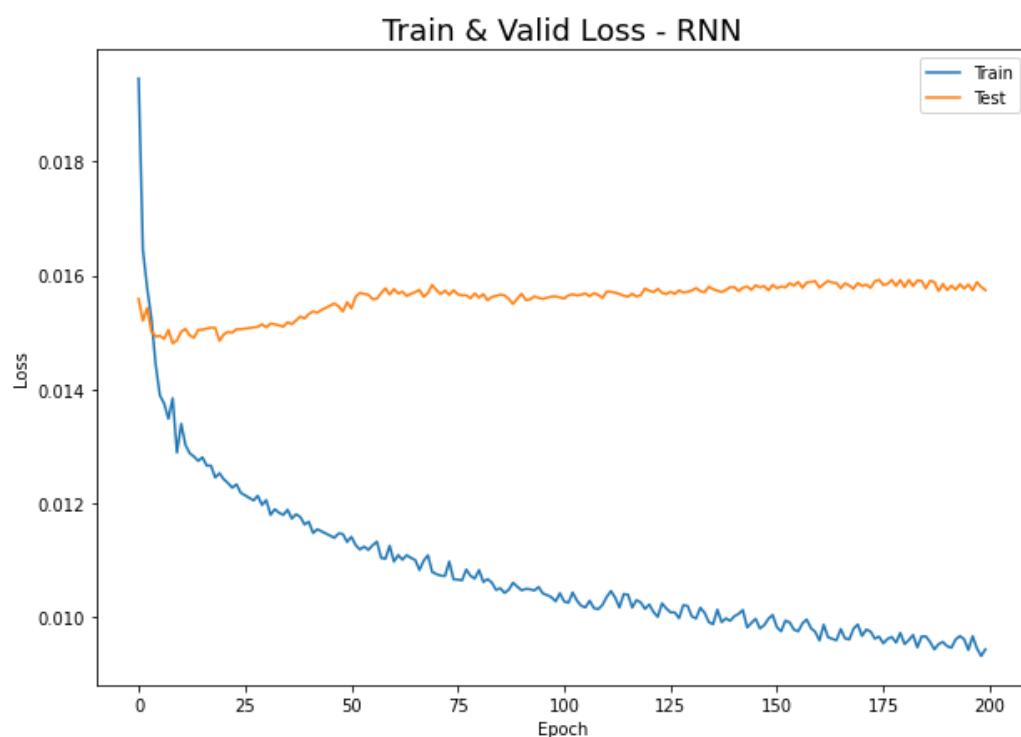
The total Training Time is Equal with ==> : 3.899285078048706 Sec.  

*****  

=====
```

شکل 205: زمان اجرای آموزش در حالت مدل RNN و با MAE و AdaGrad

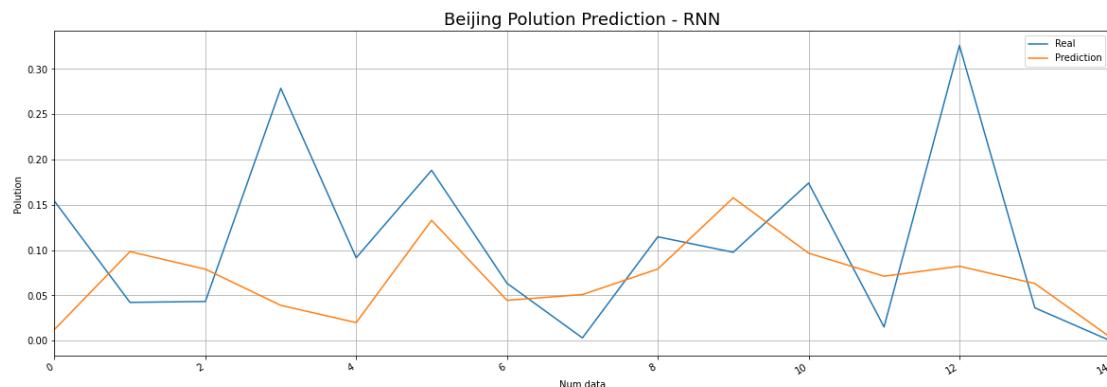
سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 206: نمودار Loss در حالت مدل RNN با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 207: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت AdaGrad و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.015758364895979562
>>>-----*****-----<<<
>>>-----<<<
#####
#
```

شکل 208: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت حتی نتیجه از حالت‌های قبلی نیز بدتر شده که با Adam کار می‌کردیم و در این حالت حتی نتوانسته است که شبکه ما روندها و کلیت صعود و نزول‌ها را نیز یاد بگیرد.

همچنین نمودار آن بسیار عملکرد بدی دارد و اصلاً به سمت همگرایی نرفته است.

► 5- بررسی شبکه RNN در حالت RMSprop و MSE: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)

```

شکل 209: تعریف مدل و Loss در مدل RNN و تابع Optimizer

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 1.0006142895668746 , Valid Loss : => 0.07229548494021097
***->>-----<<<-***
step : 1 Train loss : 0.1402466404438019 , Valid Loss : => 0.07773854533831279
***->>-----<<<-***
step : 2 Train loss : 0.033848385084420445 , Valid Loss : => 0.015196912363171578
***->>-----<<<-***
step : 3 Train loss : 0.03029071512632072 , Valid Loss : => 0.012694301704565683
***->>-----<<<-***
step : 4 Train loss : 0.011319149313494562 , Valid Loss : => 0.002987049085398515
***->>-----<<<-***
step : 5 Train loss : 0.004034650474786758 , Valid Loss : => 0.007250481471419335
***->>-----<<<-***

```

شکل 210: ایپاک اول مدل RNN

```

***->>-----<<<-***
step : 193 Train loss : 0.0012916588177904486 , Valid Loss : => 0.00214681647097071
***->>-----<<<-***
step : 194 Train loss : 0.0012364051211625337 , Valid Loss : => 0.001750387034068505
***->>-----<<<-***
step : 195 Train loss : 0.0012111773481592535 , Valid Loss : => 0.0017555223467449347
***->>-----<<<-***
step : 196 Train loss : 0.0011763758375309408 , Valid Loss : => 0.0018100121058523655
***->>-----<<<-***
step : 197 Train loss : 0.0011477863625623285 , Valid Loss : => 0.0019249794694284597
***->>-----<<<-***
step : 198 Train loss : 0.0011236334638670088 , Valid Loss : => 0.001957693990940849
***->>-----<<<-***
step : 199 Train loss : 0.0010994448116980493 , Valid Loss : => 0.0018853671538333098
***->>-----<<<-***

```

شکل 211: ایپاک آخر مدل RNN

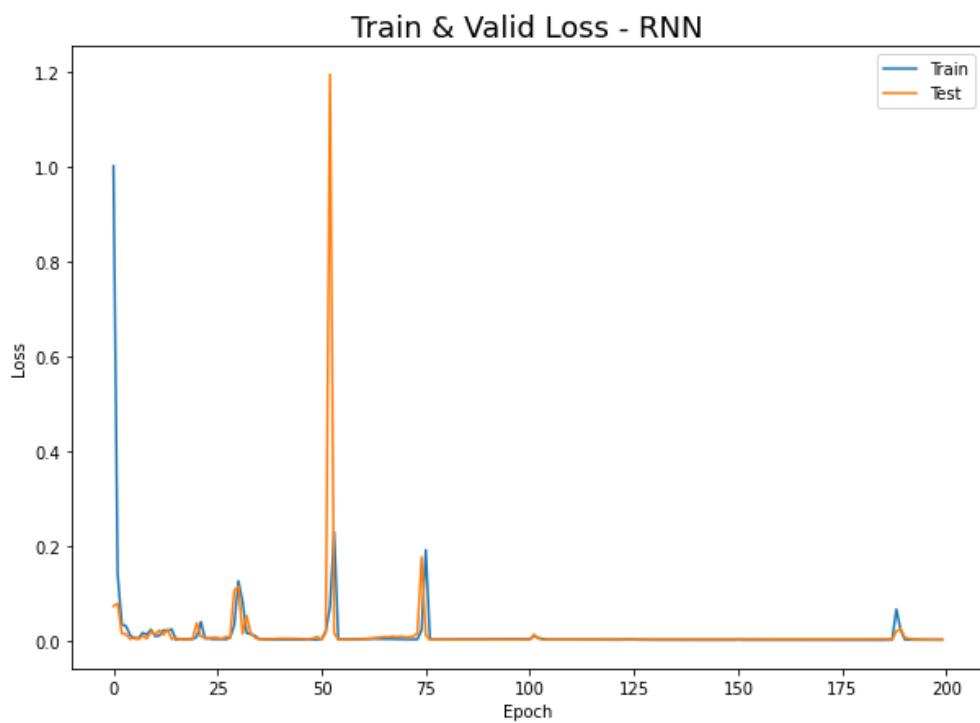
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 4.101718425750732 Sec.
*****
=====
```

شکل 212: زمان اجرای آموزش در حالت مدل RNN و RMSprop و با MSE

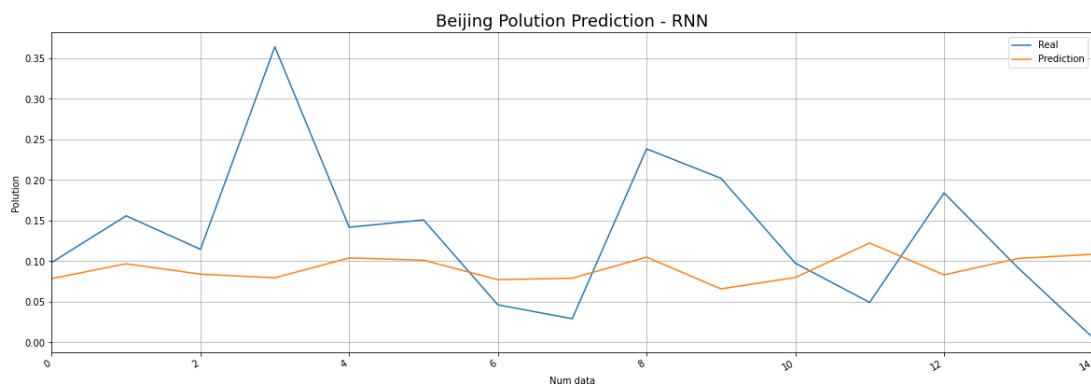
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 213: نمودار Loss در حالت مدل RNN با MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 214: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 0.0018187905040880044
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 215: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. در این حالت کمی دقت عملکرد مدل بهتر شده است نسبت به حالاتی قبلی.

همچنین نمودار Loss آن بسیار عملکرد بهتری نسبت به حالاتی Adam و AdaGrad دارد.

► 6- بررسی شبکه RNN در حالت MAE و RMSprop: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
```

شکل 216: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.33576523646712303 , Valid Loss : => 0.03546156684557597
***->>-----<<<-***
step : 1 Train loss : 0.19048140242695807 , Valid Loss : => 0.07034612993399302
***->>-----<<<-***
step : 2 Train loss : 0.10445650443434715 , Valid Loss : => 0.08418776392936707
***->>-----<<<-***
step : 3 Train loss : 0.052646561414003375 , Valid Loss : => 0.046698625385761264
***->>-----<<<-***
step : 4 Train loss : 0.03197813890874386 , Valid Loss : => 0.019071250657240548
***->>-----<<<-***
step : 5 Train loss : 0.01430604573339224 , Valid Loss : => 0.017922258377075194
***->>-----<<<-***
```

شکل 217: ایپاک اول مدل RNN

```
***->>-----<<<-***
step : 193 Train loss : 0.009659001026302577 , Valid Loss : => 0.01550788606206576
***->>-----<<<-***
step : 194 Train loss : 0.010075229350477458 , Valid Loss : => 0.016125263770421346
***->>-----<<<-***
step : 195 Train loss : 0.009525408409535885 , Valid Loss : => 0.015742106239000957
***->>-----<<<-***
step : 196 Train loss : 0.010268072374165058 , Valid Loss : => 0.01580213208993276
***->>-----<<<-***
step : 197 Train loss : 0.01049927631393075 , Valid Loss : => 0.015527755518754323
***->>-----<<<-***
step : 198 Train loss : 0.010435365699231624 , Valid Loss : => 0.016198273499806723
***->>-----<<<-***
step : 199 Train loss : 0.01027410313487053 , Valid Loss : => 0.01590216209491094
***->>-----<<<-***
```

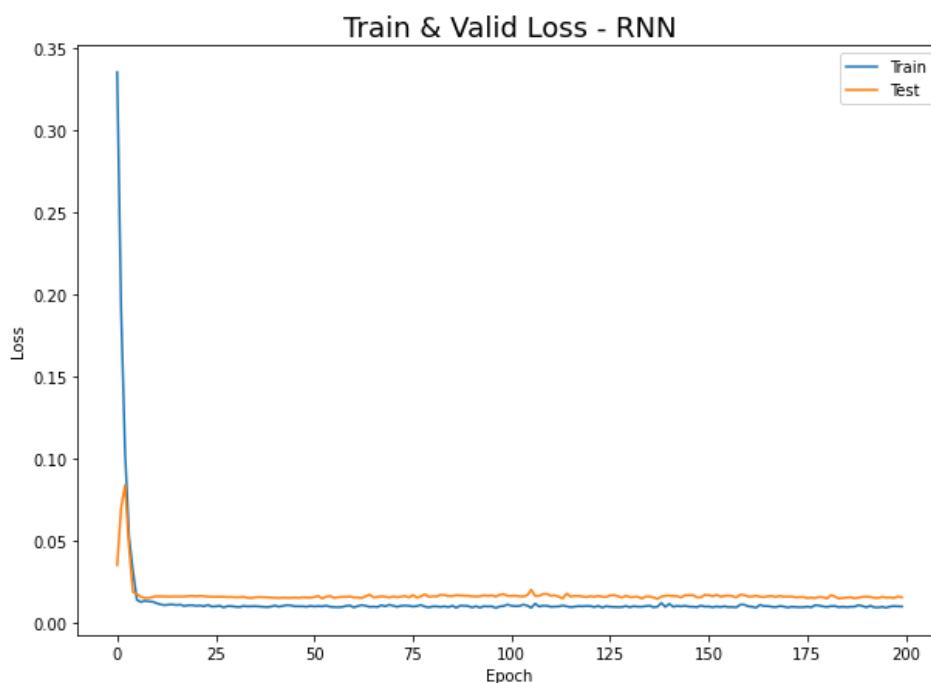
شکل 218: ایپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 3.7652227878570557 Sec.  
*****  
=====
```

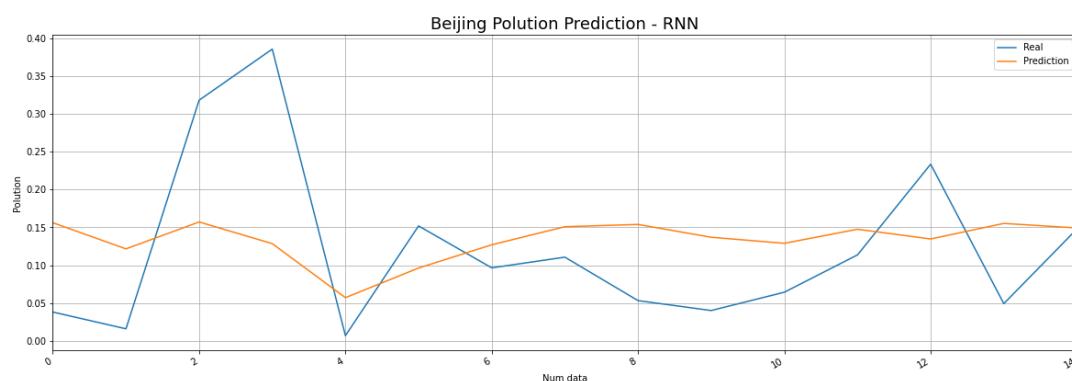
شکل 219: زمان اجرای آموزش در حالت مدل RNN و با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مربوط به این LOSS‌ها را مشاهده نمایید:



شکل 220: نمودار Loss در حالت مدل RNN با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 221: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MAE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.015815315395593645
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 222: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شیکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. در این حالت کمی دقیق عملکرد مدل بهتر شده است نسبت به حالاتی قبلی.

همچنین نمودار Loss آن بسیار عملکرد بهتری نسبت به حالاتی Adam و AdaGrad دارد.

► 7- بررسی شبکه LSTM در حالت MSE و Adam: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 223: تعریف مدل و Loss در مدل LSTM و Optimizer وتابع

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.010185169129787635 , Valid Loss : => 0.0012245348887518048
***->>-----<<-***
step : 1 Train loss : 0.00689179099475344 , Valid Loss : => 0.0009855808534969886
***->>-----<<-***
step : 2 Train loss : 0.0066926017248382175 , Valid Loss : => 0.000978816719725728
***->>-----<<-***
step : 3 Train loss : 0.006331891023243467 , Valid Loss : => 0.00105449294205755
***->>-----<<-***
step : 4 Train loss : 0.006315735254126291 , Valid Loss : => 0.001044399058446288
***->>-----<<-***
step : 5 Train loss : 0.006235987739637494 , Valid Loss : => 0.0010090152267366649
***->>-----<<-***
```

شکل 224: ایپاک اول مدل LSTM

```

***->>-----<<-***
step : 193 Train loss : 0.0023740344758455953 , Valid Loss : => 0.001332360195616881
***->>-----<<-***
step : 194 Train loss : 0.002367658765676121 , Valid Loss : => 0.001332307777677973
***->>-----<<-***
step : 195 Train loss : 0.0023612702498212457 , Valid Loss : => 0.0013322496476272743
***->>-----<<-***
step : 196 Train loss : 0.0023548692154387635 , Valid Loss : => 0.001332185138016939
***->>-----<<-***
step : 197 Train loss : 0.0023484571293617288 , Valid Loss : => 0.001332115971793731
***->>-----<<-***
step : 198 Train loss : 0.002342033968307078 , Valid Loss : => 0.0013320405346651872
***->>-----<<-***
step : 199 Train loss : 0.002335600492854913 , Valid Loss : => 0.0013319601770490409
***->>-----<<-***

```

شکل 225: ایپاک آخر مدل LSTM

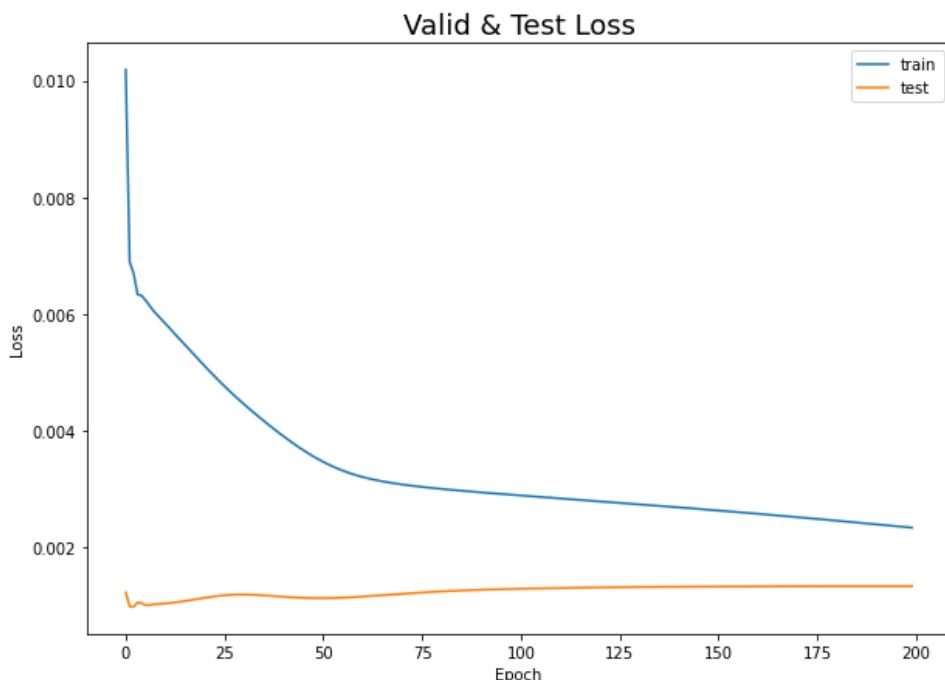
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 12.099555015563965 Sec.
*****
=====
```

شکل 226: زمان اجرای آموزش در حالت مدل LSTM و با MSE و Adam

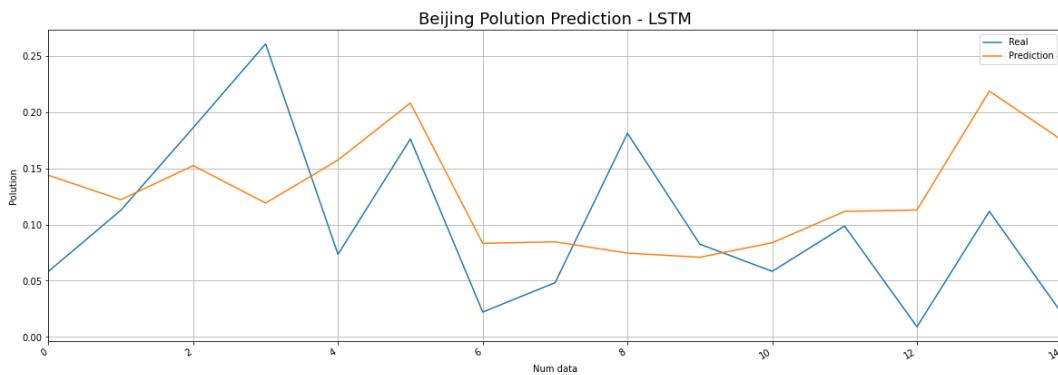
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 227: نمودار Loss در حالت مدل LSTM با Adam و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 228: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت Adam و MSE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0013319601770490409
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 229: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین در این حالت مقدار Loss به دست آمده نیز کاهش پیدا کرده که نشان می‌دهد که شبکه دقت بهتری و بالاتری نسبت به حالت قبل دارد.

نکته دیگری که خوب است که به آن اشاره کنم این است که زمان آموزش در این حالت نسبت به حالت قبلی افزایش چشم‌گیری پیدا کرده این که این موضوع را می‌توانستیم در قسمت اول سوال نیز ببینیم.

► 8- بررسی شبکه LSTM در حالت MAE و Adam: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 230: تعریف مدل و Loss و تابع Optimizer در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.05424093417823315 , Valid Loss : => 0.012866015483935674
***->>-----<<<-***
step : 1 Train loss : 0.04505716363588969 , Valid Loss : => 0.012677980462710063
***->>-----<<<-***
step : 2 Train loss : 0.04570261339346568 , Valid Loss : => 0.012471593916416168
***->>-----<<<-***
step : 3 Train loss : 0.044380438327789304 , Valid Loss : => 0.012613356361786525
***->>-----<<<-***
step : 4 Train loss : 0.04391596913337707 , Valid Loss : => 0.012780799716711044
***->>-----<<<-***
step : 5 Train loss : 0.04364517852663994 , Valid Loss : => 0.012692747761805852
***->>-----<<<-***

```

شکل 231: ایپاک اول مدل LSTM

```

***->>-----<<<-***
step : 193 Train loss : 0.030415988216797512 , Valid Loss : => 0.01352580040693283
***->>-----<<<-***
step : 194 Train loss : 0.030456421027580897 , Valid Loss : => 0.013109574218591054
***->>-----<<<-***
step : 195 Train loss : 0.029609427539010844 , Valid Loss : => 0.013420768082141876
***->>-----<<<-***
step : 196 Train loss : 0.030186622093121212 , Valid Loss : => 0.013594546169042588
***->>-----<<<-***
step : 197 Train loss : 0.029903709578017395 , Valid Loss : => 0.01357187752922376
***->>-----<<<-***
step : 198 Train loss : 0.029794937061766783 , Valid Loss : => 0.013244854162136714
***->>-----<<<-***
step : 199 Train loss : 0.029776595098276935 , Valid Loss : => 0.01321098804473877
***->>-----<<<-***

```

شکل 232: ایپاک آخر مدل LSTM

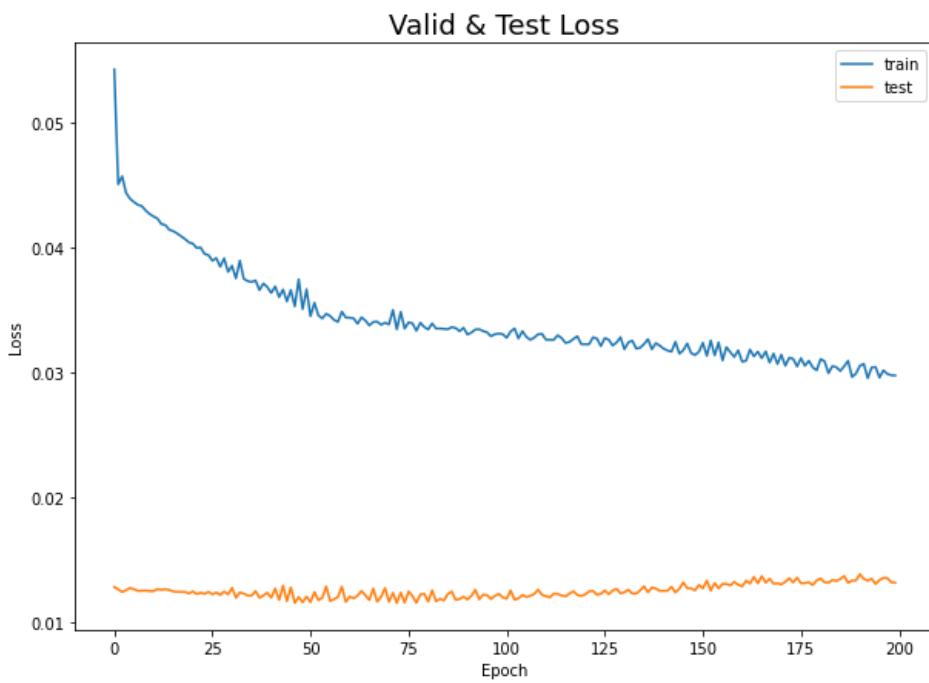
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 9.896137714385986 Sec.
*****
=====
```

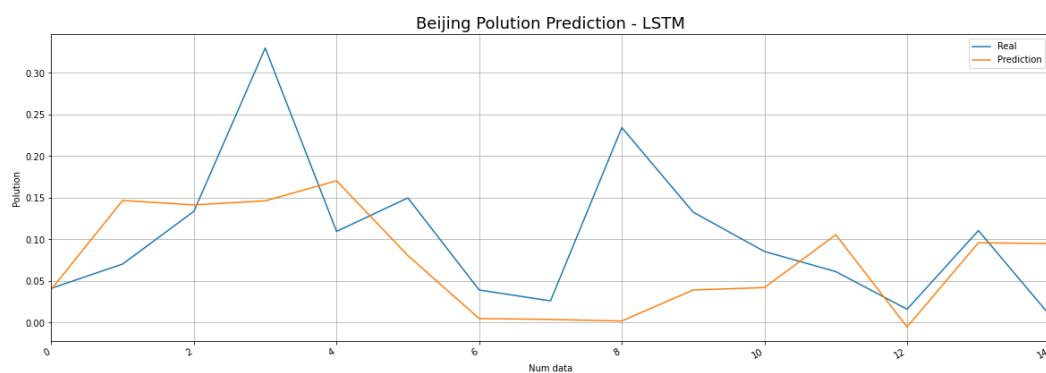
شکل 233: زمان اجرای آموزش در حالت مدل LSTM و با Adam و MAE

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 234: نمودار Loss در حالت مدل LSTM با Adam و MAE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 235: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت Adam و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.01321098804473877
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 236: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین در این حالت مقدار Loss به دست آمده نیز کاهش پیدا کرده که نشان می‌دهد که شبکه دقیق‌تر و بالاتری نسبت به حالت قبل دارد.

نکته دیگری که خوب است که به آن اشاره کنم این است که زمان آموزش در این حالت نسبت به حالت قبلی افزایش چشم‌گیری پیدا کرده این که این موضوع را می‌توانستیم در قسمت اول سوال نیز ببینیم.

► ۹- بررسی شبکه LSTM در حالت MSE و AdaGrad: حالت ماهانه

ابتدا به تعریف Loss و تابع Optimizer می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 237: تعریف مدل و Loss و تابع Optimizer در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.015322176514503856 , Valid Loss : => 0.0015447244669000307
***->>>-----<<<-***  

step : 1 Train loss : 0.01199770420207642 , Valid Loss : => 0.0012360813096165657
***->>>-----<<<-***  

step : 2 Train loss : 0.010481801772645365 , Valid Loss : => 0.001097583801796039
***->>>-----<<<-***  

step : 3 Train loss : 0.009572372919258972 , Valid Loss : => 0.0010359892000754674
***->>>-----<<<-***  

step : 4 Train loss : 0.008982315496541559 , Valid Loss : => 0.001014895923435688
***->>>-----<<<-***  

step : 5 Train loss : 0.008584664226509631 , Valid Loss : => 0.001016089739277959
***->>>-----<<<-***
```

شکل 238: 5 ایپاک اول مدل LSTM

```

***->>-----<<-***
step : 193 Train loss : 0.007439215496803323 , Valid Loss : => 0.0013414836488664151
***->>-----<<-***
step : 194 Train loss : 0.007438628080611427 , Valid Loss : => 0.0013416664364437263
***->>-----<<-***
step : 195 Train loss : 0.007438042604674896 , Valid Loss : => 0.0013418484789629777
***->>-----<<-***
step : 196 Train loss : 0.007437458743030826 , Valid Loss : => 0.0013420306146144866
***->>-----<<-***
step : 197 Train loss : 0.007436876573289434 , Valid Loss : => 0.0013422122225165366
***->>-----<<-***
step : 198 Train loss : 0.007436296079928676 , Valid Loss : => 0.0013423936441540718
***->>-----<<-***
step : 199 Train loss : 0.007435716968029737 , Valid Loss : => 0.0013425748174389204
***->>-----<<-***

```

شکل 239: ایپاک آخر مدل LSTM

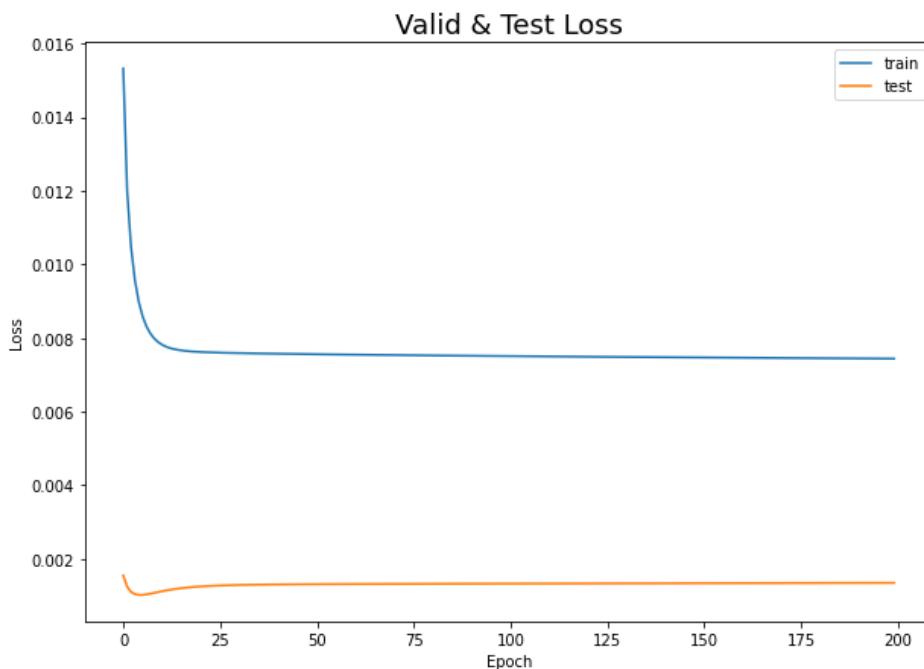
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 8.89835262298584 Sec.
*****
=====
```

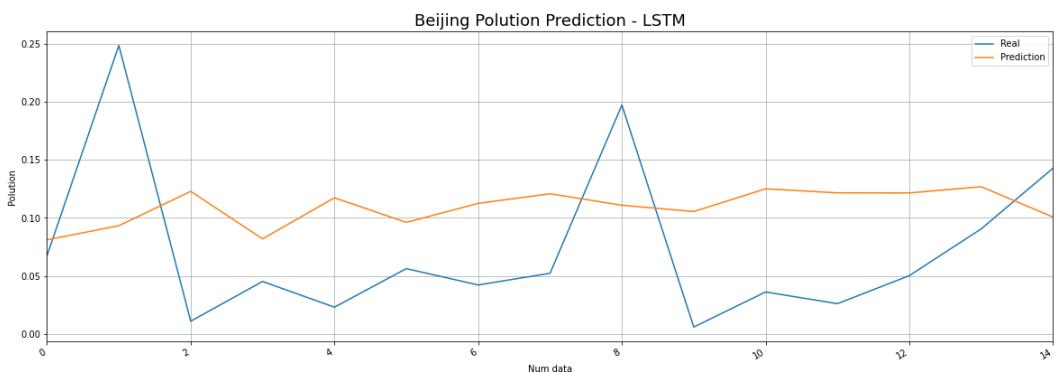
شکل 240: زمان اجرای آموزش در حالت مدل LSTM و با MSE و AdaGrad

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 241: نمودار Loss در حالت مدل LSTM با MSE و AdaGrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 242: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت AdaGrad و MSE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0013425748174389204
>>>-----*****-----<<<
>>>----------<<<
#####

```

شکل 243: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین در این حالت مقدار Loss به دست آمده نیز کاهش پیدا کرده که نشان می‌دهد که شبکه دقت بهتری و بالاتری نسبت به حالت قبل دارد.

نکته دیگری که خوب است که به آن اشاره کنم این است که زمان آموزش در این حالت نسبت به حالت قبلی افزایش چشم‌گیری پیدا کرده این که این موضوع را می‌توانستیم در قسمت اول سوال نیز ببینیم. با بررسی حالت AdaGrad این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف‌تر است.

► 10- بررسی شبکه LSTM در حالت AdaGrad و MAE: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)

```

شکل 244: تعریف مدل و Loss و تابع Optimizer در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.06657924105723699 , Valid Loss : => 0.014905717720588048
***->>>-----<<<-***>
step : 1 Train loss : 0.05417056605219841 , Valid Loss : => 0.013276852170626323
***->>>-----<<<-***>
step : 2 Train loss : 0.04865399884680907 , Valid Loss : => 0.012453635782003402
***->>>-----<<<-***>
step : 3 Train loss : 0.046136669876674814 , Valid Loss : => 0.01237548291683197
***->>>-----<<<-***>
step : 4 Train loss : 0.04509344461063544 , Valid Loss : => 0.012574477990468343
***->>>-----<<<-***>
step : 5 Train loss : 0.04465406884749731 , Valid Loss : => 0.0126502625644207
***->>>-----<<<-***>

```

شکل 245: ایپاک اول مدل LSTM

```

***->>>-----<<<-***>
step : 193 Train loss : 0.04173403829336166 , Valid Loss : => 0.012707350154717764
***->>>-----<<<-***>
step : 194 Train loss : 0.04172622275849183 , Valid Loss : => 0.012704433997472127
***->>>-----<<<-***>
step : 195 Train loss : 0.041717136402924855 , Valid Loss : => 0.012709558755159379
***->>>-----<<<-***>
step : 196 Train loss : 0.04171266034245491 , Valid Loss : => 0.012706655263900756
***->>>-----<<<-***>
step : 197 Train loss : 0.04170488603413105 , Valid Loss : => 0.01270375947157542
***->>>-----<<<-***>
step : 198 Train loss : 0.04169634481271108 , Valid Loss : => 0.012708855420351028
***->>>-----<<<-***>
step : 199 Train loss : 0.04169138645132383 , Valid Loss : => 0.012705971052249273
***->>>-----<<<-***>

```

شکل 246: ایپاک آخر مدل LSTM

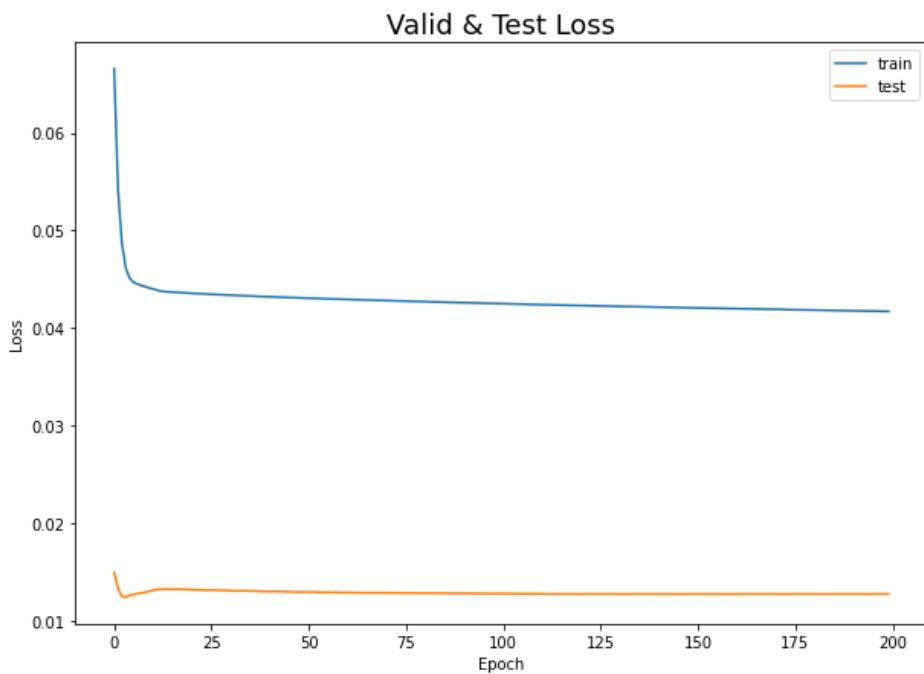
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
***** The total Training Time is Equal with ==> : 9.03040885925293 Sec. *****
=====
```

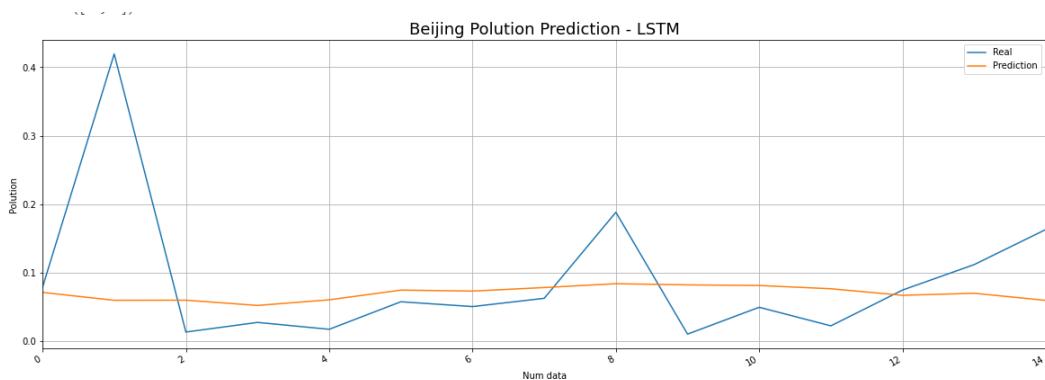
شکل 247: زمان اجرای آموزش در حالت مدل LSTM و با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 248: نمودار Loss و MAE در حالت مدل LSTM با AdaGrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 249: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت AdaGrad و MSE

```
#####
>>>-----<<<
>>>-----*****
**** Test Loss :==>> 0.012705971052249273
>>>-----*****
>>>-----<<<
#####
```

شکل 250: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین در این حالت مقدار Loss به دست آمده نیز کاهش پیدا کرده که نشان می‌دهد که شبکه دقت بهتری و بالاتری نسبت به حالت قبل دارد.

نکته دیگری که خوب است که به آن اشاره کنم این است که زمان آموزش در این حالت نسبت به حالت قبلی افزایش چشم‌گیری پیدا کرده این که این موضوع را می‌توانستیم در قسمت اول سوال نیز ببینیم.

با بررسی حالت AdaGrad این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف‌تر است.

► 11- بررسی شبکه LSTM در حالت ماهانه: MSE و RMSprop

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شكل 251: تعریف مدل و Optimizer وتابع Loss در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss :  0.00809717207836608 , Valid Loss : =>  0.0007982475062211354
***->>>-----<<<-***  

step : 1 Train loss :  0.006477952189743519 , Valid Loss : =>  0.0007910632373144229
***->>>-----<<<-***  

step : 2 Train loss :  0.006403069524094462 , Valid Loss : =>  0.0007821890059858561
***->>>-----<<<-***  

step : 3 Train loss :  0.006340295014282068 , Valid Loss : =>  0.0007741581027706464
***->>>-----<<<-***  

step : 4 Train loss :  0.00628313211032804 , Valid Loss : =>  0.0007672654464840889
***->>>-----<<<-***  

step : 5 Train loss :  0.0062300517068554955 , Valid Loss : =>  0.0007613658361757795
***->>>-----<<<-***
```

شكل 252: ایپاک اول مدل LSTM

```

***->>-----<<<-***  

step : 193 Train loss : 0.0035993786218265693 , Valid Loss : => 0.0012201224453747272  

***->>-----<<<-***  

step : 194 Train loss : 0.003583979730804761 , Valid Loss : => 0.001226484247793754  

***->>-----<<<-***  

step : 195 Train loss : 0.003568490454927087 , Valid Loss : => 0.0012328704819083215  

***->>-----<<<-***  

step : 196 Train loss : 0.00355291236191988 , Valid Loss : => 0.0012392794092496236  

***->>-----<<<-***  

step : 197 Train loss : 0.0035372430613885325 , Valid Loss : => 0.0012457091050843397  

***->>-----<<<-***  

step : 198 Train loss : 0.003521483779574434 , Valid Loss : => 0.0012521578619877497  

***->>-----<<<-***  

step : 199 Train loss : 0.0035056340973824264 , Valid Loss : => 0.0012586233206093311  

***->>-----<<<-***
```

شکل 253: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

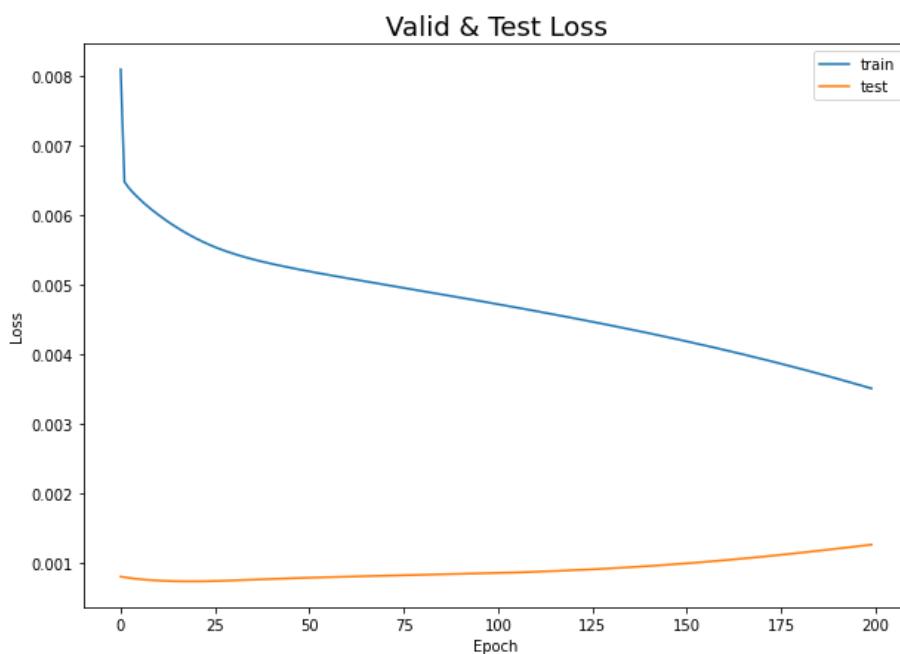
The total Training Time is Equal with ==> : 9.167251348495483 Sec.  

*****  

=====
```

شکل 254: زمان اجرای آموزش در حالت مدل LSTM و با RMSprop و MSE

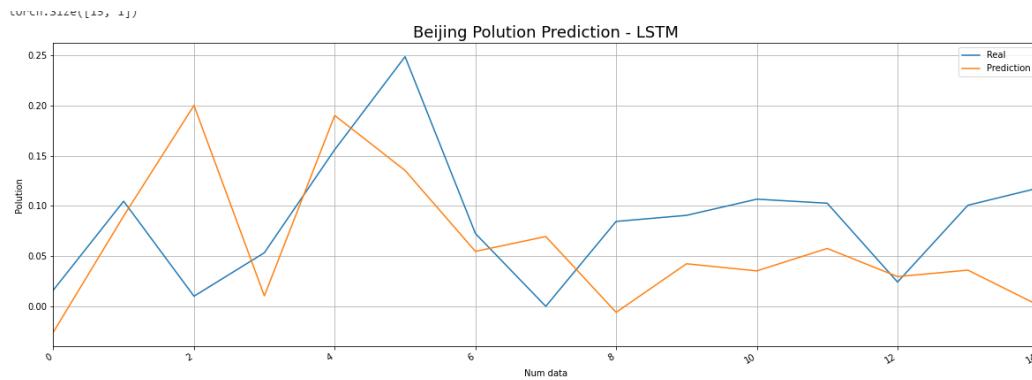
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 255: نمودار Loss در حالت مدل LSTM با RMSprop و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 256: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت MSE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0012586233206093311
>>>-----*****-----<<<
>>>-----*****-----<<<
#####
```

شکل 257: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین در این حالت مقدار Loss به دست آمده نیز کاهش پیدا کرده که نشان می‌دهد که شبکه دقت بهتری و بالاتری نسبت به حالت قبل دارد.

نکته دیگری که خوب است که به آن اشاره کنم این است که زمان آموزش در این حالت نسبت به حالت قبلی افزایش چشم‌گیری پیدا کرده این که این موضوع را می‌توانستیم در قسمت اول سوال نیز ببینیم.

با بررسی حالت RMSprop این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف‌تر است.

► 12- بررسی شبکه LSTM در حالت MAE و RMSprop: حالت ماهانه

ابتدا به تعریف Optimizer و تابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)

```

شکل 258: تعریف مدل و Loss در مدل LSTM و Optimizer

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.05290079166491826 , Valid Loss : => 0.017354288200537363
***->>-----<<<-***
step : 1 Train loss : 0.045542619004845616 , Valid Loss : => 0.017305648823579153
***->>-----<<<-***
step : 2 Train loss : 0.04461119025945663 , Valid Loss : => 0.016463365654150644
***->>-----<<<-***
step : 3 Train loss : 0.0435268538693587 , Valid Loss : => 0.01642754077911377
***->>-----<<<-***
step : 4 Train loss : 0.04349229782819748 , Valid Loss : => 0.016380682835976282
***->>-----<<<-***
step : 5 Train loss : 0.042963125618795554 , Valid Loss : => 0.016067357112963993
***->>-----<<<-***

```

شکل 259: ایپاک اول مدل LSTM

```

***->>-----<<<-***
step : 193 Train loss : 0.028671859515209994 , Valid Loss : => 0.01593522777160009
***->>-----<<<-***
step : 194 Train loss : 0.029308885149657727 , Valid Loss : => 0.015945013364156088
***->>-----<<<-***
step : 195 Train loss : 0.02844027305642764 , Valid Loss : => 0.016392490764458974
***->>-----<<<-***
step : 196 Train loss : 0.02925392227868239 , Valid Loss : => 0.016055882473786674
***->>-----<<<-***
step : 197 Train loss : 0.02877275695403417 , Valid Loss : => 0.015726429720719654
***->>-----<<<-***
step : 198 Train loss : 0.02898490953569611 , Valid Loss : => 0.01596513589223226
***->>-----<<<-***
step : 199 Train loss : 0.028325286197165647 , Valid Loss : => 0.015990523993968962
***->>-----<<<-***

```

شکل 260: ایپاک آخر مدل LSTM

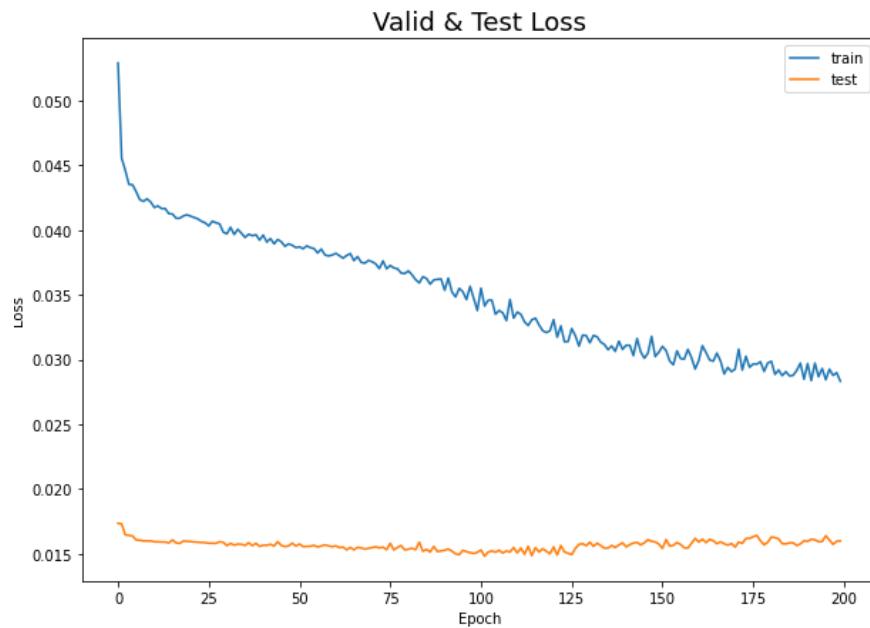
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 11.569271564483643 Sec.
*****
=====
```

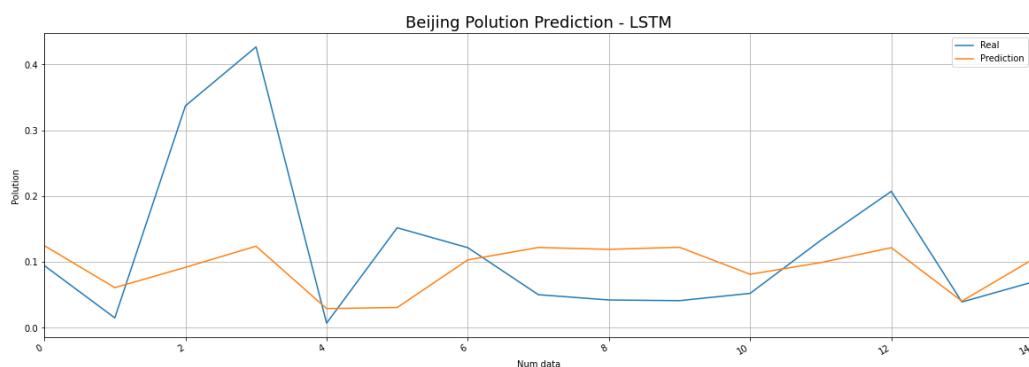
شکل 261: زمان اجرای آموزش در حالت مدل LSTM و MAE و RMSprop

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 262: نمودار Loss و MAE در حالت مدل LSTM با RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 263: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت RMSprop و MAE

```
#####
>>> -----<<<
>>> _*****_*****_*****_*****_-----<<<
**** Test Loss :==>> 0.015990523993968962
>>> _*****_*****_*****_*****_-----<<<
>>> -----<<<
#####

```

شکل 264: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که

ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین در این حالت مقدار Loss به دست آمده نیز کاهش پیدا کرده که نشان می‌دهد که شبکه دقیق‌تر و بالاتری نسبت به حالت قبل دارد.

نکته دیگری که خوب است که به آن اشاره کنم این است که زمان آموزش در این حالت نسبت به حالت قبلی افزایش چشم‌گیری پیدا کرده این که این موضوع را می‌توانستیم در قسمت اول سوال نیز ببینیم. با بررسی حالت RMSprop این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف‌تر است.

► 13- بررسی شبکه GRU در حالت Adam و MSE: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 265: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.0021845926833339037 , Valid Loss : => 0.0015523009933531284
***->>>-----<<<-***  
step : 1 Train loss : 0.0016418621232151054 , Valid Loss : => 0.0012070327376325926
***->>>-----<<<-***  
step : 2 Train loss : 0.0009616474260110408 , Valid Loss : => 0.001112275260190169
***->>>-----<<<-***  
step : 3 Train loss : 0.0010621836030622945 , Valid Loss : => 0.0011134418503691752
***->>>-----<<<-***  
step : 4 Train loss : 0.0009189019445329904 , Valid Loss : => 0.001107342968073984
***->>>-----<<<-***  
step : 5 Train loss : 0.0008966563595458866 , Valid Loss : => 0.0011490948343028625
***->>>-----<<<-***
```

شکل 266: ایپاک اول مدل GRU

```

***->>-----<<<_***  

step : 193 Train loss : 3.975139038630004e-06 , Valid Loss : => 0.0010384719197948774  

***->>-----<<<_***  

step : 194 Train loss : 3.825875623988395e-06 , Valid Loss : => 0.000989306733633081  

***->>-----<<<_***  

step : 195 Train loss : 2.9232175256765915e-06 , Valid Loss : => 0.001011874076599876  

***->>-----<<<_***  

step : 196 Train loss : 2.1924388147454012e-06 , Valid Loss : => 0.0010529283123711746  

***->>-----<<<_***  

step : 197 Train loss : 2.5471837238910667e-06 , Valid Loss : => 0.0010348830837756395  

***->>-----<<<_***  

step : 198 Train loss : 2.2059851039557543e-06 , Valid Loss : => 0.0010250623182704052  

***->>-----<<<_***  

step : 199 Train loss : 1.7553040879647596e-06 , Valid Loss : => 0.001051918293039004  

***->>-----<<<_***
```

شکل 267: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

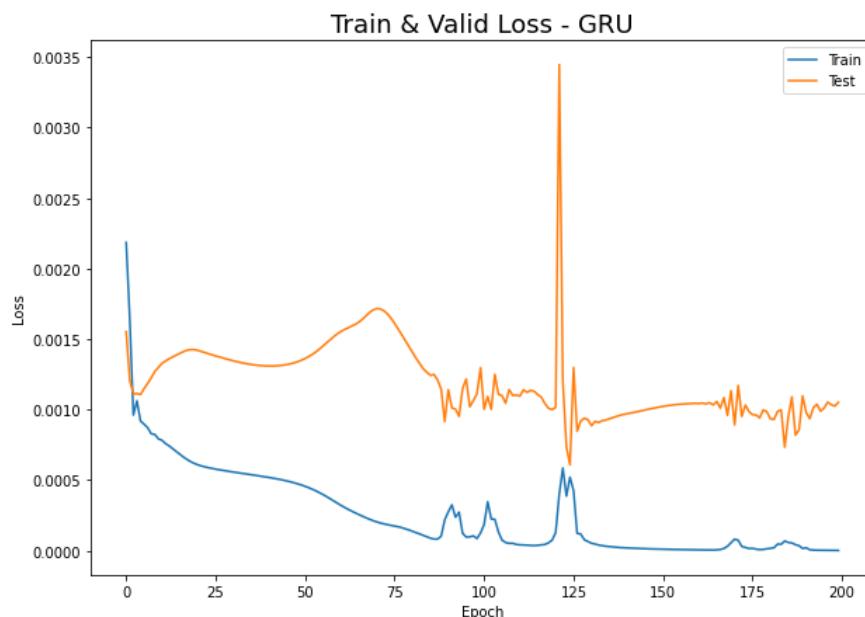
The total Training Time is Equal with ==> : 9.314673662185669 Sec.  

*****  

=====
```

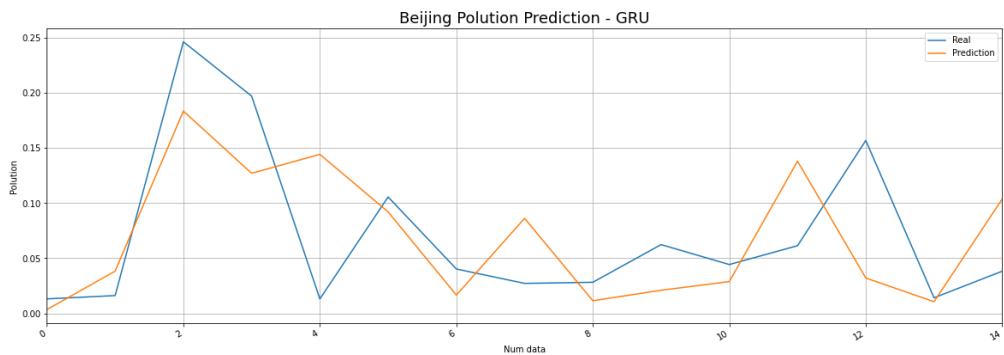
شکل 268: زمان اجرای آموزش در حالت مدل GRU و با MSE و Adam

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 269: نمودار Loss در حالت مدل GRU با MSE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 270: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU و Adam در حالت MSE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0010731142014265061
>>>-----*****-----<<<
>>>-----*****-----<<<
#####

```

شکل 271: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

► ۱۴- بررسی شبکه GRU در حالت MAE و Adam: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 272: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.019588156342506408 , Valid Loss : => 0.024042530357837676
***->>-----<<<-***  

step : 1 Train loss : 0.016114087477326393 , Valid Loss : => 0.014620605607827504
***->>-----<<<-***  

step : 2 Train loss : 0.014282376170158387 , Valid Loss : => 0.021374131242434185
***->>-----<<<-***  

step : 3 Train loss : 0.0148455161601305 , Valid Loss : => 0.01403677836060524
***->>-----<<<-***  

step : 4 Train loss : 0.014238129705190658 , Valid Loss : => 0.020608225464820863
***->>-----<<<-***  

step : 5 Train loss : 0.013994542807340622 , Valid Loss : => 0.01658615047732989
***->>-----<<<-***
```

شکل ۲۷۳: ایپاک اول مدل GRU

```

***->>-----<<<-***  

step : 193 Train loss : 0.004506983980536461 , Valid Loss : => 0.02186861584583918
***->>-----<<<-***  

step : 194 Train loss : 0.006029460933059454 , Valid Loss : => 0.02179307540257772
***->>-----<<<-***  

step : 195 Train loss : 0.004112636037170887 , Valid Loss : => 0.021655177076657612
***->>-----<<<-***  

step : 196 Train loss : 0.0036328443698585033 , Valid Loss : => 0.02236797958612442
***->>-----<<<-***  

step : 197 Train loss : 0.004739597793668508 , Valid Loss : => 0.026951263844966888
***->>-----<<<-***  

step : 198 Train loss : 0.004566189981997013 , Valid Loss : => 0.02354553689559301
***->>-----<<<-***  

step : 199 Train loss : 0.003657941613346338 , Valid Loss : => 0.021844636897246042
***->>-----<<<-***
```

شکل ۲۷۴: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

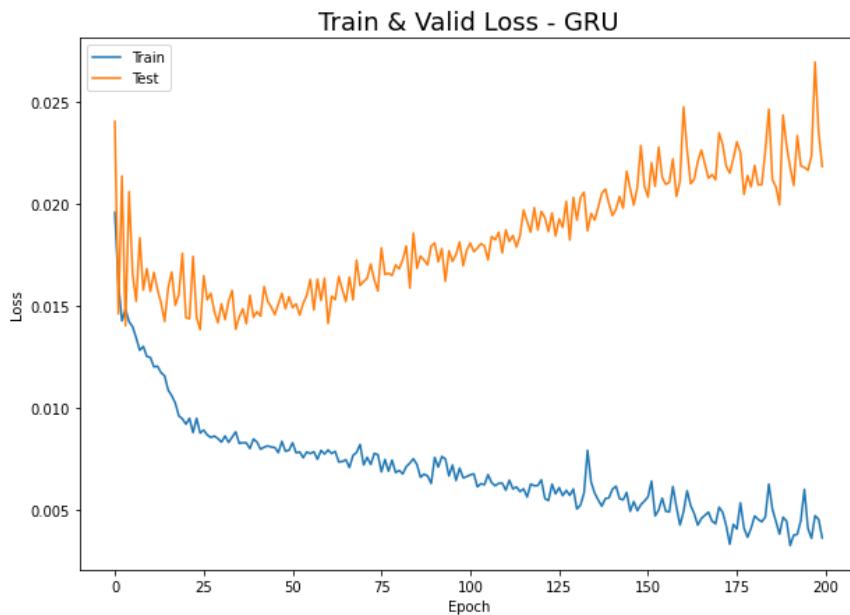
=====
*****  

The total Training Time is Equal with ==> : 9.067959070205688 Sec.
*****  

=====
```

شکل ۲۷۵: زمان اجرای آموزش در حالت مدل GRU و با MAE و Adam

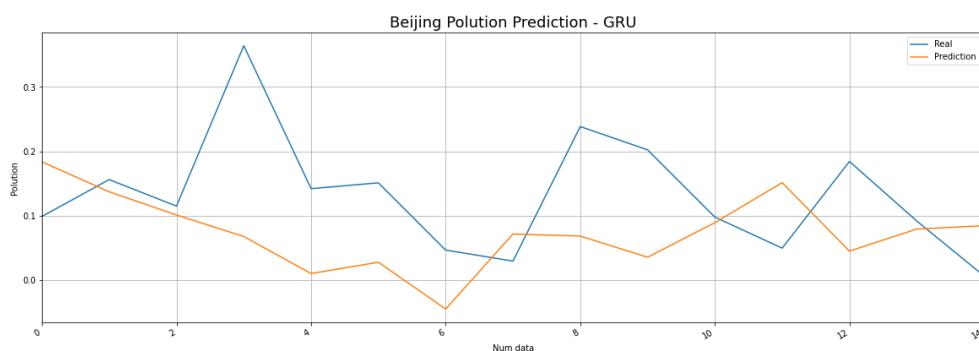
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 276: نمودار Loss در حالت مدل GRU با Adam و MAE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 277: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت Adam و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.023277106881141662
>>>-----*****-----<<<
>>>-----*****-----<<<
#####
```

شکل 278: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که

ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

► ۱۵- بررسی شبکه GRU در حالت MSE و AdaGrad: حالت ماهانه

ابتدا به تعریف Loss و تابع Optimizer می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)
```

شکل 279: تعریف مدل و Loss در مدل GRU و Optimizer

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.0019223870709538459 , Valid Loss : => 0.003715145929406087
***->>>-----<<<-***>
step : 1 Train loss : 0.001521001891233027 , Valid Loss : => 0.002792979124933481
***->>>-----<<<-***>
step : 2 Train loss : 0.0014880760014057159 , Valid Loss : => 0.002834400006880363
***->>>-----<<<-***>
step : 3 Train loss : 0.0014664105558767915 , Valid Loss : => 0.002868843001003067
***->>>-----<<<-***>
step : 4 Train loss : 0.0014500367874279618 , Valid Loss : => 0.002893775003030896
***->>>-----<<<-***>
step : 5 Train loss : 0.0014367671171203256 , Valid Loss : => 0.0029126636683940886
***->>>-----<<<-***>
```

شکل 280: ایپاک اول مدل GRU

```
***->>>-----<<<-***>
step : 193 Train loss : 0.000990842718165368 , Valid Loss : => 0.002945407945662737
***->>>-----<<<-***>
step : 194 Train loss : 0.0009894172241911292 , Valid Loss : => 0.0029445963290830454
***->>>-----<<<-***>
step : 195 Train loss : 0.0009879992273636163 , Valid Loss : => 0.0029437870097657045
***->>>-----<<<-***>
step : 196 Train loss : 0.0009865885321050883 , Valid Loss : => 0.002942979506527384
***->>>-----<<<-***>
step : 197 Train loss : 0.0009851851942948998 , Valid Loss : => 0.0029421741763750713
***->>>-----<<<-***>
step : 198 Train loss : 0.0009837890742346645 , Valid Loss : => 0.002941370972742637
***->>>-----<<<-***>
step : 199 Train loss : 0.0009824003954418004 , Valid Loss : => 0.0029405700353284676
***->>>-----<<<-***>
```

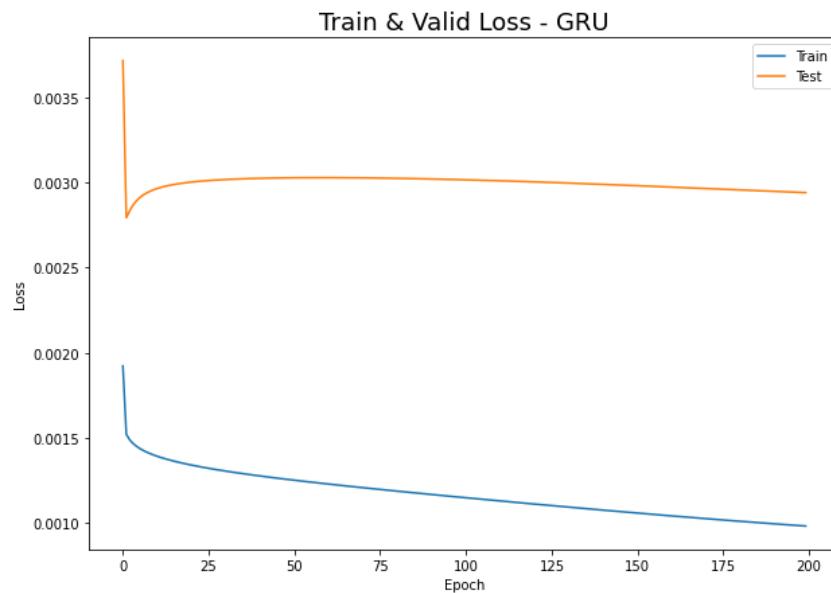
شکل 281: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 7.517634153366089 Sec.
*****
=====
```

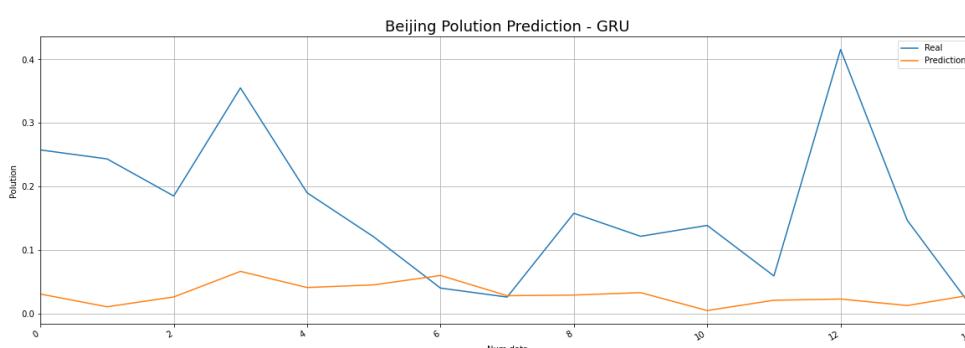
شکل 282: زمان اجرای آموزش در حالت مدل GRU و با MSE

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 283: نمودار Loss در حالت مدل GRU با MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 284: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت AdaGrad و MSE

```

#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0029393878765404226
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 285: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

این مدل در حالت Adam نسبت به Adagrad عملکرد بدتری دارد.

► 16- بررسی شبکه GRU در حالت MAE و AdaGrad: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)

```

شکل 286: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.013048278465867043 , Valid Loss : => 0.01176509956518809
***->>>-----<<<-***
step : 1 Train loss : 0.011267924644052983 , Valid Loss : => 0.011625002821286519
***->>>-----<<<-***
step : 2 Train loss : 0.010963444262742997 , Valid Loss : => 0.012001677850882212
***->>>-----<<<-***
step : 3 Train loss : 0.010729704461991786 , Valid Loss : => 0.011617215971151987
***->>>-----<<<-***
step : 4 Train loss : 0.010494877137243747 , Valid Loss : => 0.011901815980672836
***->>>-----<<<-***
step : 5 Train loss : 0.010376480109989643 , Valid Loss : => 0.011593007544676463
***->>>-----<<<-***

```

شکل 287: ایپاک اول مدل GRU

```

***_>>>-----<<<_***_
step : 193 Train loss : 0.0077898970432579515 , Valid Loss : => 0.013848469158013662
***_>>>-----<<<_***_
step : 194 Train loss : 0.007801560219377279 , Valid Loss : => 0.013751251995563507
***_>>>-----<<<_***_
step : 195 Train loss : 0.007785757593810558 , Valid Loss : => 0.013754665851593018
***_>>>-----<<<_***_
step : 196 Train loss : 0.007780355997383594 , Valid Loss : => 0.013752870013316472
***_>>>-----<<<_***_
step : 197 Train loss : 0.007788361050188541 , Valid Loss : => 0.013745635251204173
***_>>>-----<<<_***_
step : 198 Train loss : 0.0078051406517624855 , Valid Loss : => 0.013836531341075898
***_>>>-----<<<_***_
step : 199 Train loss : 0.007822534069418907 , Valid Loss : => 0.013842471440633138
***_>>>-----<<<_***_

```

شکل 288: ایپاک آخر مدل GRU

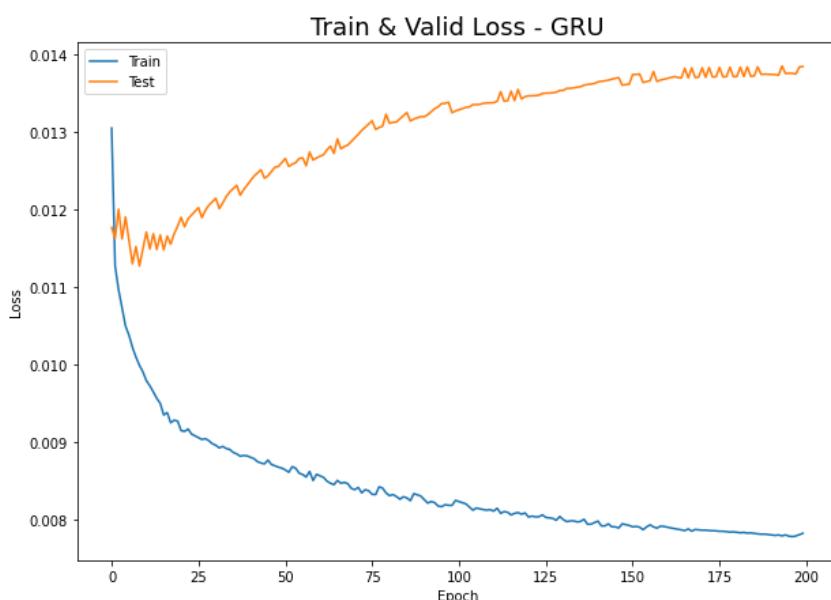
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 8.1457200050354 Sec.
*****
=====
```

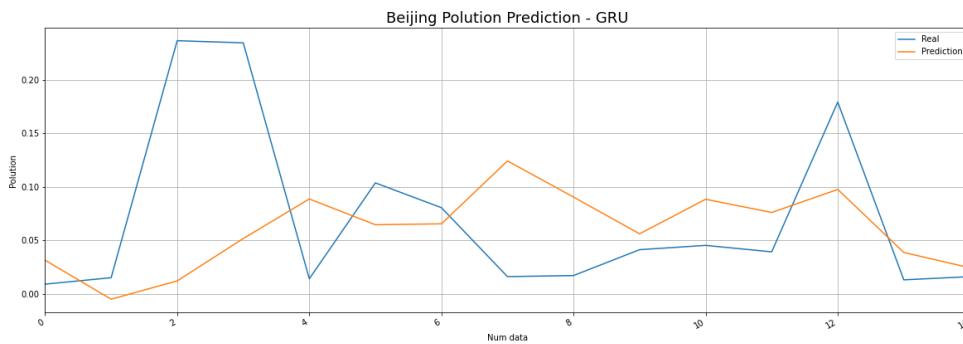
شکل 289: زمان اجرای آموزش در حالت مدل GRU و با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 290: نمودار Loss در حالت مدل GRU با MAE و Adagrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 291: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت AdaGrad و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.013842413822809855
>>>-----*****-----<<<
>>>-----*****-----<<<
#####

```

شکل 292: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بدتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

این مدل در حالت Adagrad نسبت به Adam عملکرد بدتری دارد.

► 17- بررسی شبکه GRU در حالت MSE و RMSprop: حالت ماهانه

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
```

شکل 293: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.01250736155663617 , Valid Loss : => 0.0006105029761480789
***->>>-----<<<-***  
step : 1 Train loss : 0.0010996185045223682 , Valid Loss : => 0.0005922800356832643
***->>>-----<<<-***  
step : 2 Train loss : 0.0010654387157410384 , Valid Loss : => 0.0005959787600052853
***->>>-----<<<-***  
step : 3 Train loss : 0.0010349584417417646 , Valid Loss : => 0.0006016867196497818
***->>>-----<<<-***  
step : 4 Train loss : 0.0010065036994637922 , Valid Loss : => 0.0006091835132489602
***->>>-----<<<-***  
step : 5 Train loss : 0.000979468297737185 , Valid Loss : => 0.0006183611927554012
***->>>-----<<<-***
```

شکل 294: ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 193 Train loss : 2.450401547321235e-05 , Valid Loss : => 0.0011197992600500584
***->>>-----<<<-***  
step : 194 Train loss : 2.3075032659107818e-05 , Valid Loss : => 0.0011413903286059698
***->>>-----<<<-***  
step : 195 Train loss : 2.021878073719563e-05 , Valid Loss : => 0.0011334729691346486
***->>>-----<<<-***  
step : 196 Train loss : 1.8045376891677732e-05 , Valid Loss : => 0.0011530995679398378
***->>>-----<<<-***  
step : 197 Train loss : 2.0413934926182265e-05 , Valid Loss : => 0.0011045077194770177
***->>>-----<<<-***  
step : 198 Train loss : 2.9330746256164274e-05 , Valid Loss : => 0.0014978666479388873
***->>>-----<<<-***  
step : 199 Train loss : 0.0001250496595457662 , Valid Loss : => 0.0008936316395799319
***->>>-----<<<-***
```

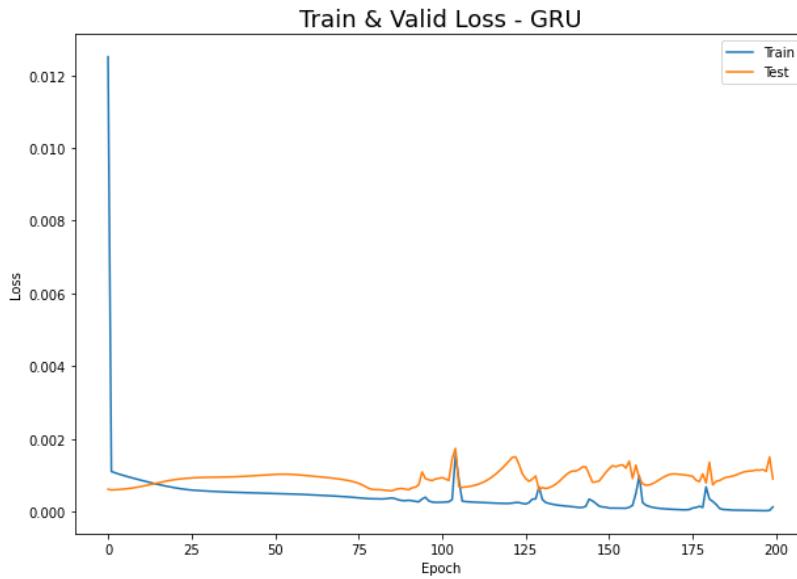
شکل 295: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 7.5065758228302 Sec.  
*****  
=====
```

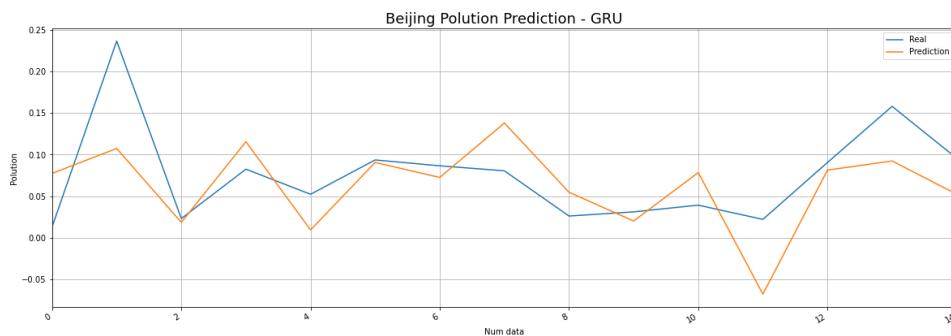
شکل 296: زمان اجرای آموزش در حالت مدل GRU و RMSprop و MSE

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 297: نمودار Loss در حالت مدل GRU با MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 298: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت AdaGrad و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0007934756887455781
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 299: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت

به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

این مدل در حالت RMSprop عملکرد بهتری از Adam داشت.

► 18- بررسی شبکه GRU در حالت MAE و RMSprop

ابتدا به تعریف Loss و تابع Optimizer می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
```

شکل 300: تعریف مدل و Loss در مدل Optimizer

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.03140339955687523 , Valid Loss : => 0.01059296987950802
***->>>-----<<<-***  
step : 1 Train loss : 0.012180081829428672 , Valid Loss : => 0.010258256271481515
***->>>-----<<<-***  
step : 2 Train loss : 0.011524131931364536 , Valid Loss : => 0.010563452790180842
***->>>-----<<<-***  
step : 3 Train loss : 0.01124218661338091 , Valid Loss : => 0.009831580768028895
***->>>-----<<<-***  
step : 4 Train loss : 0.01067491427063942 , Valid Loss : => 0.010048153251409531
***->>>-----<<<-***  
step : 5 Train loss : 0.010432233586907386 , Valid Loss : => 0.009968983381986618
***->>>-----<<<-***
```

شکل 301: ابیاک اول مدل GRU

```
***->>>-----<<<-***  
step : 193 Train loss : 0.005000329818576574 , Valid Loss : => 0.015425625443458556
***->>>-----<<<-***  
step : 194 Train loss : 0.0043570675142109395 , Valid Loss : => 0.014447487394014994
***->>>-----<<<-***  
step : 195 Train loss : 0.0047614420391619205 , Valid Loss : => 0.015581753353277843
***->>>-----<<<-***  
step : 196 Train loss : 0.0048350486718118195 , Valid Loss : => 0.014028463264306386
***->>>-----<<<-***  
step : 197 Train loss : 0.005144393239170313 , Valid Loss : => 0.014212998747825622
***->>>-----<<<-***  
step : 198 Train loss : 0.004511733036488295 , Valid Loss : => 0.013347369184096654
***->>>-----<<<-***  
step : 199 Train loss : 0.003926908187568187 , Valid Loss : => 0.015087953706582388
***->>>-----<<<-***
```

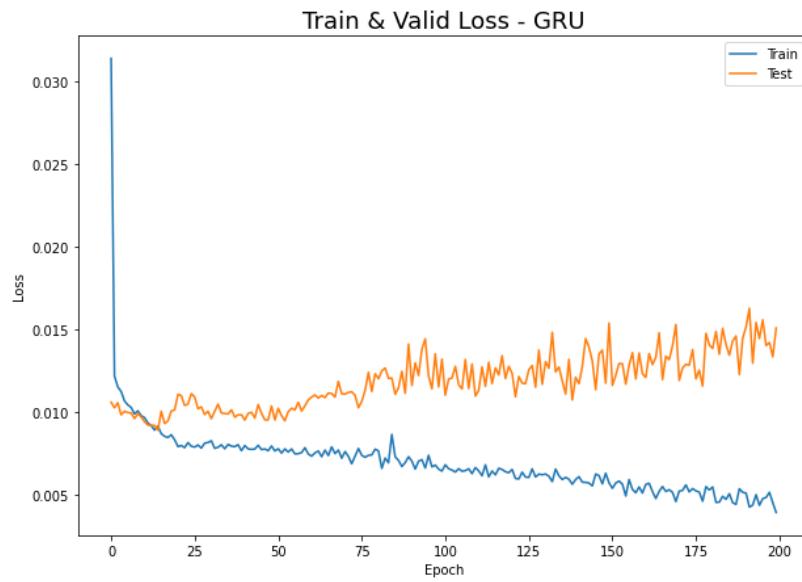
شکل 302: ابیاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
***** The total Training Time is Equal with ==> : 7.5070106983184814 Sec. ****
=====
```

شکل 303: زمان اجرای آموزش در حالت مدل GRU و با MAE و RMSprop

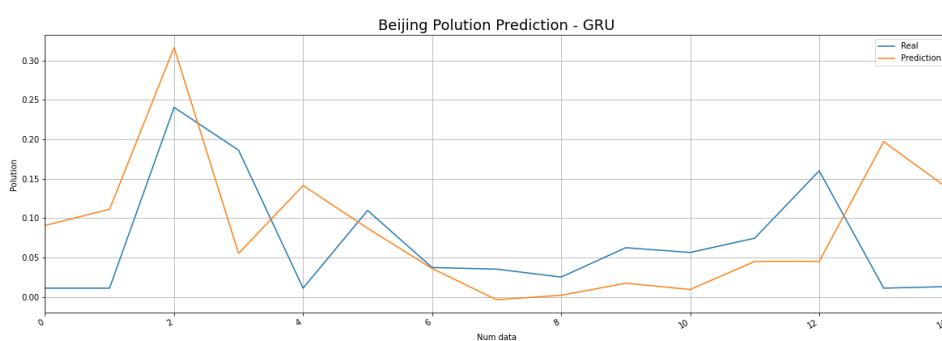
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 304: نمودار Loss در حالت مدل GRU با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 305: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MAE و AdaGrad

```

#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.01439053863286972
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 306: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که خوب شبکه عملکرد خیلی خوبی ندارد و یکی از علت‌های اصلی آن نیز کم بودن داده هست. و این که ما داده کافی برای آموزش کامل و خوب شبکه را در اختیار نداریم. اما در این حالت عملکرد شبکه نسبت به حالت قبلی که RNN بود عملکرد بهتری دارد و علاوه بر این که روند کلی شبکه را نشان داده توانسته است که کمی دقیق‌تر این کار را انجام دهد.

همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

این مدل در حالت RMSprop عملکرد بهتری از Adam داشت.

حالات هفتگی:

در این حالت **Batch_Size=10** هست. همچنین **Epoch=200** هست.
در این حالت در مجموع **18** حالت را باید بررسی کنم که در ادامه آن‌ها را قرار می‌دهم.

۱- بررسی شبکه RNN در حالت Adam و MSE: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

```

شکل 307: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می پردازیم:

```
step : 0 Train loss : 0.0007520293700508773 , Valid Loss : => 0.0009591940790414811
***->>>-----<<<-***  
step : 1 Train loss : 0.0005628790589980781 , Valid Loss : => 0.0005514817545190454
***->>>-----<<<-***  
step : 2 Train loss : 0.0005438408282186304 , Valid Loss : => 0.0006391296489164233
***->>>-----<<<-***  
step : 3 Train loss : 0.0004988538955027858 , Valid Loss : => 0.000654854727908969
***->>>-----<<<-***  
step : 4 Train loss : 0.0004798056640928345 , Valid Loss : => 0.0006682850047945977
***->>>-----<<<-***  
step : 5 Train loss : 0.0004664356349080446 , Valid Loss : => 0.0006777792284265161
***->>>-----<<<-***
```

شکل 308: آنچه اول مدل RNN

```
***->>>-----<<<-***  
step : 193 Train loss : 0.0004446094489789435 , Valid Loss : => 0.0005521920695900917
***->>>-----<<<-***  
step : 194 Train loss : 0.0004361716631267752 , Valid Loss : => 0.0005539805721491575
***->>>-----<<<-***  
step : 195 Train loss : 0.0004343099282344892 , Valid Loss : => 0.0005596303706988693
***->>>-----<<<-***  
step : 196 Train loss : 0.00043324475908385854 , Valid Loss : => 0.0005588962975889444
***->>>-----<<<-***  
step : 197 Train loss : 0.00043310670409395935 , Valid Loss : => 0.0005553634976968169
***->>>-----<<<-***  
step : 198 Train loss : 0.00042388293548442776 , Valid Loss : => 0.0005631602508947253
***->>>-----<<<-***  
step : 199 Train loss : 0.0004185719676094041 , Valid Loss : => 0.0005594020616263151
***->>>-----<<<-***
```

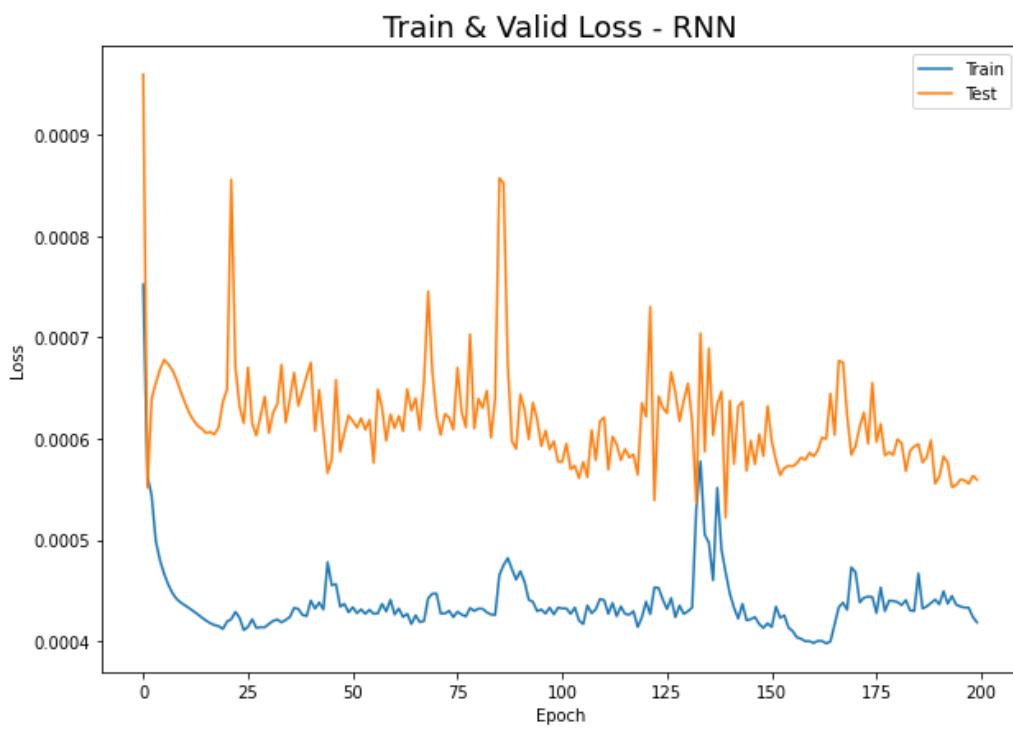
شکل 309: آنچه آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 11.916874885559082 Sec.  
*****  
=====
```

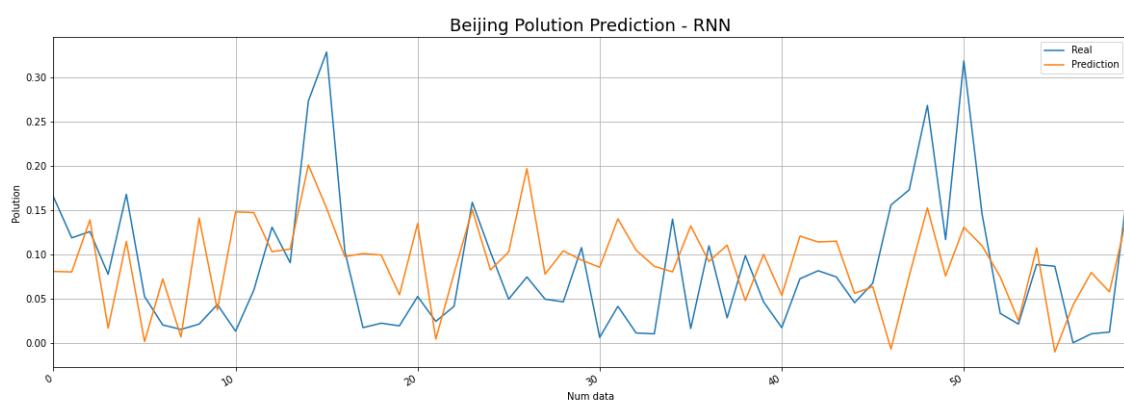
شکل 310: زمان اجرای آموزش در حالت مدل RNN و با Adam و MSE

سپس در ادامه می توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 311: نمودار Loss در حالت مدل RNN با MSE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 312: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و Adam

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 0.00042751741795135395
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 313: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که عملکرد این مدل در حالت هفتگی از ماهانه کمی بهتر است و علت آن نیز بیشتر شدن دیتاهای ما هست اما خب همچنان دقیق قابل قبولی ندارد اما خب نتایج بهتر شده است.

► 2- بررسی شبکه RNN در حالت MAE و Adam: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 314: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.007959505099625815 , Valid Loss : => 0.006796197146177292
***->>>-----<<<-***  

step : 1 Train loss : 0.006984745977180345 , Valid Loss : => 0.006634955704212189
***->>>-----<<<-***  

step : 2 Train loss : 0.00649917269391673 , Valid Loss : => 0.006387952342629432
***->>>-----<<<-***  

step : 3 Train loss : 0.006297951156184787 , Valid Loss : => 0.00619570828974247
***->>>-----<<<-***  

step : 4 Train loss : 0.00630930255921114 , Valid Loss : => 0.006287764087319374
***->>>-----<<<-***  

step : 5 Train loss : 0.0062614216868366514 , Valid Loss : => 0.006246946081519127
***->>>-----<<<-***
```

شکل 315: ایپاک اول مدل RNN

```
***->>>-----<<<-***  

step : 193 Train loss : 0.005506046515490327 , Valid Loss : => 0.006675793826580048
***->>>-----<<<-***  

step : 194 Train loss : 0.005583058146848565 , Valid Loss : => 0.006383706107735634
***->>>-----<<<-***  

step : 195 Train loss : 0.005592631655079978 , Valid Loss : => 0.0065853364765644075
***->>>-----<<<-***  

step : 196 Train loss : 0.00557140013469117 , Valid Loss : => 0.006444305256009102
***->>>-----<<<-***  

step : 197 Train loss : 0.0055662364033716065 , Valid Loss : => 0.00652654156088829
***->>>-----<<<-***  

step : 198 Train loss : 0.005558530223511514 , Valid Loss : => 0.006771872341632843
***->>>-----<<<-***  

step : 199 Train loss : 0.0055742977630524404 , Valid Loss : => 0.006727525517344474
***->>>-----<<<-***
```

شکل 316: ایپاک آخر مدل RNN

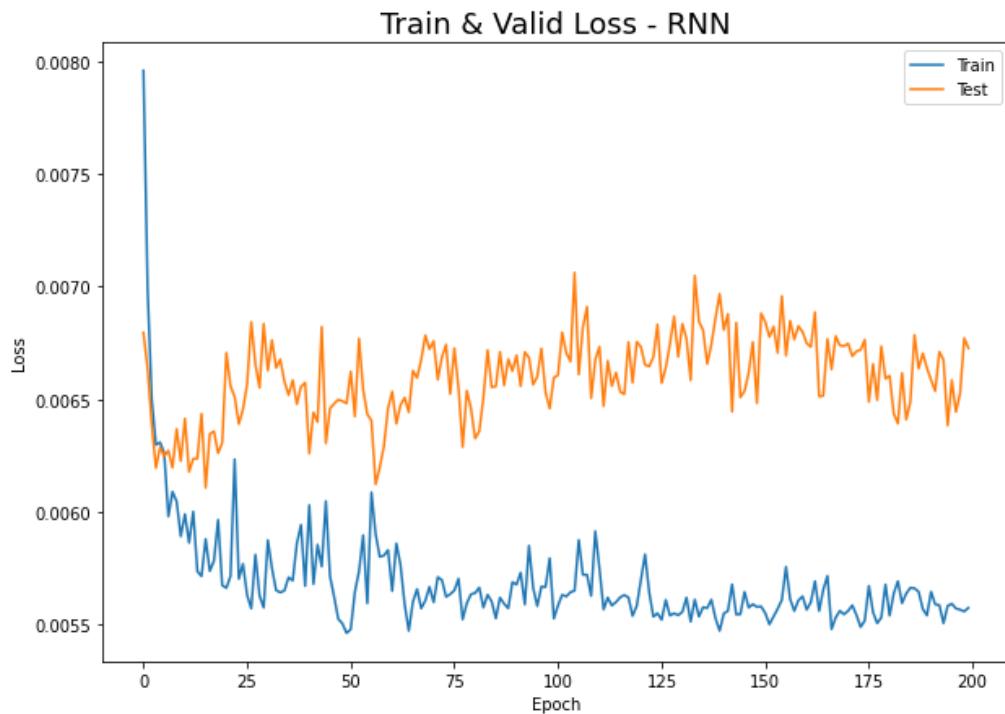
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
=====
The total Training Time is Equal with ==> : 11.899601697921753 Sec.
=====
=====
```

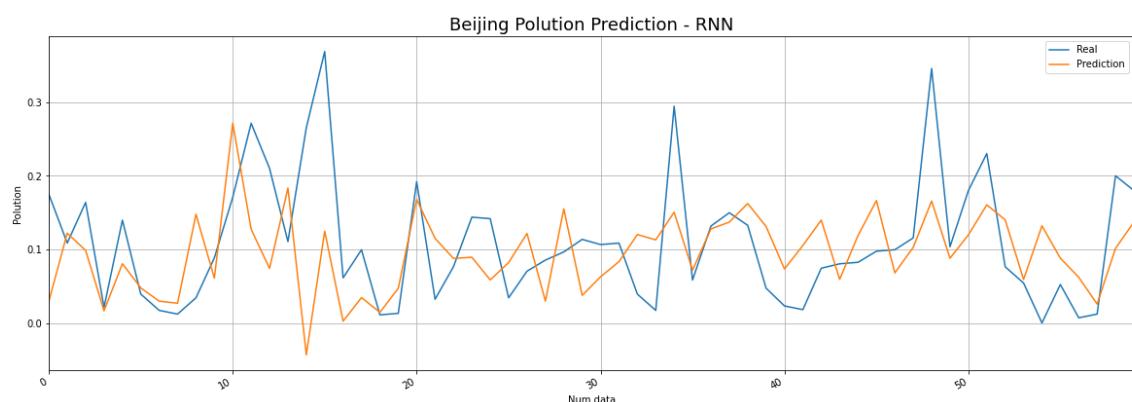
شکل 317: زمان اجرای آموزش در حالت مدل RNN و با MAE و Adam

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 318: نمودار Loss در حالت مدل RNN با MAE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 319: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MAE و Adam

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.005635856446765718
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 320: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که عملکرد این مدل در حالت هفتگی از ماهانه کمی بهتر است و علت آن نیز بیشتر شدن دیتاهای ما هست اما خب همچنان دقیق قابل قبولی ندارد اما خب نتایج بهتر شده است.

► 3- بررسی شبکه RNN در حالت MSE و ADAGrad

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=3, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)
```

شکل 321: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.0010606285816590702 , Valid Loss : => 0.0009409535024315119
***->>-----<<<-
step : 1 Train loss : 0.0008123834672871799 , Valid Loss : => 0.0008737457543611526
***->>-----<<<-
step : 2 Train loss : 0.000768155176636009 , Valid Loss : => 0.0008263641269877553
***->>-----<<<-
step : 3 Train loss : 0.0007454140494311494 , Valid Loss : => 0.0007995697483420372
***->>-----<<<-
step : 4 Train loss : 0.000731519190594554 , Valid Loss : => 0.0007831043284386396
***->>-----<<<-
step : 5 Train loss : 0.0007214890183171346 , Valid Loss : => 0.0007721257070079446
***->>-----<<<-
```

شکل 322: ایپاک اول مدل RNN

```

***->>>-----<<<-***  

step : 193 Train loss : 0.0004971705898199053 , Valid Loss : => 0.0007511678524315357  

***->>>-----<<<-***  

step : 194 Train loss : 0.000496915222278663 , Valid Loss : => 0.000751222250983119  

***->>>-----<<<-***  

step : 195 Train loss : 0.0004966617949927846 , Valid Loss : => 0.0007512763608247041  

***->>>-----<<<-***  

step : 196 Train loss : 0.0004964103351258451 , Valid Loss : => 0.0007513301679864526  

***->>>-----<<<-***  

step : 197 Train loss : 0.0004961607861332596 , Valid Loss : => 0.0007513838447630406  

***->>>-----<<<-***  

step : 198 Train loss : 0.0004959131979073088 , Valid Loss : => 0.0007514371583238244  

***->>>-----<<<-***  

step : 199 Train loss : 0.000495667464011127 , Valid Loss : => 0.0007514902111142874  

***->>>-----<<<-***
```

شکل 323: آییاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

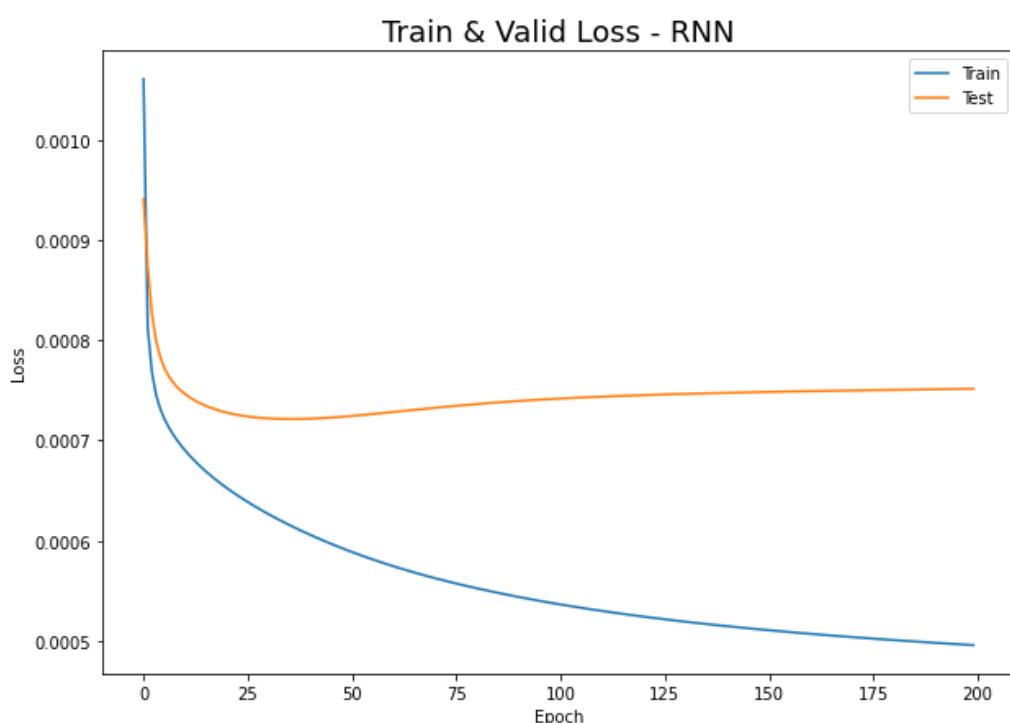
The total Training Time is Equal with ==> : 10.705456256866455 Sec.  

*****  

=====
```

شکل 324: زمان اجرای آموزش در حالت مدل RNN و با AdaGrad و MSE

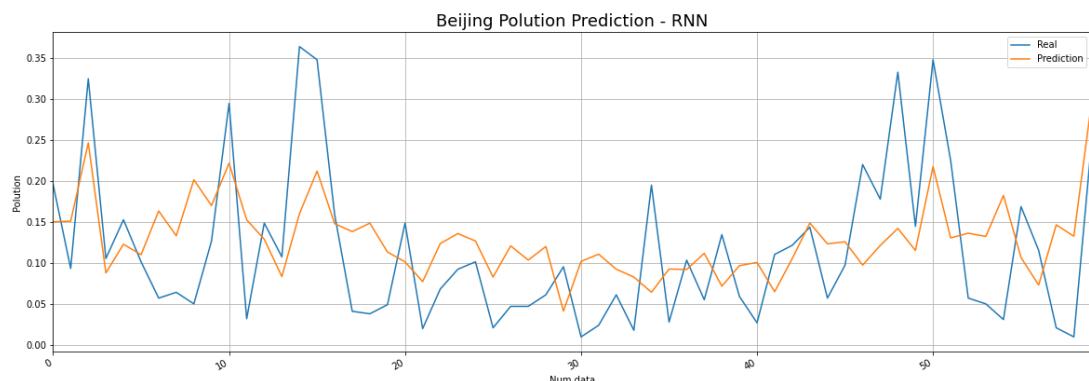
سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 325: نمودار Loss در حالت مدل RNN با AdaGrad و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد)

مشاهده نمایید:



شکل 326: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و AdaGrad

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0005695931934973314
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 327: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که عملکرد این مدل در حالت هفتگی از ماهانه کمی بهتر است و علت آن نیز بیشتر شدن دیتاهای ما هست اما خب همچنان دقیق قبولی ندارد اما خب نتایج بهتر شده است.
همچنین در این حالت نسبت به حالت Adam عملکرد مدل ما بدتر شده است.

► 4- بررسی شبکه RNN در حالت MAE و ADAGrad: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)
```

شکل 328: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.00851184759466421 , Valid Loss : => 0.008075930327177048
***->>-----<<<-***  
step : 1 Train loss : 0.007566668714086215 , Valid Loss : => 0.007793698608875275
***->>-----<<<-***  
step : 2 Train loss : 0.007301731460860797 , Valid Loss : => 0.007730736210942268
***->>-----<<<-***  
step : 3 Train loss : 0.007124747282692364 , Valid Loss : => 0.007727203592658043
***->>-----<<<-***  
step : 4 Train loss : 0.007080733900268873 , Valid Loss : => 0.007718616649508476
***->>-----<<<-***  
step : 5 Train loss : 0.006966002108085723 , Valid Loss : => 0.007675807476043701
***->>-----<<<-***
```

شکل ۵.329 ایپاک اول مدل RNN

```
***->>-----<<<-***  
step : 193 Train loss : 0.0056897819751784914 , Valid Loss : => 0.007999802678823472
***->>-----<<<-***  
step : 194 Train loss : 0.00568779202266818 , Valid Loss : => 0.008005304634571076
***->>-----<<<-***  
step : 195 Train loss : 0.005677839173447518 , Valid Loss : => 0.008015481680631638
***->>-----<<<-***  
step : 196 Train loss : 0.005684882641902991 , Valid Loss : => 0.007979395240545273
***->>-----<<<-***  
step : 197 Train loss : 0.00568526295856351 , Valid Loss : => 0.007995158731937409
***->>-----<<<-***  
step : 198 Train loss : 0.005679166485511121 , Valid Loss : => 0.008006001859903335
***->>-----<<<-***  
step : 199 Train loss : 0.005666101777127811 , Valid Loss : => 0.008012739717960357
***->>-----<<<-***
```

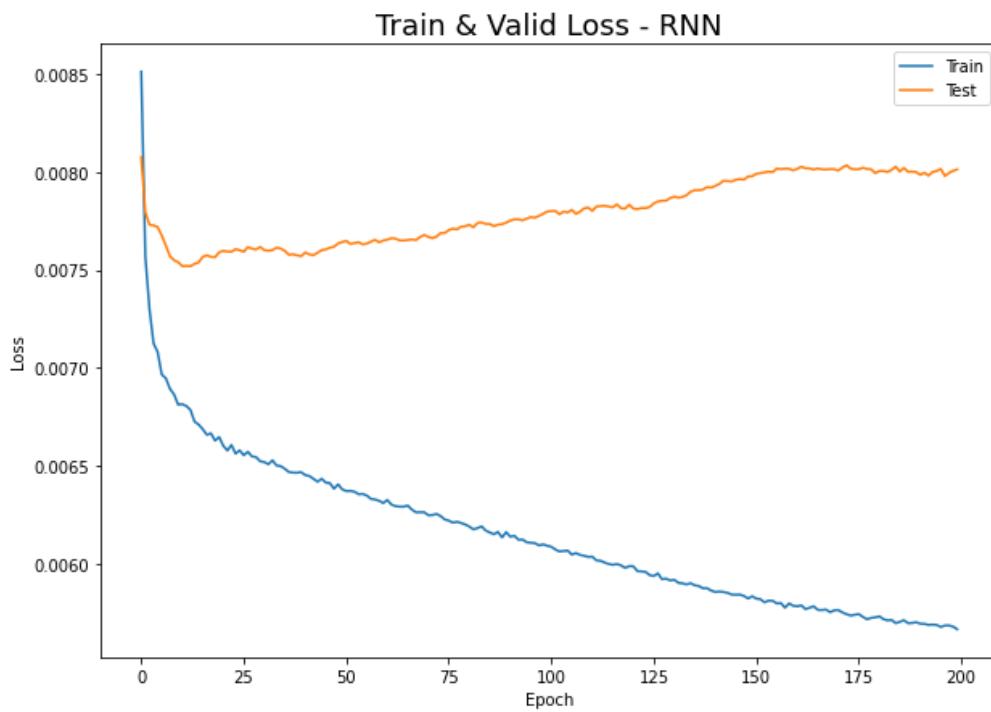
شکل 5.330 ایپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 10.952620029449463 Sec.  
*****  
=====
```

شکل 331: زمان اجرای آموزش در حالت مدل RNN و با MAE و AdaGrad

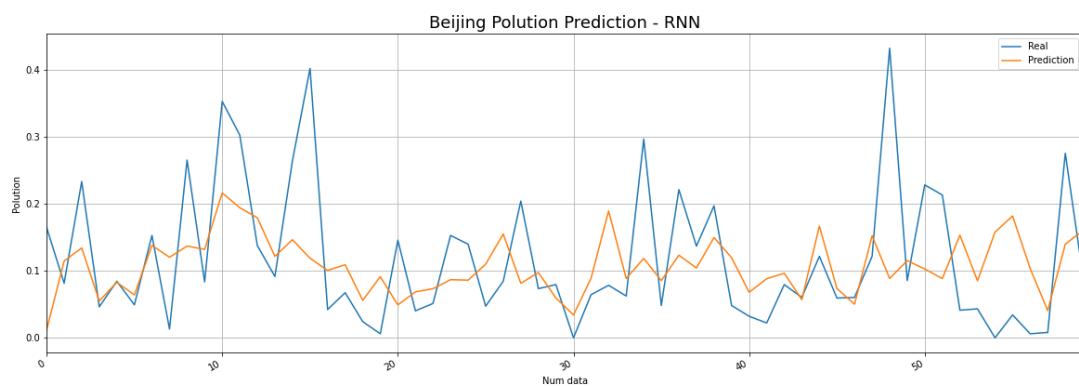
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 332: نمودار Loss در حالت مدل RNN با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد)

مشاهده نمایید:



شکل 333: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MAE و AdaGrad

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 0.006157001861858935
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 334: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که عملکرد این مدل در حالت هفتگی از ماهانه کمی بهتر است و علت آن نیز بیشتر شدن دیتاهای ما هست اما خب همچنان دقیق قبولی ندارد اما خب نتایج بهتر شده است.

همچنین در این حالت نسبت به حالت Adam عملکرد مدل ما بدتر شده است.

► 5- بررسی شبکه RNN در حالت MSE و RMSprop: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)

```

شکل 335: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.42293899882407415 , Valid Loss : => 0.1767342746257782
***->>-----<<<-*** 
step : 1 Train loss : 0.017086519859731197 , Valid Loss : => 0.1662548291683197
***->>-----<<<-*** 
step : 2 Train loss : 0.1475309899875096 , Valid Loss : => 0.1383575689792633
***->>-----<<<-*** 
step : 3 Train loss : 0.03732739882100196 , Valid Loss : => 0.003862455878406763
***->>-----<<<-*** 
step : 4 Train loss : 0.002377420362262499 , Valid Loss : => 0.0012833349965512752
***->>-----<<<-*** 
step : 5 Train loss : 0.009163000394723245 , Valid Loss : => 0.04274552345275879
***->>-----<<<-*** 

```

شکل 336: ایپاک اول مدل RNN

```

***->>-----<<<-*** 
step : 193 Train loss : 0.0007443728873373143 , Valid Loss : => 0.0013338189478963613
***->>-----<<<-*** 
step : 194 Train loss : 0.0007526220299214835 , Valid Loss : => 0.0013504767511039973
***->>-----<<<-*** 
step : 195 Train loss : 0.0007573940153677194 , Valid Loss : => 0.0013911513611674308
***->>-----<<<-*** 
step : 196 Train loss : 0.0007778361938627703 , Valid Loss : => 0.0014563883468508721
***->>-----<<<-*** 
step : 197 Train loss : 0.0008135641803077999 , Valid Loss : => 0.0013328457064926624
***->>-----<<<-*** 
step : 198 Train loss : 0.0007508538275336226 , Valid Loss : => 0.0012398921325802803
***->>-----<<<-*** 
step : 199 Train loss : 0.0007204356603324414 , Valid Loss : => 0.0012464567460119724
***->>-----<<<-*** 

```

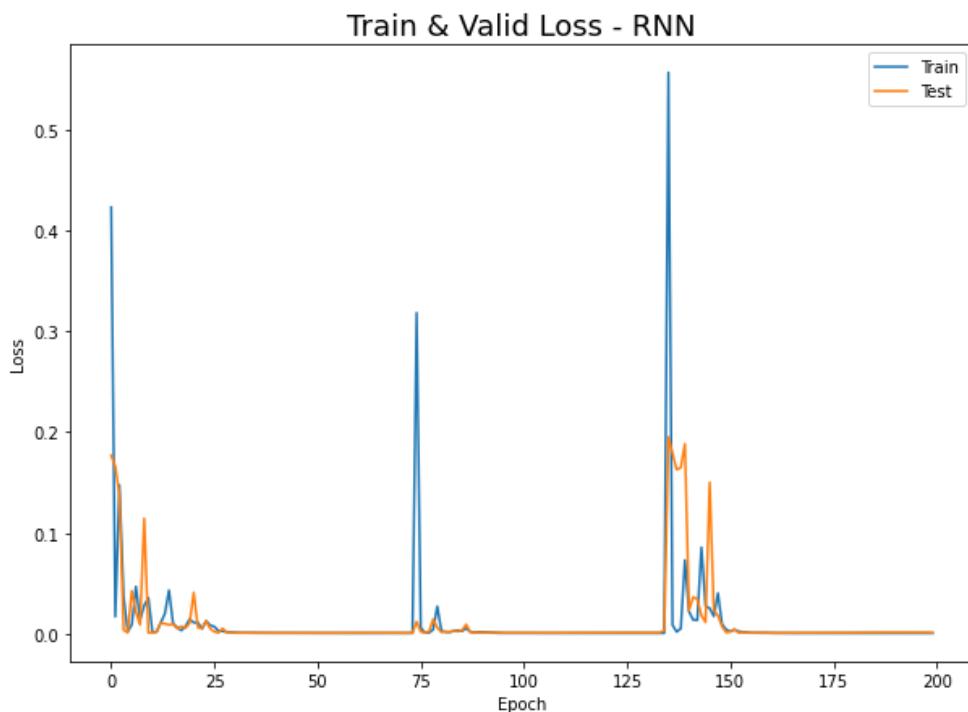
شکل 337: ایپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 10.242300510406494 Sec.
*****
=====
```

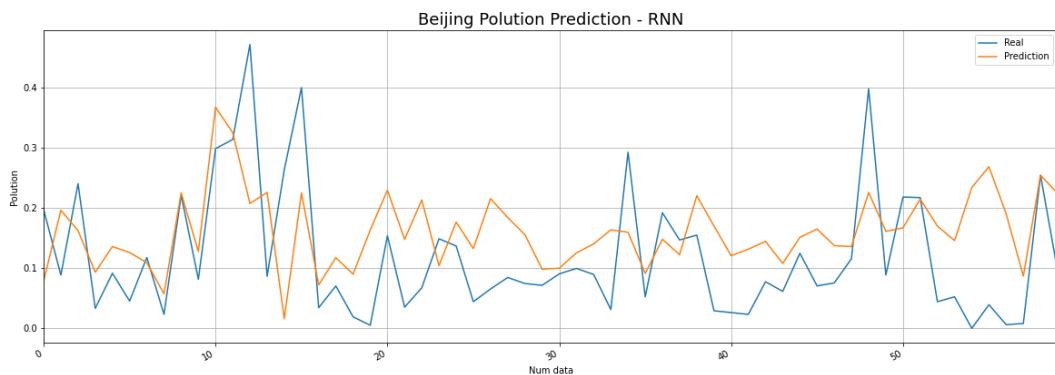
شکل 338: زمان اجرای آموزش در حالت مدل RNN و با MSE و RMSprop

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 339: نمودار Loss در حالت مدل RNN با MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 340: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0008121379961570104
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 341: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که عملکرد این مدل در حالت هفتگی از ماهانه کمی بهتر است و علت آن نیز بیشتر شدن دیتاهای ما هست اما خب همچنان دقیق قابل قبولی ندارد اما خب نتایج بهتر شده است.

همچنین در این حالت نسبت به حالت Adam عملکرد مدل ما بدتر شده است. اما نسبت به حالت AdaGrad عملکرد بهتری داریم.

► 6- بررسی شبکه RNN در حالت هفتگی و MAE و RMSprop: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
```

شکل 342: تعریف مدل و Optimizer وتابع Loss در مدل RNN

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.37466458117678053 , Valid Loss : => 0.9808194303512573
***->>-----<<<-***  

step : 1 Train loss : 0.17611170496259418 , Valid Loss : => 0.9066828155517578
***->>-----<<<-***  

step : 2 Train loss : 0.1353177757490249 , Valid Loss : => 0.8512255668640136
***->>-----<<<-***  

step : 3 Train loss : 0.17067112947503726 , Valid Loss : => 0.0956670606136322
***->>-----<<<-***  

step : 4 Train loss : 0.14259279576085862 , Valid Loss : => 0.16137797832489015
***->>-----<<<-***  

step : 5 Train loss : 0.12987794384715104 , Valid Loss : => 0.057552874088287354
***->>-----<<<-***
```

شکل 343: ایپاک اول مدل RNN

```

***->>-----<<<-***  

step : 193 Train loss : 0.010564996648047652 , Valid Loss : => 0.01196239836513996
***->>-----<<<-***  

step : 194 Train loss : 0.010209138735774018 , Valid Loss : => 0.01162354052066803
***->>-----<<<-***  

step : 195 Train loss : 0.010382866948133423 , Valid Loss : => 0.012179526016116143
***->>-----<<<-***  

step : 196 Train loss : 0.010229289567186719 , Valid Loss : => 0.011693881452083587
***->>-----<<<-***  

step : 197 Train loss : 0.010360622610009852 , Valid Loss : => 0.01207060471177101
***->>-----<<<-***  

step : 198 Train loss : 0.010251665022224188 , Valid Loss : => 0.011859092116355895
***->>-----<<<-***  

step : 199 Train loss : 0.010234564915299416 , Valid Loss : => 0.012986558079719544
***->>-----<<<-***
```

شکل 344: ایپاک آخر مدل RNN

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

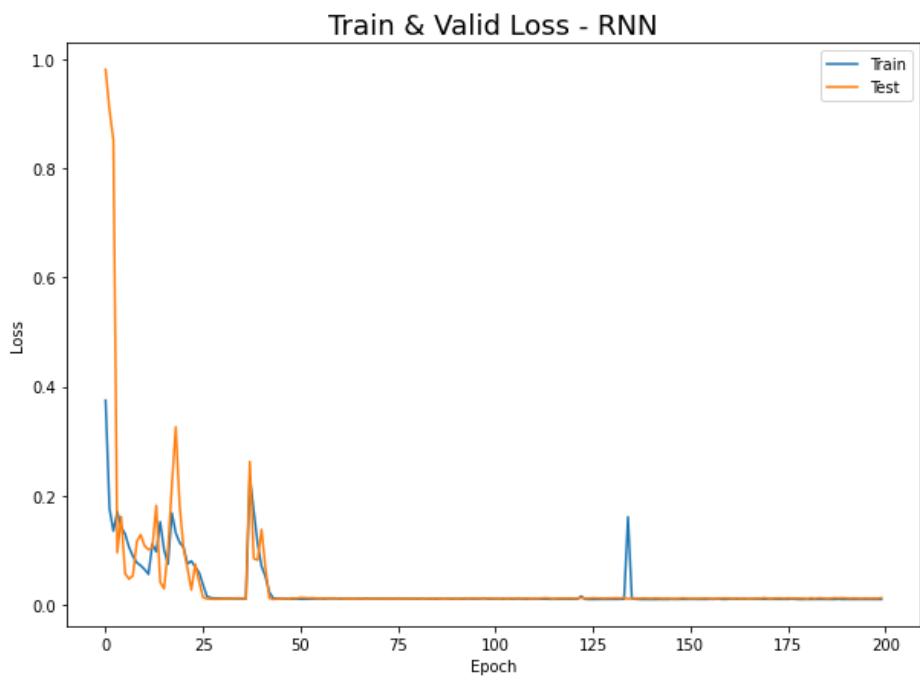
=====
*****  

The total Training Time is Equal with ==> : 18.308834552764893 Sec.
*****  

=====
```

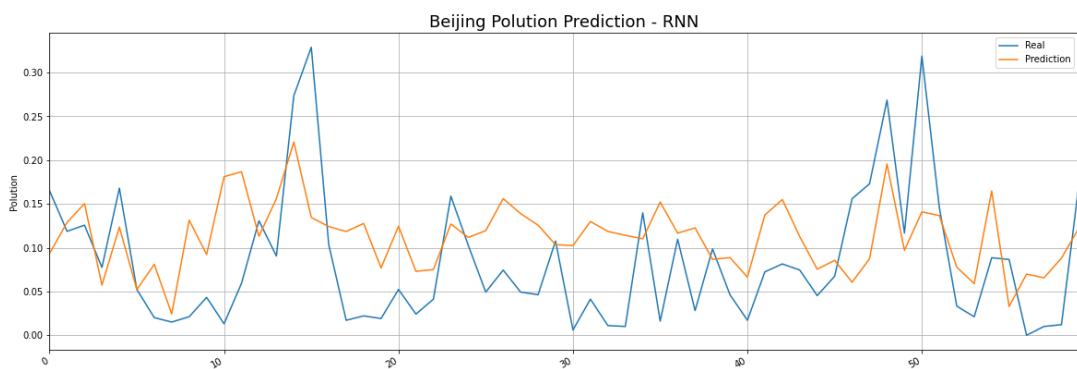
شکل 345: زمان اجرای آموزش در حالت مدل RNN و با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 346: نمودار Loss در حالت مدل RNN با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 347: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MAE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
*** Test Loss :==>> 0.010924649522418067
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 348: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که عملکرد این مدل در حالت هفتگی از ماهانه کمی بهتر است و علت آن نیز بیشتر شدن دیتاهاست ما هست اما خب همچنان دقیق قبولی ندارد اما خب نتایج بهتر شده است.

همچنین در این حالت نسبت به حالت Adam عملکرد مدل ما بدتر شده است. اما نسبت به حالت AdaGrad عملکرد بهتری داریم.

► 7- بررسی شبکه LSTM در حالت MSE و Adam: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 349: تعریف مدل و Optimizer وتابع Loss در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.009618360749445855 , Valid Loss : => 0.002201768378727138
***->>>-----<<<-***  

step : 1 Train loss : 0.008714244002476335 , Valid Loss : => 0.002156115318648517
***->>>-----<<<-***  

step : 2 Train loss : 0.008537898915819823 , Valid Loss : => 0.002132171988487244
***->>>-----<<<-***  

step : 3 Train loss : 0.008418496660888196 , Valid Loss : => 0.002114263349212706
***->>>-----<<<-***  

step : 4 Train loss : 0.008312262995168566 , Valid Loss : => 0.002098443559370935
***->>>-----<<<-***  

step : 5 Train loss : 0.00821354845073074 , Valid Loss : => 0.0020834537385962903
***->>>-----<<<-***
```

شکل 350: ایپاک اول مدل LSTM

```
***->>>-----<<<-***  

step : 193 Train loss : 0.0014104947372106835 , Valid Loss : => 0.002499753530137241
***->>>-----<<<-***  

step : 194 Train loss : 0.0013995699310908093 , Valid Loss : => 0.0024935583118349315
***->>>-----<<<-***  

step : 195 Train loss : 0.0013756244597607293 , Valid Loss : => 0.0025343707529827952
***->>>-----<<<-***  

step : 196 Train loss : 0.0013869076954870251 , Valid Loss : => 0.002526629492640495
***->>>-----<<<-***  

step : 197 Train loss : 0.0013849012526043225 , Valid Loss : => 0.0025666171079501508
***->>>-----<<<-***  

step : 198 Train loss : 0.0013427216235140805 , Valid Loss : => 0.0025467436714097856
***->>>-----<<<-***  

step : 199 Train loss : 0.0013282379174779635 , Valid Loss : => 0.0026324358070269225
***->>>-----<<<-***
```

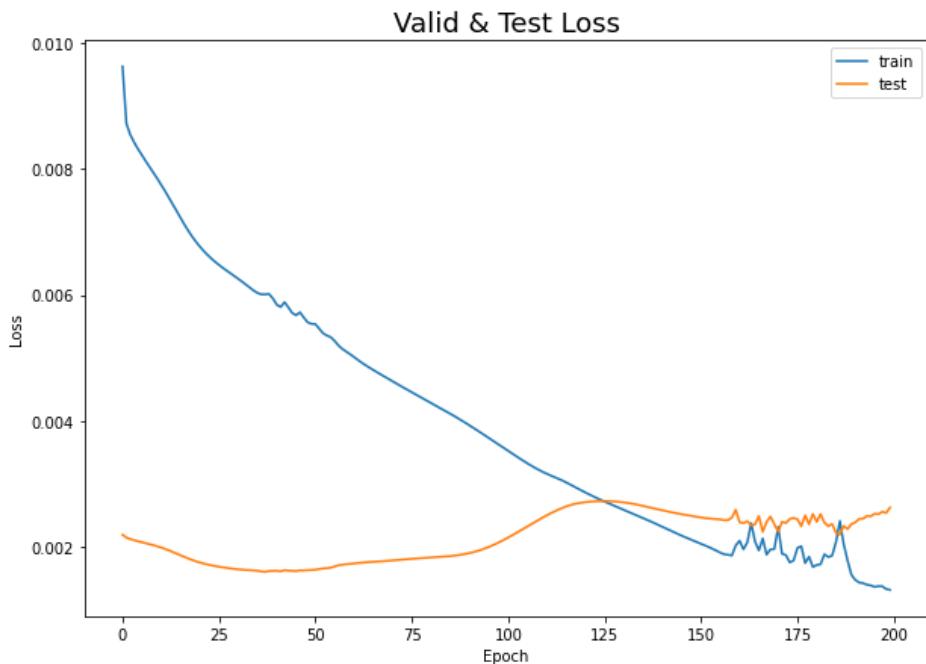
شکل 351: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 62.00759506225586 Sec.  
*****  
=====
```

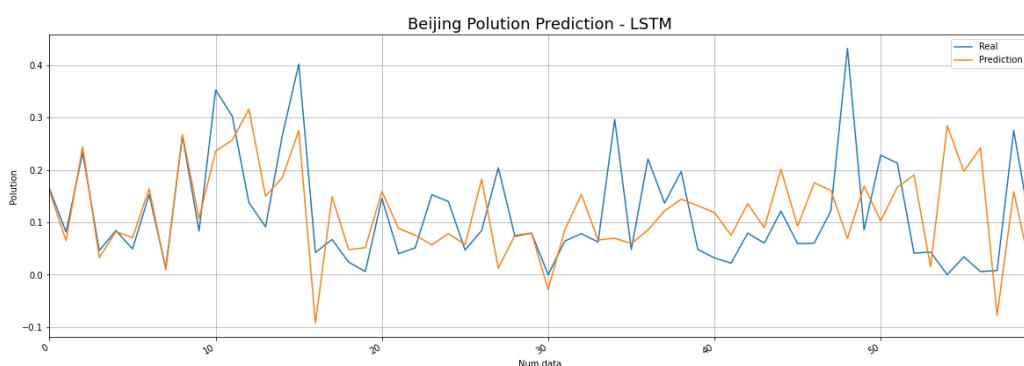
شکل 352: زمان اجرای آموزش در حالت مدل LSTM و با Adam و MSE

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 353: نمودار Loss در حالت مدل LSTM با Adam و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 354: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت Adam و MSE

```

#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0008688949955242042
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 355: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقیق شبكه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقیق شبكه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

► 8- بررسی شبکه LSTM در حالت MAE و Adam: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 356: تعریف مدل و Loss وتابع Optimizer در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.05415902193635702 , Valid Loss : => 0.012735426425933838
***->>-----<<<-***
step : 1 Train loss : 0.05090161379426718 , Valid Loss : => 0.012344134226441383
***->>-----<<<-***
step : 2 Train loss : 0.050389051139354706 , Valid Loss : => 0.01215990498661995
***->>-----<<<-***
step : 3 Train loss : 0.05011952731758356 , Valid Loss : => 0.01199258878827095
***->>-----<<<-***
step : 4 Train loss : 0.04978443708270788 , Valid Loss : => 0.011622858121991158
***->>-----<<<-***
step : 5 Train loss : 0.049112596996128556 , Valid Loss : => 0.011304673478007317
***->>-----<<<-***

```

شکل 357: ایپاک اول مدل LSTM

```

***->>-----<<-***  

step : 193 Train loss : 0.029941273080185057 , Valid Loss : => 0.012086506634950638  

***->>-----<<-***  

step : 194 Train loss : 0.029794043684378265 , Valid Loss : => 0.011895116865634919  

***->>-----<<-***  

step : 195 Train loss : 0.029217073395848274 , Valid Loss : => 0.011960019171237946  

***->>-----<<-***  

step : 196 Train loss : 0.02881829483434558 , Valid Loss : => 0.012212571352720261  

***->>-----<<-***  

step : 197 Train loss : 0.028948631137609482 , Valid Loss : => 0.012401792109012605  

***->>-----<<-***  

step : 198 Train loss : 0.029975066911429166 , Valid Loss : => 0.012071225568652153  

***->>-----<<-***  

step : 199 Train loss : 0.029710893761366607 , Valid Loss : => 0.013118869960308074  

***->>-----<<-***
```

شکل 358: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

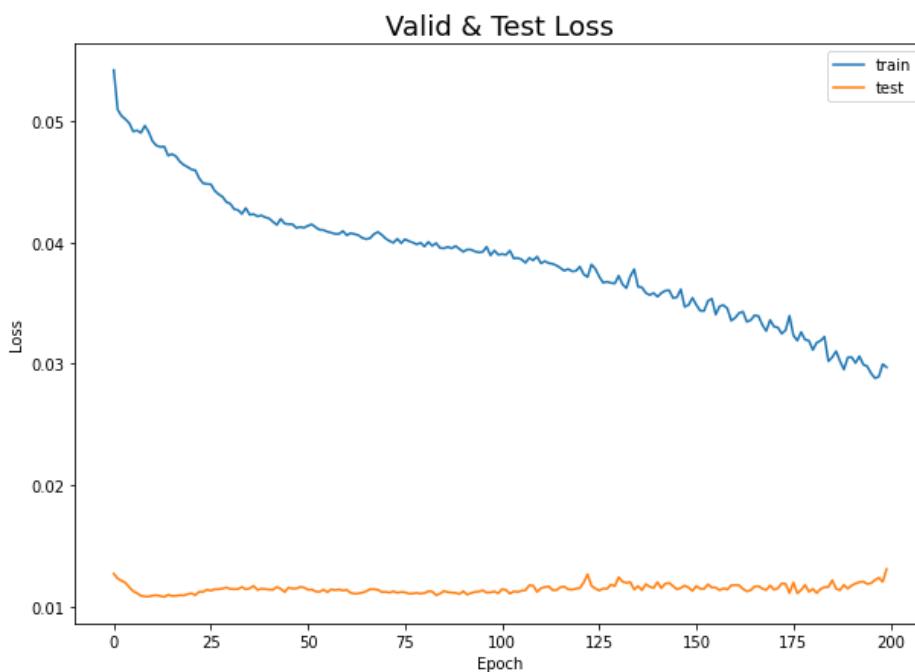
The total Training Time is Equal with ==> : 66.65807032585144 Sec.  

*****  

=====
```

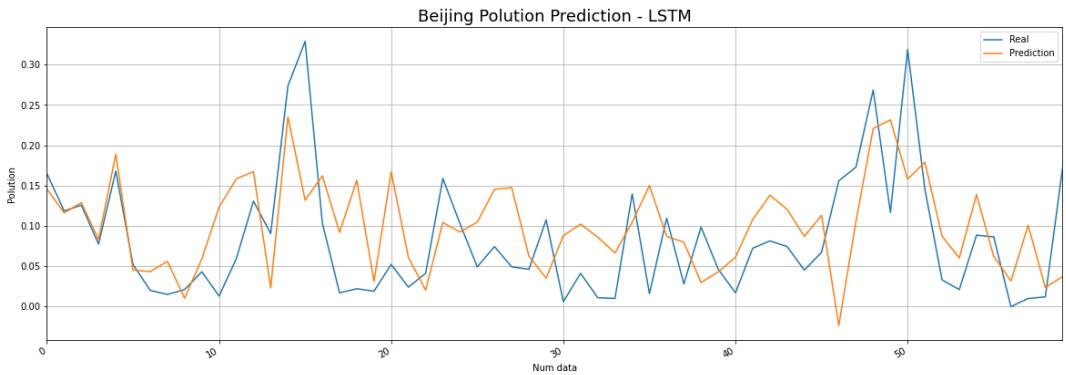
شکل 359: زمان اجرای آموزش در حالت مدل LSTM و با Adam

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 360: نمودار Loss در حالت مدل LSTM با Adam و MAE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 361: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت Adam و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
*** Test Loss :==>> 0.008864995704165527
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 362: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است. همچنین خوب است به این نکته اشاره کنم که نمودار Loss عملکرد خیلی خوبی ندارد به این دلیل که مقدار Loss مربوط به Val از Train بدتر است و علت آن می‌تواند کم بودن داده‌های تست باشد.

► 9- بررسی شبکه LSTM در حالت AdaGrad و MSE

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.0003)
```

شکل 363: تعریف مدل و Optimizer وتابع Loss در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می پردازیم:

```
step : 0 Train loss : 0.005589308749185875 , Valid Loss : => 0.000992159562301822
***->>>-----<<<-***  
step : 1 Train loss : 0.005421364514622837 , Valid Loss : => 0.0009888068865984679
***->>>-----<<<-***  
step : 2 Train loss : 0.005386170404963195 , Valid Loss : => 0.0009865177050232888
***->>>-----<<<-***  
step : 3 Train loss : 0.005361894904635847 , Valid Loss : => 0.0009846019529504701
***->>>-----<<<-***  
step : 4 Train loss : 0.005342748973052949 , Valid Loss : => 0.0009829241415718571
***->>>-----<<<-***  
step : 5 Train loss : 0.005326654859818519 , Valid Loss : => 0.0009814209036994725
***->>>-----<<<-***
```

شکل 364: ایپاک اول مدل LSTM

```
***->>>-----<<<-***  
step : 193 Train loss : 0.004728850519750267 , Valid Loss : => 0.00094628602033481
***->>>-----<<<-***  
step : 194 Train loss : 0.004727252492448315 , Valid Loss : => 0.0009463355992920697
***->>>-----<<<-***  
step : 195 Train loss : 0.004725660182884894 , Valid Loss : => 0.0009463859722018241
***->>>-----<<<-***  
step : 196 Train loss : 0.004724073160323314 , Valid Loss : => 0.0009464371926151216
***->>>-----<<<-***  
step : 197 Train loss : 0.0047224914265098054 , Valid Loss : => 0.0009464892256073654
***->>>-----<<<-***  
step : 198 Train loss : 0.0047209151682909576 , Valid Loss : => 0.0009465421130880713
***->>>-----<<<-***  
step : 199 Train loss : 0.004719344471814111 , Valid Loss : => 0.0009465957060456276
***->>>-----<<<-***
```

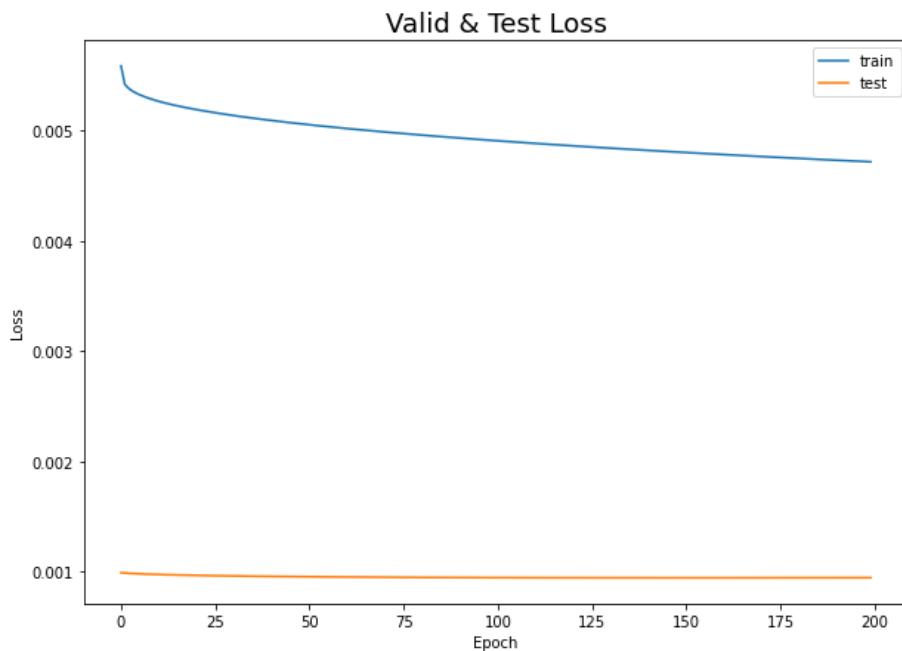
شکل 365: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 60.02805280685425 Sec.  
*****  
=====
```

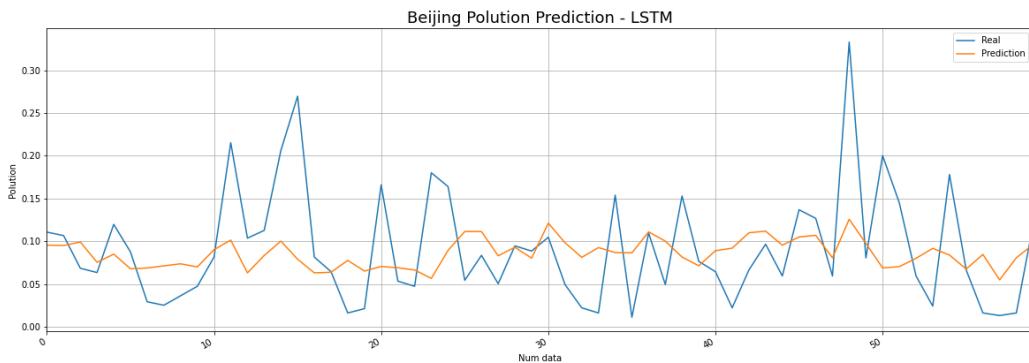
شکل 366: زمان اجرای آموزش در حالت مدل LSTM و با AdaGrad و MSE

سپس در ادامه می توانید نمودار مربوط به این Loss ها مشاهده نمایید:



شکل 367: نمودار Loss و AdaGrad با LSTM در حالت مدل

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 368: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت AdaGrad و MSE

```
#####
>>>-----<<<
>>>-----*****
**** Test Loss :==>>> 0.0008803275804634072
>>>-----*****
>>>-----<<<
#####
```

شکل 369: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز

افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی ببینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

همچنین خوب است به این نکته اشاره کنم که نمودار Loss عملکرد خیلی خوبی ندارد به این دلیل که مقدار Loss مربوط به Val از Train بدتر است و علت آن می‌تواند کم بودن داده‌های تست باشد.

با بررسی حالت AdaGrad این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف تر است.

► 10- بررسی شبکه LSTM در حالت MAE و AdaGrad: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 370: تعریف مدل و Optimizer وتابع Loss در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.061729055121541025 , Valid Loss : => 0.014756776876747608
***->>-----<<<-***  
step : 1 Train loss : 0.05554937772452831 , Valid Loss : => 0.014179187640547753
***->>-----<<<-***  
step : 2 Train loss : 0.05466005928814411 , Valid Loss : => 0.013761986643075944
***->>-----<<<-***  
step : 3 Train loss : 0.054389602430164816 , Valid Loss : => 0.013404714167118073
***->>-----<<<-***  
step : 4 Train loss : 0.05410434160381555 , Valid Loss : => 0.013349419124424458
***->>-----<<<-***  
step : 5 Train loss : 0.05366771962493658 , Valid Loss : => 0.013089622780680656
***->>-----<<<-***
```

شکل 371: ایپاک اول مدل LSTM

```

***->>-----<<<-***  

step : 193 Train loss : 0.03414423594251275 , Valid Loss : => 0.016226846501231195  

***->>-----<<<-***  

step : 194 Train loss : 0.03402831112034619 , Valid Loss : => 0.016067175790667534  

***->>-----<<<-***  

step : 195 Train loss : 0.03412987489253282 , Valid Loss : => 0.01595175713300705  

***->>-----<<<-***  

step : 196 Train loss : 0.033089674925431606 , Valid Loss : => 0.015916610434651375  

***->>-----<<<-***  

step : 197 Train loss : 0.03353325236588717 , Valid Loss : => 0.016091407015919684  

***->>-----<<<-***  

step : 198 Train loss : 0.03322386441752315 , Valid Loss : => 0.0163567928224802  

***->>-----<<<-***  

step : 199 Train loss : 0.033996440898627044 , Valid Loss : => 0.016306012198328973  

***->>-----<<<-***
```

شکل 372: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

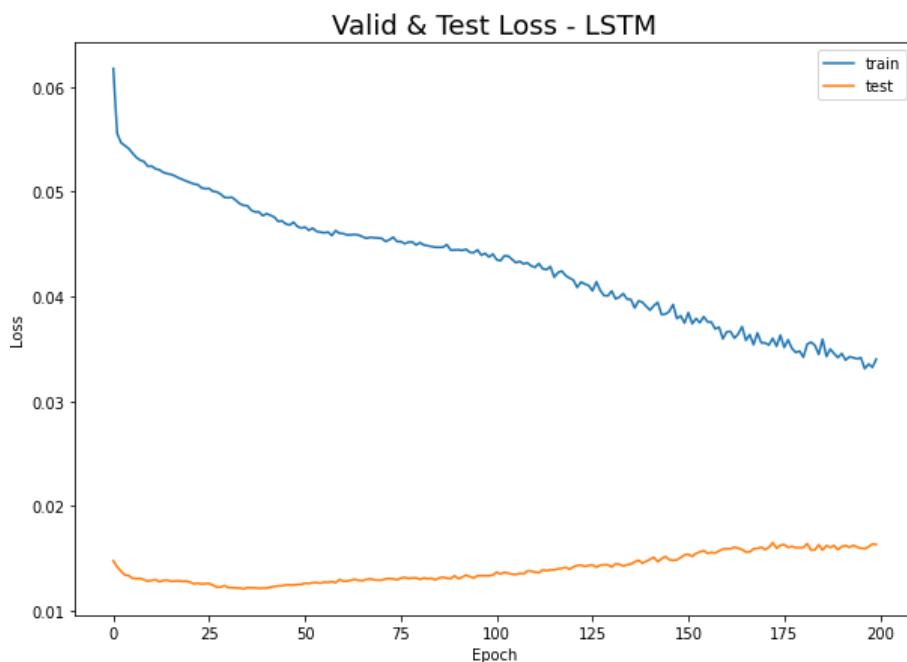
The total Training Time is Equal with ==> : 55.541584491729736 Sec.  

*****  

=====
```

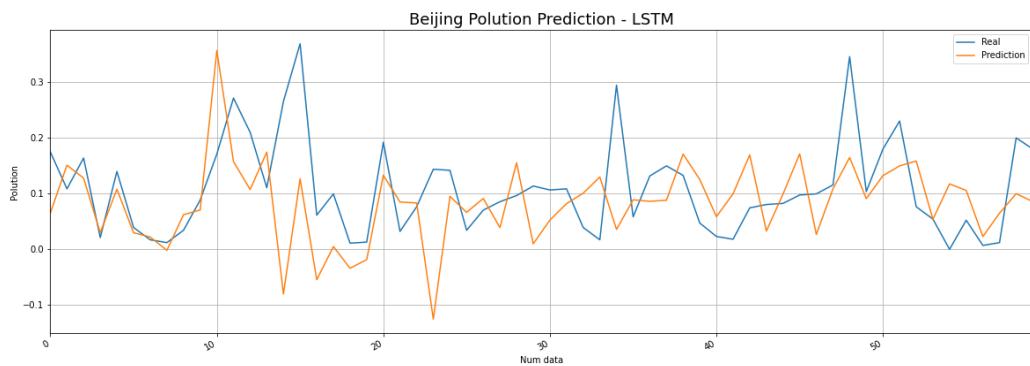
شکل 373: زمان اجرای آموزش در حالت مدل LSTM و با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 374: نمودار Loss در حالت مدل LSTM با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 375: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت AdaGrad و MAE

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.009736149953234763
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 376: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقیق شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقیق شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است. همچنین خوب است به این نکته اشاره کنم که نمودار Loss عملکرد خیلی خوبی ندارد به این دلیل که مقدار Loss مربوط به Val از Train بدتر است و علت آن می‌تواند کم بودن داده‌های تست باشد.

با بررسی حالت AdaGrad این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف تر است.

► 11- بررسی شبکه LSTM در حالت RMSprop و MSE: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)

```

شکل 377: تعریف مدل و Loss و تابع Optimizer در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.006421088291099295 , Valid Loss : => 0.0011208784382324667
***->>>-----<<<-***  

step : 1 Train loss : 0.005492636051494628 , Valid Loss : => 0.0011085695284418762
***->>>-----<<<-***  

step : 2 Train loss : 0.005348074453650042 , Valid Loss : => 0.001096154834376648
***->>>-----<<<-***  

step : 3 Train loss : 0.005236409847857431 , Valid Loss : => 0.0010838207346387208
***->>>-----<<<-***  

step : 4 Train loss : 0.005136014590971172 , Valid Loss : => 0.001071534852962941
***->>>-----<<<-***  

step : 5 Train loss : 0.005040577446343377 , Valid Loss : => 0.001059353940654546
***->>>-----<<<-***  


```

شکل 378: ایپاک اول مدل LSTM

```

***->>>-----<<<-***  

step : 193 Train loss : 0.0018291729694465174 , Valid Loss : => 0.0015865282015874982
***->>>-----<<<-***  

step : 194 Train loss : 0.0017460205644601956 , Valid Loss : => 0.0019015969173051418
***->>>-----<<<-***  

step : 195 Train loss : 0.0018000955251045525 , Valid Loss : => 0.0015728787938132883
***->>>-----<<<-***  

step : 196 Train loss : 0.0017252874898258596 , Valid Loss : => 0.001894424573983997
***->>>-----<<<-***  

step : 197 Train loss : 0.001764847867307253 , Valid Loss : => 0.0015714870346710086
***->>>-----<<<-***  

step : 198 Train loss : 0.0017048092113691382 , Valid Loss : => 0.0018899524165317417
***->>>-----<<<-***  

step : 199 Train loss : 0.0017346716049360111 , Valid Loss : => 0.0015662303613498806
***->>>-----<<<-***  


```

شکل 379: ایپاک آخر مدل LSTM

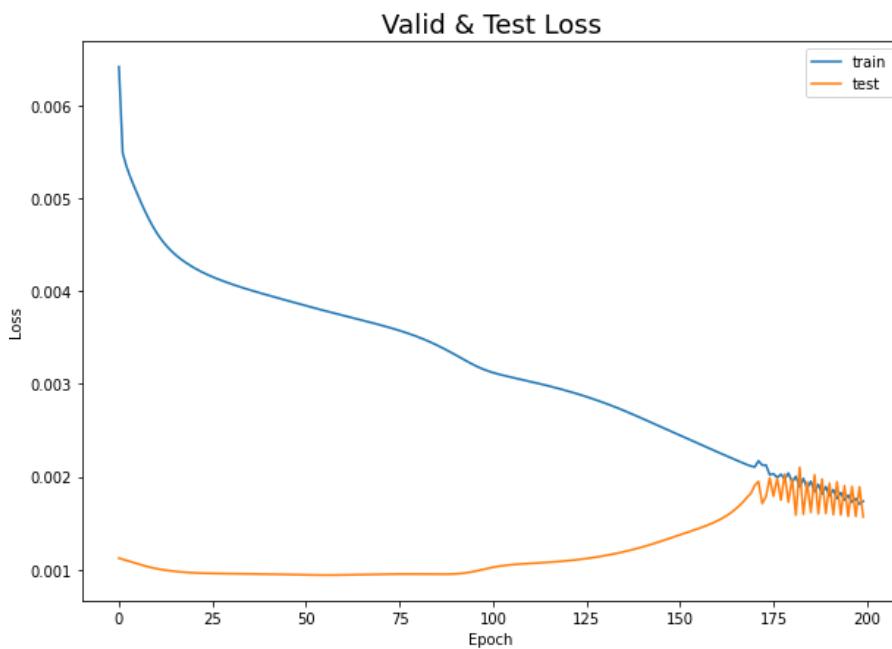
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 57.12970781326294 Sec.
*****
=====
```

شکل 380: زمان اجرای آموزش در حالت مدل LSTM و RMSprop و MSE

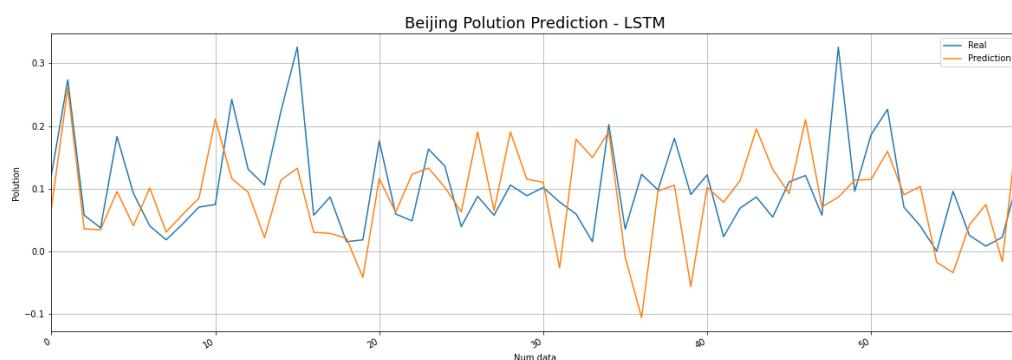
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 381: نمودار Loss در حالت مدل LSTM با RMSprop و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد)

مشاهده نمایید:



شکل 382: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت RMSprop و MSE

```
#####
>>>-----<<<
>>>-----*****-----<<<
*** Test Loss :==>> 0.0006946152289553235
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 383: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز

افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

همچنین خوب است به این نکته اشاره کنم که نمودار Loss عملکرد خیلی خوبی ندارد به این دلیل که مقدار Loss مربوط به Val از Train بدتر است و علت آن می‌تواند کم بودن داده‌های تست باشد.

با بررسی حالت RMSprop این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف تر است.

► 12- بررسی شبکه LSTM در حالت MAE و RMSprop: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 384: تعریف مدل و Optimizer وتابع Loss در مدل LSTM

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss :  0.06769805908203125 , Valid Loss : =>  0.015439792089164257
***->>>-----<<<-***  
step : 1 Train loss :  0.0631901264935732 , Valid Loss : =>  0.015461174845695496
***->>>-----<<<-***  
step : 2 Train loss :  0.06222976926714182 , Valid Loss : =>  0.015461396165192128
***->>>-----<<<-***  
step : 3 Train loss :  0.061249075196683406 , Valid Loss : =>  0.015490798652172089
***->>>-----<<<-***  
step : 4 Train loss :  0.06035180237144232 , Valid Loss : =>  0.015533504970371724
***->>>-----<<<-***  
step : 5 Train loss :  0.05985966894775629 , Valid Loss : =>  0.015558520518243312
***->>>-----<<<-***
```

شکل 385: ایپاک اول مدل LSTM

```

***->>-----<<-***  

step : 193 Train loss : 0.028671859515209994 , Valid Loss : => 0.01593522777160009  

***->>-----<<-***  

step : 194 Train loss : 0.029308885149657727 , Valid Loss : => 0.015945013364156088  

***->>-----<<-***  

step : 195 Train loss : 0.02844027305642764 , Valid Loss : => 0.016392490764458974  

***->>-----<<-***  

step : 196 Train loss : 0.02925392227868239 , Valid Loss : => 0.016055882473786674  

***->>-----<<-***  

step : 197 Train loss : 0.02877275695403417 , Valid Loss : => 0.015726429728719654  

***->>-----<<-***  

step : 198 Train loss : 0.02898490953569611 , Valid Loss : => 0.01596513589223226  

***->>-----<<-***  

step : 199 Train loss : 0.028325286197165647 , Valid Loss : => 0.015990523993968962  

***->>-----<<-***
```

شکل 386: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

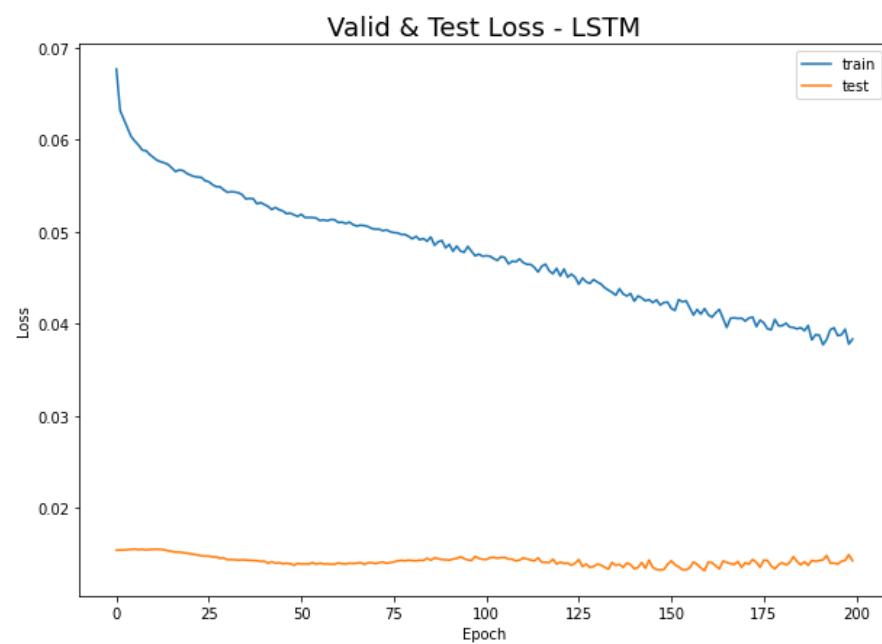
The total Training Time is Equal with ==> : 54.34002089500427 Sec.  

*****  

=====
```

شکل 387: زمان اجرای آموزش در حالت مدل LSTM و با MAE و RMSprop

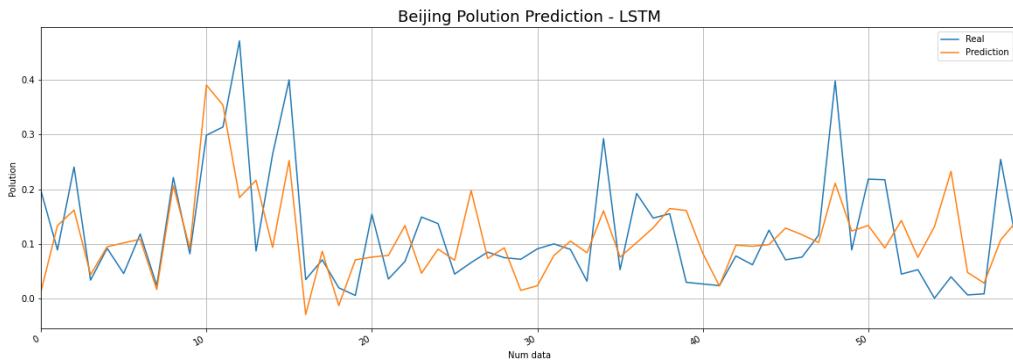
سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 388: نمودار Loss با LSTM در حالت مدل MAE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد)

مشاهده نمایید:



شکل 389: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت MAE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss ==>>> 0.009631662523107869
>>>-----*****-----<<<
>>>-----*****-----<<<
#####
#
```

شکل 390: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است. همچنین خوب است به این نکته اشاره کنم که نمودار Loss عملکرد خیلی خوبی ندارد به این دلیل که مقدار Loss مربوط به Val از Train بدتر است و علت آن می‌تواند کم بودن داده‌های تست باشد.

با بررسی حالت RMSprop این مدل نسبت به Adam معلوم می‌شود که عملکرد این حالت ضعیف تر است.

► 13- بررسی شبکه GRU در حالت Adam و MSE: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 391: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.0029467621646333663 , Valid Loss : => 0.0021262213960289954
***->>>-----<<<-***  
step : 1 Train loss : 0.0019586405223457233 , Valid Loss : => 0.001630306299775839
***->>>-----<<<-***  
step : 2 Train loss : 0.0017895524606241712 , Valid Loss : => 0.0014788082474842667
***->>>-----<<<-***  
step : 3 Train loss : 0.0017266193532296235 , Valid Loss : => 0.0014081459725275637
***->>>-----<<<-***  
step : 4 Train loss : 0.0016806778586691334 , Valid Loss : => 0.001366512798704207
***->>>-----<<<-***  
step : 5 Train loss : 0.0016350458409371122 , Valid Loss : => 0.0013306803372688592
***->>>-----<<<-***  
...
```

شکل ۳۹۲: آپیک اول مدل GRU

```
***->>>-----<<<-***  
step : 193 Train loss : 0.00013701615756699106 , Valid Loss : => 0.003515254065860063
***->>>-----<<<-***  
step : 194 Train loss : 0.00024077140211981412 , Valid Loss : => 0.0036829482857137917
***->>>-----<<<-***  
step : 195 Train loss : 0.00030825410898874646 , Valid Loss : => 0.003691119900904596
***->>>-----<<<-***  
step : 196 Train loss : 0.0002546275714675652 , Valid Loss : => 0.003691816688515246
***->>>-----<<<-***  
step : 197 Train loss : 0.00028188301747875463 , Valid Loss : => 0.0030462604761123655
***->>>-----<<<-***  
step : 198 Train loss : 0.0002322149361480981 , Valid Loss : => 0.0030823575844988226
***->>>-----<<<-***  
step : 199 Train loss : 0.000146500199306978 , Valid Loss : => 0.0030483274231664836
***->>>-----<<<-***
```

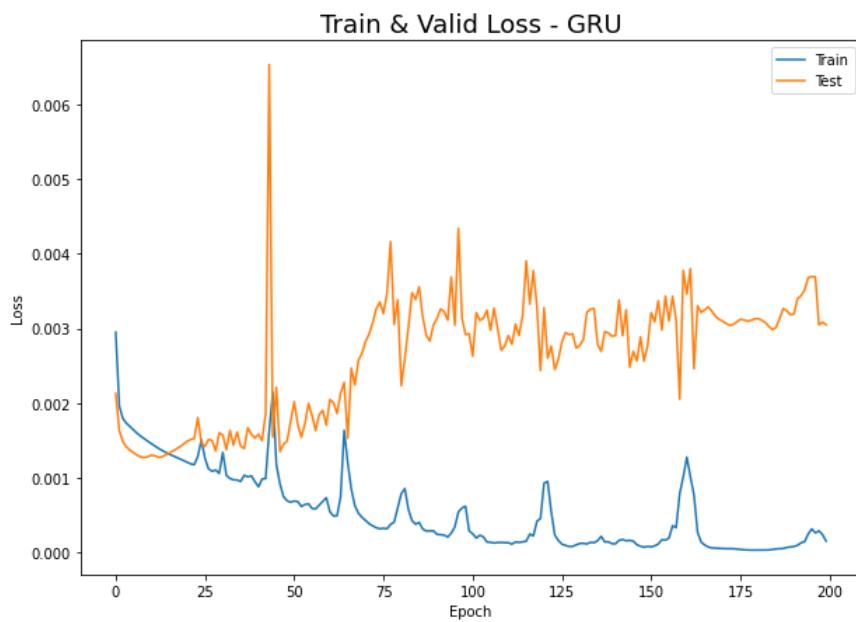
شکل ۳۹۳: آپیک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 60.030548095703125 Sec.  
*****  
=====
```

شکل ۳۹۴: زمان اجرای آموزش در حالت مدل GRU و با Adam و MSE

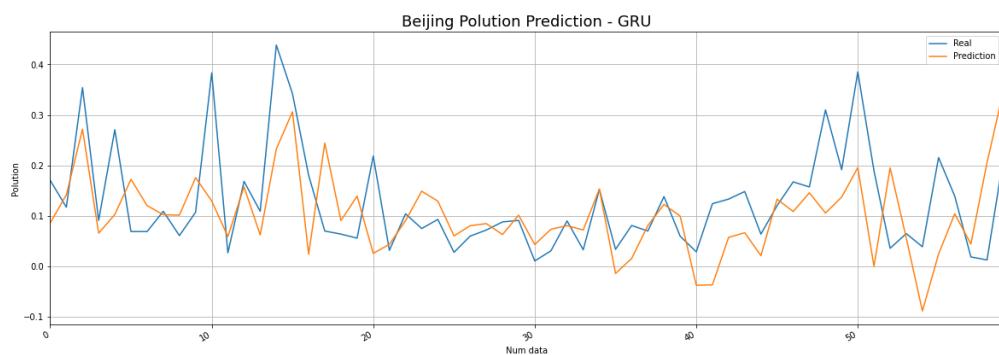
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 395: نمودار Loss در حالت مدل GRU با MSE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 396: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MSE و Adam

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0008031314616562754
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 397: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را

می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

با بررسی حالت GRU در مقایسه با LSTM مشخص می‌شود که این مدل عملکرد ضعیفتری در مقایسه با LSTM دارد.

۱۴- بررسی شبکه GRU در حالت MAE و Adam: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل ۳۹۸: تعریف مدل وتابع Optimizer و Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.013542875643109992 , Valid Loss : => 0.010131923072040082
***->>>-----<<<-***  
step : 1 Train loss : 0.012758186378843933 , Valid Loss : => 0.010750231929123402
***->>>-----<<<-***  
step : 2 Train loss : 0.01195196771018562 , Valid Loss : => 0.011590601950883865
***->>>-----<<<-***  
step : 3 Train loss : 0.012001783294337137 , Valid Loss : => 0.012914895713329315
***->>>-----<<<-***  
step : 4 Train loss : 0.011956132487172172 , Valid Loss : => 0.010769130475819112
***->>>-----<<<-***  
step : 5 Train loss : 0.011108941806568986 , Valid Loss : => 0.010208301730453968
***->>>-----<<<-***
```

شکل ۳۹۹: ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 193 Train loss : 0.004043161869049072 , Valid Loss : => 0.011837176941335202
***->>>-----<<<-***  
step : 194 Train loss : 0.003453403924192701 , Valid Loss : => 0.01246729403734207
***->>>-----<<<-***  
step : 195 Train loss : 0.003937220812908241 , Valid Loss : => 0.01277549222111702
***->>>-----<<<-***  
step : 196 Train loss : 0.004644836347904943 , Valid Loss : => 0.012436630725860596
***->>>-----<<<-***  
step : 197 Train loss : 0.0046216830079044615 , Valid Loss : => 0.013225165866315365
***->>>-----<<<-***  
step : 198 Train loss : 0.003937321892451672 , Valid Loss : => 0.01178866695612669
***->>>-----<<<-***  
step : 199 Train loss : 0.0041158808949625205 , Valid Loss : => 0.012298450917005538
***->>>-----<<<-***
```

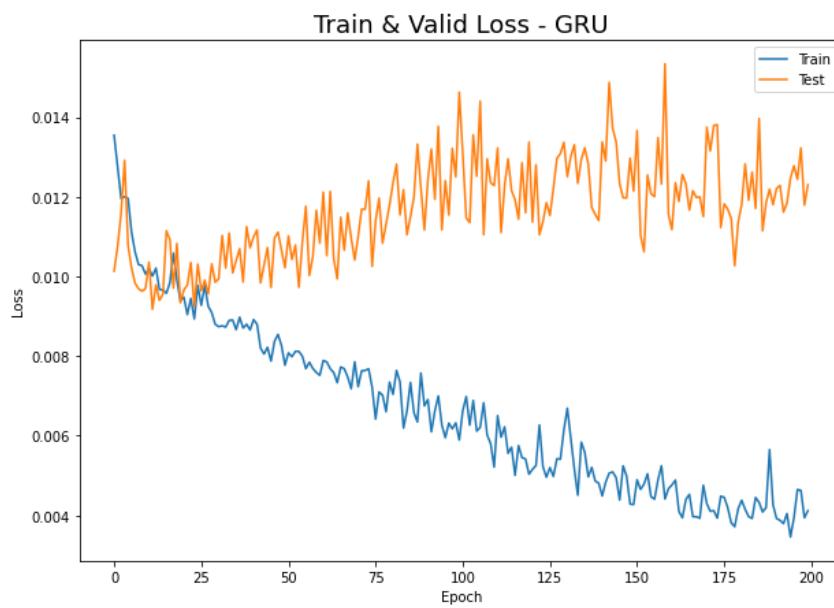
شکل ۴۰۰: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
***** The total Training Time is Equal with ==> : 57.08388900756836 Sec. ****=
=====
```

شکل 401: زمان اجرای آموزش در حالت مدل GRU و با MAE و Adam

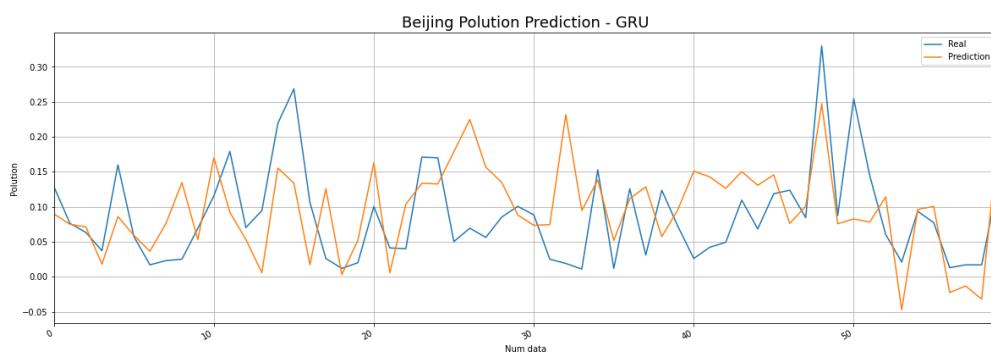
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 402: نمودار Loss در حالت مدل GRU با MAE و Adam

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 403: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MAE و Adam

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>>> 0.0055354155356153135
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 404: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی ببینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

با بررسی حالت GRU در مقایسه با LSTM مشخص می‌شود که این مدل عملکرد ضعیفتری در مقایسه با LSTM دارد.

► 15- بررسی شبکه GRU در حالت MSE و AdaGrad: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

شکل 405: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.013542875643109992 , Valid Loss : => 0.010131923072040082
***->>>-----<<-***
step : 1 Train loss : 0.012758186370843933 , Valid Loss : => 0.010750231929123402
***->>>-----<<-***
step : 2 Train loss : 0.01195196771018562 , Valid Loss : => 0.011590601950883865
***->>>-----<<-***
step : 3 Train loss : 0.012001783294337137 , Valid Loss : => 0.012914895713329315
***->>>-----<<-***
step : 4 Train loss : 0.011956132487172172 , Valid Loss : => 0.010769130475819112
***->>>-----<<-***
step : 5 Train loss : 0.011108941806568986 , Valid Loss : => 0.010208301730453968
***->>>-----<<-***
```

شکل 406: ایپاک اول مدل GRU

```

***->>-----<<<-***  

step : 193 Train loss : 0.004043161869049072 , Valid Loss : => 0.011837176941335202  

***->>-----<<<-***  

step : 194 Train loss : 0.003453403924192701 , Valid Loss : => 0.01246729403734207  

***->>-----<<<-***  

step : 195 Train loss : 0.003937220812908241 , Valid Loss : => 0.01277549222111702  

***->>-----<<<-***  

step : 196 Train loss : 0.004644836347904943 , Valid Loss : => 0.012436630725860596  

***->>-----<<<-***  

step : 197 Train loss : 0.0046216830079044615 , Valid Loss : => 0.013225165866315365  

***->>-----<<<-***  

step : 198 Train loss : 0.003937321892451672 , Valid Loss : => 0.01178866695612669  

***->>-----<<<-***  

step : 199 Train loss : 0.0041158808949625205 , Valid Loss : => 0.012298450917005538  

***->>-----<<<-***
```

شکل 407: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

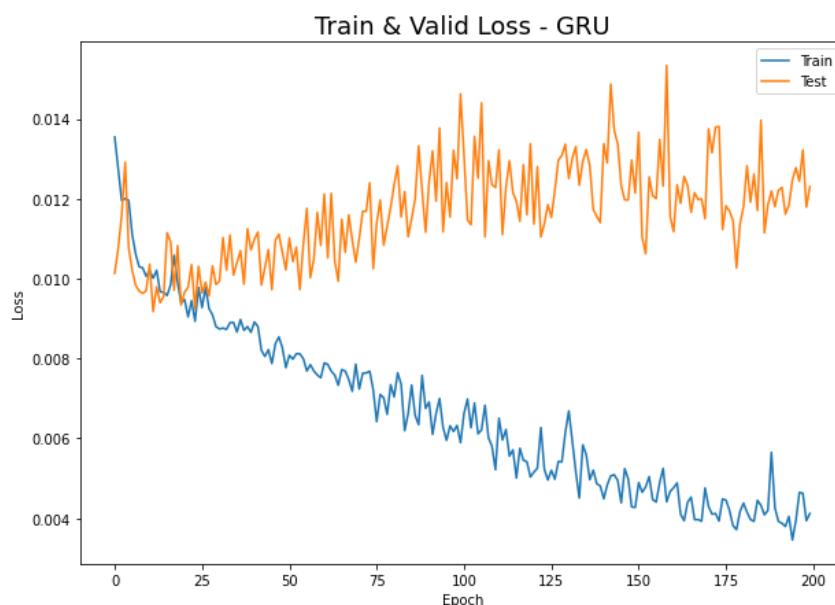
The total Training Time is Equal with ==> : 57.08388900756836 Sec.  

*****  

=====
```

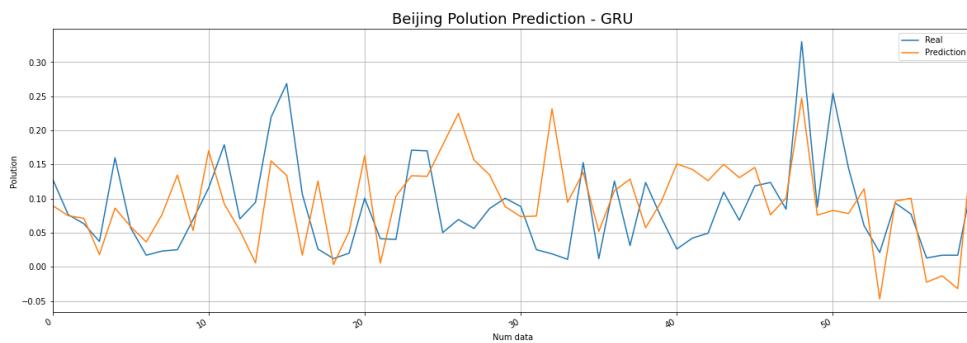
شکل 408: زمان اجرای آموزش در حالت مدل GRU و با MSE و AdaGrad

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 409: نمودار Loss در حالت مدل GRU با Adagrad و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (60 عدد) مشاهده نمایید:



شکل 410: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MSE و AdaGrad

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0055354155356153135
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 411: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی ببینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

با بررسی حالت GRU در مقایسه با LSTM مشخص می‌شود که این مدل عملکرد ضعیفتری در مقایسه با LSTM دارد.

این مدل در حالت Adam در مقایسه با Adagrad بدتری دارد.

► 16- بررسی شبکه GRU در حالت MAE و AdaGrad: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.Adagrad(model.parameters(), lr=0.001)
```

شکل 412: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.015041442064657098 , Valid Loss : => 0.01520942833274603
***->>>-----<<<-***  

step : 1 Train loss : 0.014574785903096198 , Valid Loss : => 0.015102971270680427
***->>>-----<<<-***  

step : 2 Train loss : 0.014420741543705975 , Valid Loss : => 0.015266174785792827
***->>>-----<<<-***  

step : 3 Train loss : 0.014213431826127427 , Valid Loss : => 0.015260315462946893
***->>>-----<<<-***  

step : 4 Train loss : 0.014134467721340202 , Valid Loss : => 0.015259940698742867
***->>>-----<<<-***  

step : 5 Train loss : 0.014031469360703513 , Valid Loss : => 0.015255977436900138
***->>>-----<<<-***
```

شکل 413: ایپاک اول مدل GRU

```

***->>>-----<<<-***  

step : 193 Train loss : 0.012580376646171013 , Valid Loss : => 0.014689932093024254
***->>>-----<<<-***  

step : 194 Train loss : 0.012576877028636989 , Valid Loss : => 0.014686032831668853
***->>>-----<<<-***  

step : 195 Train loss : 0.01257001594862058 , Valid Loss : => 0.014688003733754158
***->>>-----<<<-***  

step : 196 Train loss : 0.012570552102157047 , Valid Loss : => 0.014684306308627128
***->>>-----<<<-***  

step : 197 Train loss : 0.012567048848030112 , Valid Loss : => 0.014680448845028877
***->>>-----<<<-***  

step : 198 Train loss : 0.01255564336620626 , Valid Loss : => 0.014680656939744949
***->>>-----<<<-***  

step : 199 Train loss : 0.012560767777973698 , Valid Loss : => 0.014676948711276054
***->>>-----<<<-***
```

شکل 414: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

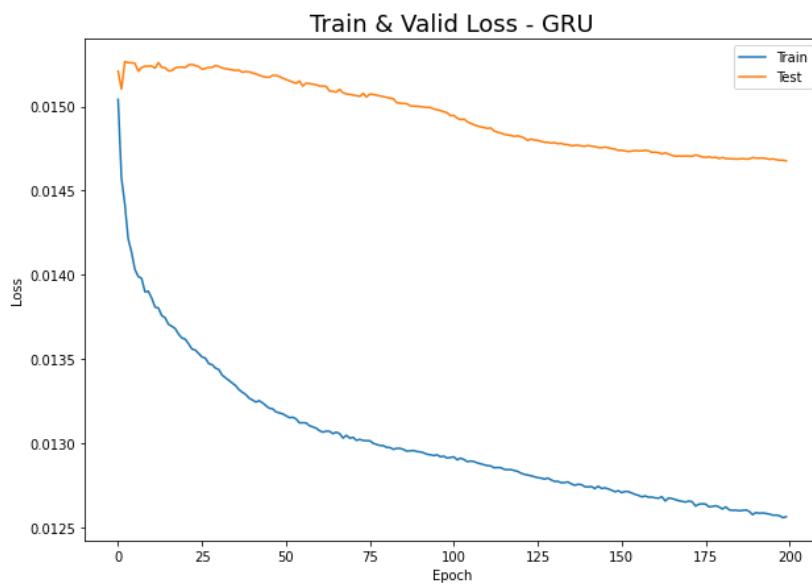
The total Training Time is Equal with ==> : 47.53907513618469 Sec.  

*****  

=====
```

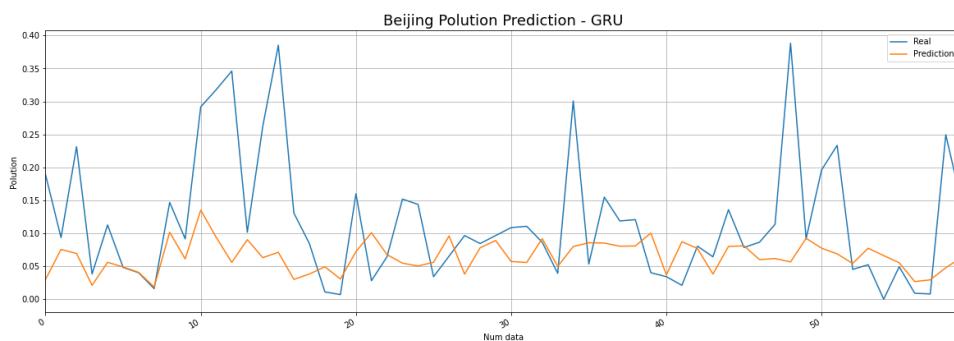
شکل 415: زمان اجرای آموزش در حالت مدل GRU و با MAE و AdaGrad

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 416: نمودار Loss در حالت مدل GRU با MAE و Adagrad

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 417: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MAE و AdaGrad

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.012758495742898612
>>>-----*****-----<<<
>>>-----*****-----<<<
#####

```

شکل 418: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقیق‌تر شبکه بهتر است نسبت به RNN و این موضوع را

می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خوب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است.

با بررسی حالت GRU در مقایسه با LSTM مشخص می‌شود که این مدل عملکرد ضعیفتری در مقایسه با LSTM دارد.

این مدل در حالت Adagrad نسبت به Adam عملکرد بدتری دارد.

► ۱۷- بررسی شبکه GRU در حالت MSE و RMSprop: حالت هفتگی

ابتدا به تعریف Loss وتابع Optimizer می‌پردازیم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)
```

شکل ۴۱۹: تعریف مدل و Loss در مدل GRU وتابع Optimizer

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 0.010514660189593477 , Valid Loss : => 0.0014839759736787529
***->>>-----<<<-***  
step : 1 Train loss : 0.0014772396412722412 , Valid Loss : => 0.0013422312925104052
***->>>-----<<<-***  
step : 2 Train loss : 0.0014539063004555092 , Valid Loss : => 0.001314274511532858
***->>>-----<<<-***  
step : 3 Train loss : 0.0014332593124847682 , Valid Loss : => 0.001290256503270939
***->>>-----<<<-***  
step : 4 Train loss : 0.001413391915793043 , Valid Loss : => 0.00126781880739145
***->>>-----<<<-***  
step : 5 Train loss : 0.0013927916901940037 , Valid Loss : => 0.0012448502308689058
***->>>-----<<<-***
```

شکل ۴۲۰: ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 193 Train loss : 0.00010859360700123943 , Valid Loss : => 0.0011404943792149424
***->>>-----<<<-***  
step : 194 Train loss : 0.00010082576194517537 , Valid Loss : => 0.0011131049878895282
***->>>-----<<<-***  
step : 195 Train loss : 0.0001161471624246111 , Valid Loss : => 0.001444111093878746
***->>>-----<<<-***  
step : 196 Train loss : 0.00011685214459638311 , Valid Loss : => 0.0012046369165182114
***->>>-----<<<-***  
step : 197 Train loss : 9.511470698704388e-05 , Valid Loss : => 0.0009521121857687831
***->>>-----<<<-***  
step : 198 Train loss : 8.503608443702216e-05 , Valid Loss : => 0.0011618149024434387
***->>>-----<<<-***  
step : 199 Train loss : 9.799274092594194e-05 , Valid Loss : => 0.001239409614354372
***->>>-----<<<-***
```

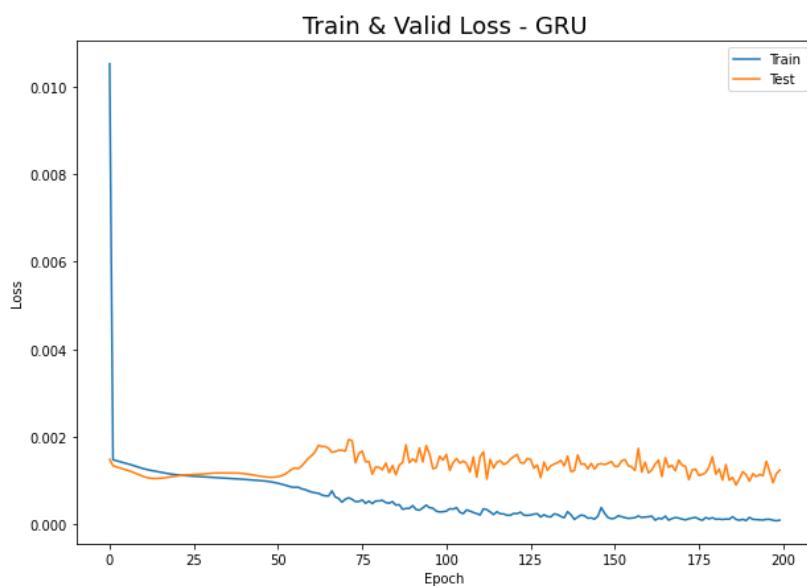
شکل ۴۲۱: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
***** The total Training Time is Equal with ==> : 49.87543869018555 Sec. ****=
=====
```

شکل 422: زمان اجرای آموزش در حالت مدل GRU و RMSprop با MSE

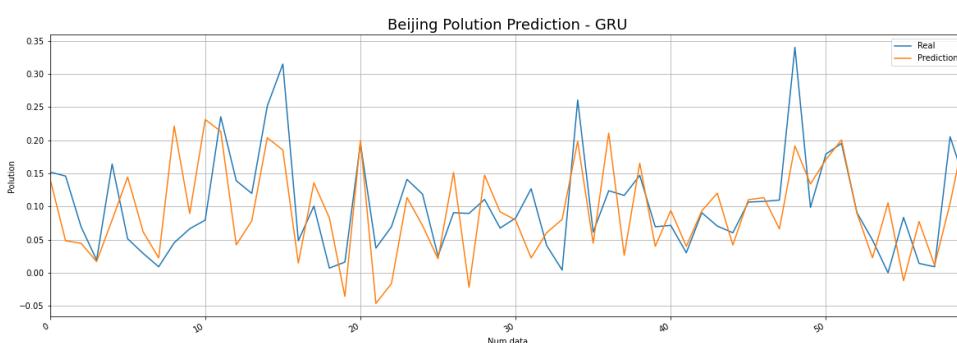
سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 423: نمودار Loss در حالت مدل GRU با RMSprop و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد)

مشاهده نمایید:



شکل 424: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت AdaGrad و RMSprop

```

#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.0004479300225544388
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 425: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقت شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی بینیم. اما خب همچنان دقت شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است. همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقت‌های مشاهبی را به دست بیاورد.

این مدل در حالت RMSprop عملکرد بهتری از Adam داشت.

► 18- بررسی شبکه GRU در حالت MAE و RMSprop: حالت هفتگی

ابتدا به تعریف Optimizer وتابع Loss می‌پردازیم:

```

1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=6, n_hidden_layers=233, n_layers=1)
3 criterion = nn.L1Loss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.001)

```

شکل 426: تعریف مدل و Optimizer وتابع Loss در مدل GRU

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 0.024896869706433445 , Valid Loss : => 0.016006122305989266
***->>>-----<<<_***
step : 1 Train loss : 0.015312880517116614 , Valid Loss : => 0.014499163702130318
***->>>-----<<<_***
step : 2 Train loss : 0.0147985379876835 , Valid Loss : => 0.015831734016537667
***->>>-----<<<_***
step : 3 Train loss : 0.014573877278183187 , Valid Loss : => 0.015393048077821731
***->>>-----<<<_***
step : 4 Train loss : 0.014397973314459835 , Valid Loss : => 0.015180471539497375
***->>>-----<<<_***
step : 5 Train loss : 0.013875613361597062 , Valid Loss : => 0.01567119315266609
***->>>-----<<<_***

```

شکل 427: ایپاک اول مدل GRU

```

***->>>-----<<<-***  

step : 193 Train loss : 0.004561621700191781 , Valid Loss : => 0.016419209390878677  

***->>>-----<<<-***  

step : 194 Train loss : 0.005735218001618272 , Valid Loss : => 0.014874628260731697  

***->>>-----<<<-***  

step : 195 Train loss : 0.004974885067592065 , Valid Loss : => 0.014723653867840768  

***->>>-----<<<-***  

step : 196 Train loss : 0.004972255664567152 , Valid Loss : => 0.015407330431044102  

***->>>-----<<<-***  

step : 197 Train loss : 0.004912737211478608 , Valid Loss : => 0.01666088916361332  

***->>>-----<<<-***  

step : 198 Train loss : 0.004739481955766678 , Valid Loss : => 0.016076497957110403  

***->>>-----<<<-***  

step : 199 Train loss : 0.004303269729106909 , Valid Loss : => 0.014291177168488503  

***->>>-----<<<-***
```

شکل 428: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

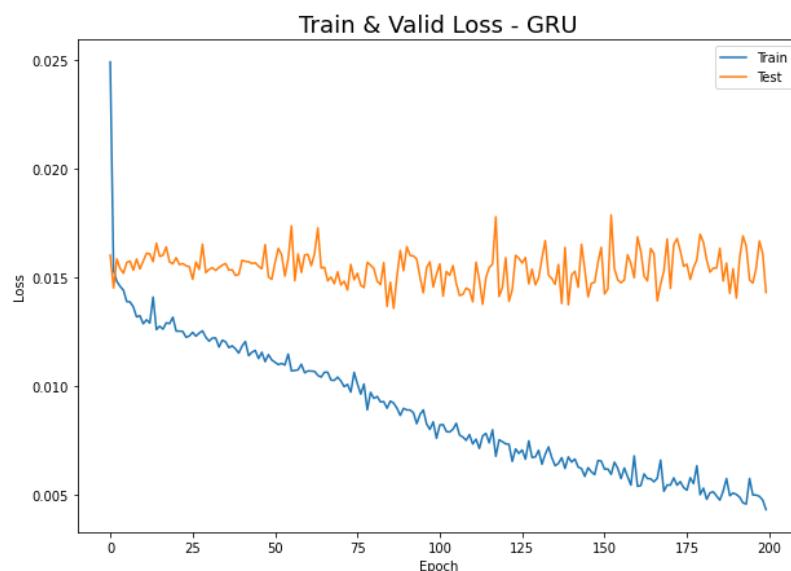
The total Training Time is Equal with ==> : 52.46602964481245 Sec.  

*****  

=====
```

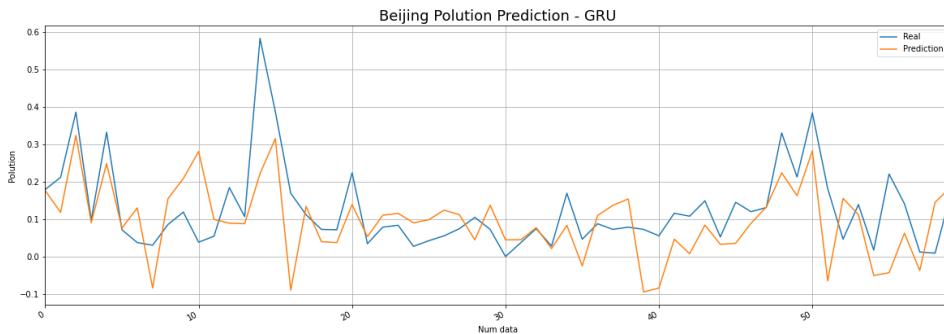
شکل 429: زمان اجرای آموزش در حالت مدل GRU و با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 430: نمودار Loss در حالت مدل GRU با MAE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌ها (15 عدد) مشاهده نمایید:



شکل 431: نمودار مقدار واقعی و پیش‌بینی شده در حالت GRU و AdaGrad

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 0.007699192843089501
>>>-----*****-----<<<
>>>-----*****-----<<<
#####

```

شکل 432: مقدار Loss به دست آمده برای داده‌های تست

نتیجه: از بررسی نمودار مقدار واقعی و پیش‌بینی شده و همچنین نمودار Loss به این نتیجه می‌رسیم که شبکه و مدل ما نسبت به حالت RNN عملکرد بسیار بهتری دارد و همچنین زمان آموزش آن نیز افزایش یافته است. اما نکته مهم این است که دقیق شبکه بهتر است نسبت به RNN و این موضوع را می‌توانیم به خوبی در نمودار مقایسه مقدار واقعی و پیش‌بینی ببینیم. اما خب همچنان دقیق شبکه نسبت به حالت ایده‌آل پایین است و آن نیز مانند همه حالت‌های قبل به دلیل کم بودن داده‌های آموزشی است. همچنین این شبکه در مقایسه با LSTM نیز عملکرد خوبی داشته و توانسته است دقیق‌های مشاهده‌ی را به دست بیاورد.

این مدل در حالت RMSprop عملکرد بهتری از Adam داشت.

✓ نتیجه‌گیری پایانی: از بررسی حاتهای هفتگی و ماهانه به این نتیجه می‌رسیم که حالت هفتگی به دلیل تعداد بیشتر داده‌ها عملکرد بهتری دارد و در کلا در هر دو حالت شبکه LSTM توانسته پیش‌بینی‌های دقیق‌تری را انجام دهد و همچنین بین دو حالت Adam و RMSprop نمی‌توان به طور قطعی گفت که کدام بهتر است و در بعضی موارد RMSprop بهتر بوده و در بعضی دیگر

E ♦ قسمت E

در این قسمت از ما خواسته شده است که اقدام به بررسی اضافه کردن لایه Dropout در شبکه بکنیم.

به این منظور من عملکرد اضافه کردن این لایه را روی حالت LSTM بررسی می‌کنم.

من شبکه LSTM با آن که Adam هست و همچنینتابع آن نیز MSE هست را در نظر می‌گیرم و به بررسی لایه Dropout بر روی آن می‌پردازم.

در ادامه می‌توانید طراحی مدل را مبتنی بر Dropout را مشاهده نمایید:

```
1 class LSTM(torch.nn.Module):
2     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1, dropout=0):
3         super(LSTM, self).__init__()
4         self.n_features = n_features
5         self.seq_len = seq_length
6
7         self.n_hidden = n_hidden_layers # number of hidden states
8         self.n_layers = n_layers # number of LSTM layers (stacked)
9         self.n_output = n_output
10
11        self.l_lstm = torch.nn.LSTM(input_size = n_features,
12                                     hidden_size = self.n_hidden,
13                                     num_layers = self.n_layers,
14                                     dropout=dropout,
15                                     batch_first = True)
16        # according to pytorch docs LSTM output is
17        # (batch_size, seq_len, num_directions * hidden_size)
18        # when considering batch_first = True
19        self.l_linear = torch.nn.Linear(self.n_hidden * self.seq_len, self.n_output)
20
21
22    def forward(self, x):
23        hidden_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requires_grad_()
24        cell_state = torch.zeros(self.n_layers, x.size(0), self.n_hidden).requires_grad_()
25        self.hidden = (hidden_state.detach(), cell_state.detach())
26
27        batch_size, seq_len, _ = x.size()
28
29        lstm_out, self.hidden = self.l_lstm(x, self.hidden)
30
31        # lstm_out(with batch_first = True) is
32        # (batch_size,seq_len,num_directions * hidden_size)
33        # for following linear layer we want to keep batch_size dimension and merge rest
34        # .contiguous() -> solves tensor compatibility error
35        x = lstm_out.contiguous().view(batch_size, -1)
36        # print("X shape :=> ", x.shape)
37        # out = self.l_linear(lstm_out[:, -1, :])
38        # print("Out Shape :=> ", lstm_out[:, -1, :].shape)
39        out = self.l_linear(x)
40
41        return out
```

شکل 433: اضافه کردن Dropout به مدل

سپس مدل طراحی شده را همراه با Optimizer و تابع Loss به صورت زیر تعریف می‌کنیم:

همچنین خوب است که به این نکته نیز توجه کنید که برای این که لایه Dropout عمل کند باید تعداد لایه‌های ما از یک بیشتر باشد، بنابراین باید تعداد لایه‌ها را به ۲ افزایش دهیم تا این لایه عمل کند. (بر طبق داکیومنت سایت Pytorch^۱)

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=2, dropout=0)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 434: تعریف مدل به همراه Loss و تابع Optimizer برای حالت بدون Dropout

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=2, dropout=0.5)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 435: تعریف مدل به همراه Loss و تابع Optimizer برای حالت همراه با Dropout

سپس اقدام به آموزش این دو شبکه طراحی شده می‌کنیم:

در ابتداء می‌توانید مقادیر Loss را مشاهده کنید:

```
step : 0 Train loss : 3.8181393481014914e-05 , Valid Loss : => 8.154279800752799e-05
***->>>-----<<<-***  

step : 1 Train loss : 2.8291751033005616e-05 , Valid Loss : => 7.708371834208568e-05
***->>>-----<<<-***  

step : 2 Train loss : 2.3657947421694795e-05 , Valid Loss : => 6.643335319434604e-05
***->>>-----<<<-***  

step : 3 Train loss : 2.2567737203401825e-05 , Valid Loss : => 4.876509735671182e-05
***->>>-----<<<-***  

step : 4 Train loss : 1.9683502486441285e-05 , Valid Loss : => 5.634933655771116e-05
***->>>-----<<<-***  

step : 5 Train loss : 2.024125326036786e-05 , Valid Loss : => 5.993242934346199e-05
***->>>-----<<<-***
```

شکل 436: مقادیر Loss در ۵ ایپاک اول در حالت بدون Dropout

```
step : 0 Train loss : 3.831009663020571e-05 , Valid Loss : => 8.108256369208297e-05
***->>>-----<<<-***  

step : 1 Train loss : 2.8544279226722815e-05 , Valid Loss : => 7.759670276815692e-05
***->>>-----<<<-***  

step : 2 Train loss : 2.4062650154034296e-05 , Valid Loss : => 6.684759104003509e-05
***->>>-----<<<-***  

step : 3 Train loss : 2.296148381040742e-05 , Valid Loss : => 5.110103900854786e-05
***->>>-----<<<-***  

step : 4 Train loss : 2.0153424725867806e-05 , Valid Loss : => 5.9941385872662064e-05
***->>>-----<<<-***  

step : 5 Train loss : 2.134929452246676e-05 , Valid Loss : => 6.173596150862674e-05
***->>>-----<<<-***
```

شکل 437: مقادیر Loss در ۵ ایپاک اول در حالت همراه با Dropout

¹ <https://pytorch.org/docs/stable/nn.html?highlight=lstm#torch.nn.LSTM>

```

***->>>-----<<<-***  

step : 93 Train loss : 3.901592490728944e-06 , Valid Loss : => 7.941730339856198e-06  

***->>>-----<<<-***  

step : 94 Train loss : 3.9700029893234995e-06 , Valid Loss : => 8.610165716769795e-06  

***->>>-----<<<-***  

step : 95 Train loss : 4.045206833203944e-06 , Valid Loss : => 9.333611912249277e-06  

***->>>-----<<<-***  

step : 96 Train loss : 4.1215853367854535e-06 , Valid Loss : => 1.0138943752584359e-05  

***->>>-----<<<-***  

step : 97 Train loss : 4.207227926235646e-06 , Valid Loss : => 1.1223433403453479e-05  

***->>>-----<<<-***  

step : 98 Train loss : 4.319899289597136e-06 , Valid Loss : => 1.1542080765745292e-05  

***->>>-----<<<-***  

step : 99 Train loss : 4.346101333309586e-06 , Valid Loss : => 1.3065498710299532e-05  

***->>>-----<<<-***
```

شکل 438: مقادیر Loss در 5 ایپاک آخر در حالت بدون Dropout

```

***->>>-----<<<-***  

step : 93 Train loss : 3.816857897133256e-06 , Valid Loss : => 5.615281183660651e-06  

***->>>-----<<<-***  

step : 94 Train loss : 3.909731391468086e-06 , Valid Loss : => 5.726149975089357e-06  

***->>>-----<<<-***  

step : 95 Train loss : 3.833522906643338e-06 , Valid Loss : => 6.437480090729271e-06  

***->>>-----<<<-***  

step : 96 Train loss : 3.945023435517214e-06 , Valid Loss : => 6.730240575658778e-06  

***->>>-----<<<-***  

step : 97 Train loss : 3.929749648280752e-06 , Valid Loss : => 7.224076446921875e-06  

***->>>-----<<<-***  

step : 98 Train loss : 4.042149125598371e-06 , Valid Loss : => 8.238630379006888e-06  

***->>>-----<<<-***  

step : 99 Train loss : 4.073237002982448e-06 , Valid Loss : => 8.21975646734548e-06  

***->>>-----<<<-***
```

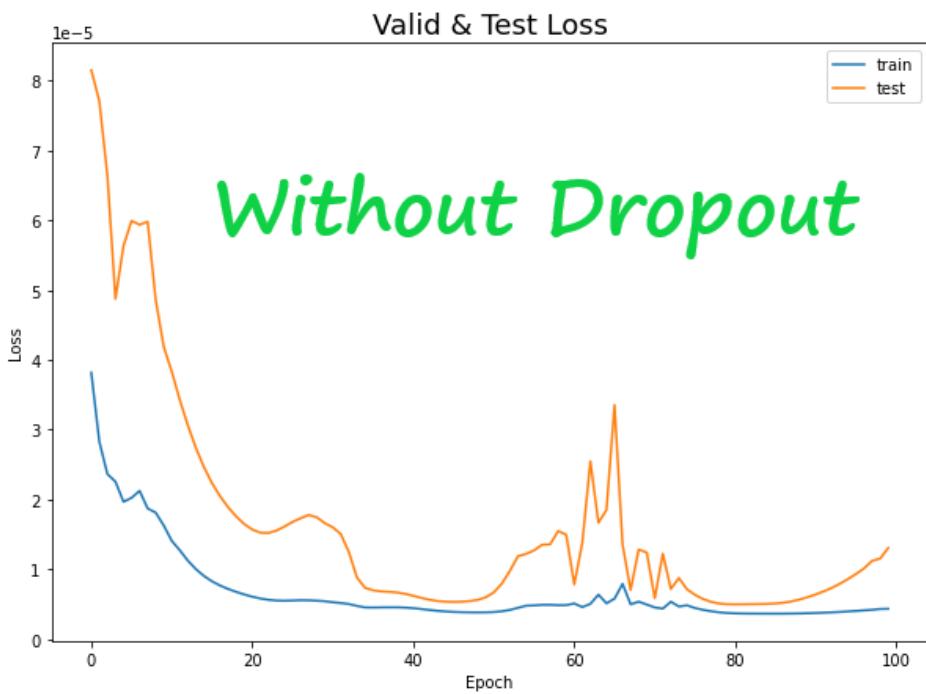
شکل 439: مقادیر Loss در 5 ایپاک آخر در حالت همراه با Dropout

در ادامه نیز می‌توانید زمان آموزش در دو شبکه را مشاهده کنید:

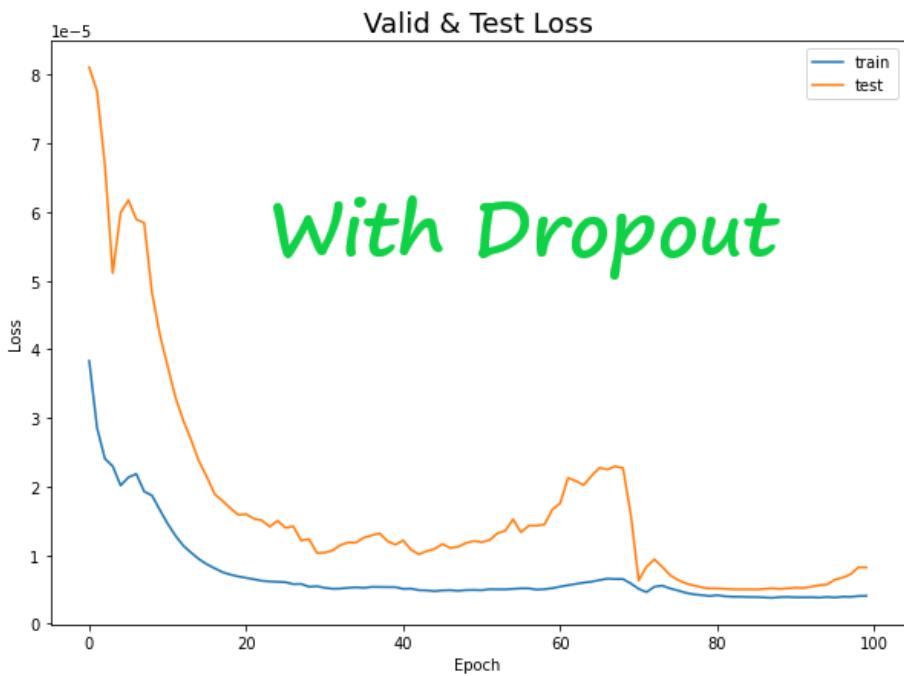
جدول 6: مقایسه زمان‌های آموزش در دو حالت با Dropout و بدون آن

Model Type	Train Time
LSTM - With Dropout	1812.784368276596 Sec.
LSTM - Without Dropout	1768.9027273654938 Sec.

اکنون در ادامه به سراغ مقایسه نمودار Loss در دو حالت می‌رویم:



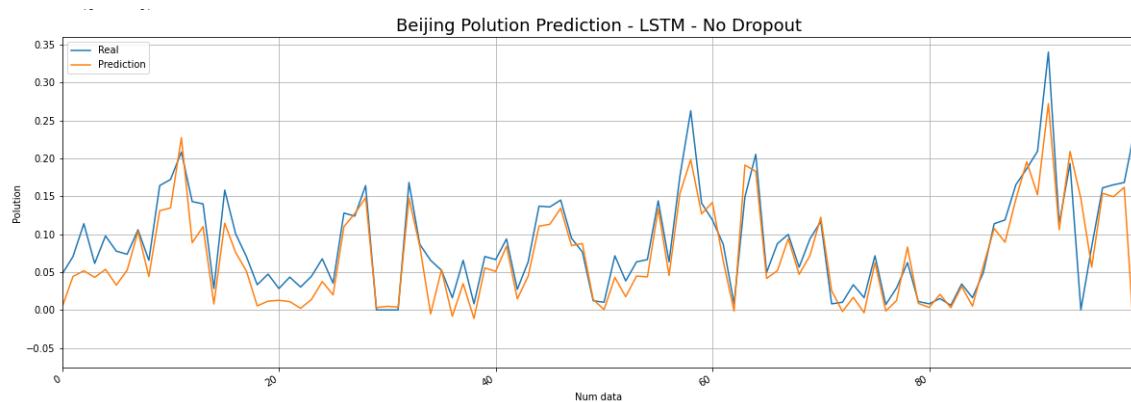
شکل 440: نمودار Loss در حالت بدون Dropout



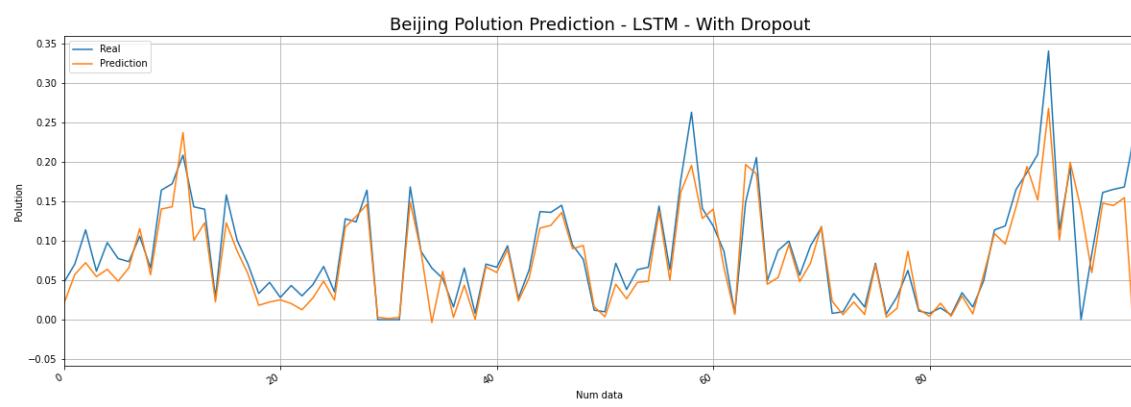
شکل 441: نمودار Loss در حالت همراه با Dropout

از مشاهده دو نمودار فوق به این نتیجه می‌رسیم که استفاده از Dropout توانسته است که عملکرد شبکه را بهبود ببخشد و از پیش آمدن Overfitting نیز جلوگیری کند و نمودار نوسانات کمتری دارد.

حال وقت آن است که به سراغ بررسی مقادیر واقعی و پیش‌بینی شده برویم:



شکل 442: مقادیر واقعی و پیش‌بینی شده در حالت بدون Dropout



شکل 443: مقادیر واقعی و پیش‌بینی شده در حالت همراه با Dropout

در ادامه نیز به بررسی مقدار Loss به دست آمده برای داده‌های تست می‌پردازیم:

جدول 7: مقادیر Loss برای حالت همراه با Dropout و بدون آن

Model	Test Loss
LSTM – without Dropout	1.0029598701900492e-05
LSTM – with Dropout	6.508351070806384e-06

✓ با مشاهده جدول بالا مشاهده می‌کنیم که مقدار Loss برای داده‌های تست در حالت همراه با Dropout کمتر شده است و بنابراین این مدل دقیق‌تری دارد.

با در نظر گرفتن تمامی جوانب به نظر می‌رسد که اضافه کردن Dropout تاثیر خوبی در فرآیند آموزش و همچنین دقت شبکه ما داشته است و توانسته است که عملکرد آن را به طور محسوسی بهبود ببخشد.

✓ فایل‌های مربوط به این سوال را می‌توانید که در دو فایل و [MiniProj_LSTM_Adam_MSE_Q1_PartE_NoDropout_Pytorch.ipynb](#) و [MiniProj_LSTM_Adam_MSE_Q1_PartE_Dropout_Pytorch.ipynb](#) در پوشه آپلود شده مشاهده نمایید.

✓ فایل‌های مربوط به این قسمت را می‌توانید که در پوشه Q1 PartD موجود در فایل‌های آپلود شده مشاهده نمایید. و سپس پس از آن کدهای حالت هفتگی را می‌توانید در داخل پوشه Weekly و کدهای حالت ماهانه را در پوشه Monthly مشاهده نمایید.

❖ قسمت F ❖

در این قسمت از ما خواسته شده است که با اضافه کردن لایه Fusion اقدام به ترکیب نتایج 3 شبکه کنیم.

اصولاً هدف از اضافه کردن این لایه این است که ما بتوانیم مشکل کم بودن داده‌ها را کاهش دهیم و رفع کنیم. و با اضافه کردن این لایه شبکه ما Robust‌تر می‌شود و با انجام این کار شبکه ما اگر داده‌های کمی داشته باشد، می‌تواند با اضافه کردن این لایه Performance بهتری داشته باشد.

برای پیاده‌سازی این روش باید ابتدا در تابع‌هایی که برای ساختن دیتابست درست کرده بودم برای این که سه حالت روزانه، هفتگی و ماهانه را بسازند تغییراتی را ایجاد کنم. که در ادامه این سه تابع را قرار می‌دهم:

```

6  def select_week(sequences, n_samples=250):
7      X, y = list(), list()
8      rand_hour = 360 #randint(0, 24)
9      for i in range(0, n_samples):
10         start_ix = rand_hour #+ 168 * i # 168 : Week hours!
11         idxs = []
12         for j in range(0, 7):
13             if j <=5:
14                 idx = start_ix + (j * 24) # Add different days in week
15                 # print("Id x Week:=> ", idx)
16                 idxs.append(idx)
17             if j == 6: # Target
18                 idy = start_ix + (j * 24)
19                 seq_x = sequences[idxs, :]
20                 seq_y = sequences[idy, 0]
21                 y.append(seq_y)
22                 X.append(seq_x)
23                 # print("Id y Week:=> ", idy)
24                 rand_hour += 1
25         return X, y

```

شکل 444: تابع سازنده دیتاست حالت هفتگی

```

28 def select_month(sequences, n_samples=250):
29     X, y = list(), list()
30     rand_hour = 0 #randint(0, 24)
31     rand_day = 0 #randint(0, 7)
32     for i in range(0, n_samples):
33         start_ix = rand_hour #+ rand_day*24 + 672 * i # 168 : Week hours!
34         idxs = []
35         for j in range(0, 4):
36             if j <=2:
37                 idx = start_ix + (j * 168) # Add different weeks
38                 idxs.append(idx)
39                 # print("Id x Month:=> ", idx)
40                 # print("Hello")
41             if j == 3: # Target
42                 idy = start_ix + (j * 168)
43                 seq_x = sequences[idxs, :]
44                 seq_y = sequences[idy, 0]
45                 # print("Id y Month:=> ", idy)
46                 y.append(seq_y)
47                 X.append(seq_x)
48
49         rand_hour += 1
50
51     return X, y

```

شکل 445: تابع سازنده دیتاست حالت ماهانه

```

54 # split a multivariate sequence into samples
55 def split_sequences(sequences, n_steps=11, n_samples=12000, start_from=493):
56     X, y = list(), list()
57     for i in range(start_from, (start_from + n_samples)):
58         # find the end of this pattern
59         end_ix = i + n_steps
60         # check if we are beyond the dataset
61         # gather input and output parts of the pattern
62         seq_x = sequences[i:end_ix, :]
63         seq_y = sequences[end_ix, 0]
64         y.append(seq_y)
65         X.append(seq_x)
66
67     return array(X), array(y)

```

شکل 446: تابع سازنده دیتاست حالت روزانه

سپس به سراغ آماده‌سازی دیتاهای به دست آمده از این توابع برای دسته‌بندی‌های تست و تربین رفتم و پس از این کار به سراغ طراحی مدل برای این حالت که بتواند لایه fusion را بپذیرد رفتم و این کار را AvgPooling به دو صورت انجام دادم یکی به حالت میانگین گیری عادی و دیگری با حالت استفاده از لایه :pytorch خود

• حالت اول: میانگین گیری عادی:

```

1. class LSTM(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
3.                  33, n_layers=1, dropout=0):
4.         super(LSTM, self).__init__()
5.         self.n_features = n_features
6.         self.seq_len = seq_length
7.
8.         self.n_hidden = n_hidden_layers # number of hidden states
9.         self.n_layers = n_layers # number of LSTM layers (stacked)
10.        self.n_output = n_output
11.
12.        self.l_lstm = torch.nn.LSTM(input_size = n_features,
13.                                     hidden_size = self.n_hidden,
14.                                     num_layers = self.n_layers,
15.                                     dropout=dropout,
16.                                     batch_first = True)
17.
18.        self.lstm2 = torch.nn.LSTM(input_size = n_features,
19.                                   hidden_size = self.n_hidden,
20.                                   num_layers = self.n_layers,
21.                                   dropout=dropout,
22.                                   batch_first = True)
23.
24.        self.lstm3 = torch.nn.LSTM(input_size = n_features,
25.                                   hidden_size = self.n_hidden,
26.                                   num_layers = self.n_layers,
27.                                   dropout=dropout,
28.                                   batch_first = True)
29.
30.        self.l_linear = torch.nn.Linear(self.n_hidden * self.seq_len, self.n_outp
ut)

```

```

30.         self.linear2 = torch.nn.Linear(self.n_hidden * 6, self.n_output)
31.         self.linear3 = torch.nn.Linear(self.n_hidden * 3, self.n_output)
32.         self.avgpool = nn.AvgPool1d(kernel_size=1, stride=3, padding=0)
33.
34.
35.     def forward(self, x1, x2, x3):
36.         hidden_state = torch.zeros(self.n_layers, x1.size(0), self.n_hidden).requires_grad_()
37.         cell_state = torch.zeros(self.n_layers, x1.size(0), self.n_hidden).requires_grad_()
38.         self.hidden = (hidden_state.detach(), cell_state.detach())
39.         batch_size, seq_len, _ = x1.size()
40.         lstm_out, self.hidden = self.l_lstm(x1, self.hidden)
41.         x = lstm_out.contiguous().view(batch_size, -1)
42.         out1 = self.l_linear(x)
43.
44.         hidden_state = torch.zeros(self.n_layers, x2.size(0), self.n_hidden).requires_grad_()
45.         cell_state = torch.zeros(self.n_layers, x2.size(0), self.n_hidden).requires_grad_()
46.         self.hidden = (hidden_state.detach(), cell_state.detach())
47.         batch_size, seq_len, _ = x2.size()
48.         lstm_out, self.hidden = self.l_lstm(x2, self.hidden)
49.         x = lstm_out.contiguous().view(batch_size, -1)
50.         out2 = self.linear2(x)
51.
52.         hidden_state = torch.zeros(self.n_layers, x3.size(0), self.n_hidden).requires_grad_()
53.         cell_state = torch.zeros(self.n_layers, x3.size(0), self.n_hidden).requires_grad_()
54.         self.hidden = (hidden_state.detach(), cell_state.detach())
55.         batch_size, seq_len, _ = x3.size()
56.         lstm_out, self.hidden = self.l_lstm(x3, self.hidden)
57.         x = lstm_out.contiguous().view(batch_size, -1)
58.         out3 = self.linear3(x)
59.
60.         x3 = (out1 + out2 + out3) / 3
61.         # print("X3 shape :=> ", x3.shape)
62.
63.         out = x3
64.
65.     return out

```

• حالت دوم : استفاده از AvgPooling

```

1. class LSTM(torch.nn.Module):
2.     def __init__(self, n_features=8, n_output=1, seq_length=11, n_hidden_layers=2
33, n_layers=1, dropout=0):
3.         super(LSTM, self).__init__()
4.         self.n_features = n_features
5.         self.seq_len = seq_length
6.
7.         self.n_hidden = n_hidden_layers # number of hidden states
8.         self.n_layers = n_layers # number of LSTM layers (stacked)
9.         self.n_output = n_output
10.
11.        self.l_lstm = torch.nn.LSTM(input_size = n_features,
12.                               hidden_size = self.n_hidden,
13.                               num_layers = self.n_layers,
14.                               dropout=dropout,

```

```

15.                         batch_first = True)
16.
17.         self.lstm2 = torch.nn.LSTM(input_size = n_features,
18.                                     hidden_size = self.n_hidden,
19.                                     num_layers = self.n_layers,
20.                                     dropout=dropout,
21.                                     batch_first = True)
22.
23.         self.lstm3 = torch.nn.LSTM(input_size = n_features,
24.                                     hidden_size = self.n_hidden,
25.                                     num_layers = self.n_layers,
26.                                     dropout=dropout,
27.                                     batch_first = True)
28.
29.         self.l_linear = torch.nn.Linear(self.n_hidden * self.seq_len, self.n_output)
30.         self.linear2 = torch.nn.Linear(self.n_hidden * 6, self.n_output)
31.         self.linear3 = torch.nn.Linear(self.n_hidden * 3, self.n_output)
32.         self.avgpool = nn.AvgPool1d(kernel_size=3, stride=3, padding=0)
33.
34.
35.     def forward(self, x1, x2, x3):
36.         hidden_state = torch.zeros(self.n_layers, x1.size(0), self.n_hidden).requires_grad_()
37.         cell_state = torch.zeros(self.n_layers, x1.size(0), self.n_hidden).requires_grad_()
38.         self.hidden = (hidden_state.detach(), cell_state.detach())
39.         batch_size, seq_len, _ = x1.size()
40.         lstm_out, self.hidden = self.l_lstm(x1, self.hidden)
41.         x = lstm_out.contiguous().view(batch_size, -1)
42.         out1 = self.l_linear(x)
43.
44.         hidden_state = torch.zeros(self.n_layers, x2.size(0), self.n_hidden).requires_grad_()
45.         cell_state = torch.zeros(self.n_layers, x2.size(0), self.n_hidden).requires_grad_()
46.         self.hidden = (hidden_state.detach(), cell_state.detach())
47.         batch_size, seq_len, _ = x2.size()
48.         lstm_out, self.hidden = self.lstm2(x2, self.hidden)
49.         x = lstm_out.contiguous().view(batch_size, -1)
50.         out2 = self.linear2(x)
51.
52.         hidden_state = torch.zeros(self.n_layers, x3.size(0), self.n_hidden).requires_grad_()
53.         cell_state = torch.zeros(self.n_layers, x3.size(0), self.n_hidden).requires_grad_()
54.         self.hidden = (hidden_state.detach(), cell_state.detach())
55.         batch_size, seq_len, _ = x3.size()
56.         lstm_out, self.hidden = self.lstm3(x3, self.hidden)
57.         x = lstm_out.contiguous().view(batch_size, -1)
58.         out3 = self.linear3(x)
59.
60.         x3 = torch.cat((out1, out2, out3), dim=1).view(1, 1, batch_size*3)
61.
62.         out = self.avgpool(x3)
63.         # print("out size => ", out.shape)
64.
65.     return out

```

در هر دو مدل بالا به این صورت عمل کردم که سه تا لایه با ورودی‌های مختلف از نوع LSTM که بهترین عملکرد را داشت تعریف کردم و سپس خروجی‌های همه این‌ها را با هم Concatenate کردم و از آن‌ها میانگین گرفتم.

حال پس از طراحی مدل‌ها به سراغ تعریف Optimization و تابع Loss رفتم:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 447: تعریف تابع Optimizer و Loss

سپس تابع آموزش را نیز به صورت زیر باید تغییر می‌دادم که بتواند سه ورودی را به مدل من بدهد و خروجی آن را دریافت کند و بتواند با استفاده از یک Batch Size = 200 روی تمامی دیتاست‌ها حرکت کند:

```
1. import time
2. start_time = time.time()
3.
4. # train_X, train_y
5. epochs = 100
6. model.train()
7. batch_size = 200
8. running_loss_history = []
9. val_running_loss_history = []
10. for epoch in range(epochs):
11.     running_loss = 0.0
12.     val_running_loss = 0.0
13.     model.train()
14.     for b in range(0, len(x_day_train), batch_size):
15.         inpt1 = x_day_train[b:b+batch_size, :, :]
16.         target1 = y_day_train[b:b+batch_size]
17.
18.         inpt2 = x_week_train[b:b+batch_size, :, :]
19.         target2 = y_week_train[b:b+batch_size]
20.
21.         inpt3 = x_month_train[b:b+batch_size, :, :]
22.         target3 = y_month_train[b:b+batch_size]
23.
24.         # print("Input Shape :=> ", inpt.shape)
25.
26.         x_batch1 = torch.tensor(inpt1, dtype=torch.float32)
27.         y_batch1 = torch.tensor(target1, dtype=torch.float32)
28.
29.         x_batch2 = torch.tensor(inpt2, dtype=torch.float32)
30.         y_batch2 = torch.tensor(target2, dtype=torch.float32)
31.
32.         x_batch3 = torch.tensor(inpt3, dtype=torch.float32)
33.         y_batch3 = torch.tensor(target3, dtype=torch.float32)
34.
```

```

35.     output = model(x_batch1, x_batch2, x_batch3)
36.     loss = criterion(output.view(-1), y_batch1)
37.
38.     running_loss += loss.item()
39.
40.     loss.backward()
41.     optimizer.step()
42.     optimizer.zero_grad()
43.
44. else:
45.
46.     with torch.no_grad(): # it will temporarorly set all the required grad f
        lags to be false
47.         model.eval()
48.         for b in range(0, len(x_day_test), batch_size):
49.             inpt1 = x_day_test[b:b+batch_size, :, :]
50.             target1 = y_day_test[b:b+batch_size]
51.
52.             inpt2 = x_week_test[b:b+batch_size, :, :]
53.             target2 = y_week_test[b:b+batch_size]
54.
55.             inpt3 = x_month_test[b:b+batch_size, :, :]
56.             target3 = y_month_test[b:b+batch_size]
57.
58.             x_batch1 = torch.tensor(inpt1, dtype=torch.float32)
59.             y_batch1 = torch.tensor(target1, dtype=torch.float32)
60.
61.             x_batch2 = torch.tensor(inpt2, dtype=torch.float32)
62.             y_batch2 = torch.tensor(target2, dtype=torch.float32)
63.
64.             x_batch3 = torch.tensor(inpt3, dtype=torch.float32)
65.             y_batch3 = torch.tensor(target3, dtype=torch.float32)
66.
67.             output_test = model(x_batch1, x_batch2, x_batch3)
68.             loss_test = criterion(output_test.view(-1), y_batch1)
69.
70.             val_running_loss += loss_test.item()
71.
72.     val_epoch_loss = val_running_loss / len(x_day_test)
73.     val_running_loss_history.append(val_epoch_loss)
74.     epoch_loss = running_loss / len(x_day_train)
75.     running_loss_history.append(epoch_loss)
76.     print('step : ', epoch, ' Train loss : ', epoch_loss, ', Valid Loss : => '
    , val_epoch_loss)
77.     print("****->>>-----<<<-***")
78.
79. total_time = time.time() - start_time
80. print("=====")
81. print("*****")
82. print("The total Training Time is Equal with ==> : {0} Sec.".format(total_time))
83. print("*****")
84. print("=====")

```

سپس پس از آموزش این شبکه من نتایج آن را در زیر قرار می‌دهم. لازم است به این نکته اشاره کنم که دو حالتی که پیشتر به آن اشاره کردم تقریباً نتایج مشابهی را دادند به همین دلیل من تنها نتایج حالت استفاده از Average Pooling را قرار می‌دهم:

ابتدا نتایج مربوط به Loss را قرار می‌دهم:

```
step : 0 Train loss : 3.51905343489273e-05 , Valid Loss : => 0.00021972281392663718
***->>>-----<<<-***  
step : 1 Train loss : 4.0000458577803026e-05 , Valid Loss : => 3.4736850531771776e-05
***->>>-----<<<-***  
step : 2 Train loss : 3.3358957014839956e-05 , Valid Loss : => 1.8175863680274537e-05
***->>>-----<<<-***  
step : 3 Train loss : 1.8889483810333466e-05 , Valid Loss : => 2.0077199093066155e-05
***->>>-----<<<-***  
step : 4 Train loss : 1.8793565444260216e-05 , Valid Loss : => 1.432152701697002e-05
***->>>-----<<<-***  
step : 5 Train loss : 1.5019071275471813e-05 , Valid Loss : => 1.24088148586452e-05
***->>>-----<<<-***
```

شکل 448: مقادیر مربوط به 5 ایپاک اول آموزش

```
***->>>-----<<<-***  
step : 93 Train loss : 4.135442347399442e-06 , Valid Loss : => 3.6793151424111178e-06
***->>>-----<<<-***  
step : 94 Train loss : 4.0151922396573895e-06 , Valid Loss : => 3.3776658722975603e-06
***->>>-----<<<-***  
step : 95 Train loss : 3.969887983219922e-06 , Valid Loss : => 3.3874990646533357e-06
***->>>-----<<<-***  
step : 96 Train loss : 3.957583190640434e-06 , Valid Loss : => 3.329248909722082e-06
***->>>-----<<<-***  
step : 97 Train loss : 3.954304209737866e-06 , Valid Loss : => 3.3393934718333185e-06
***->>>-----<<<-***  
step : 98 Train loss : 3.944170851253956e-06 , Valid Loss : => 3.3187334217169944e-06
***->>>-----<<<-***  
step : 99 Train loss : 3.943177394157155e-06 , Valid Loss : => 3.3505000901641326e-06
***->>>-----<<<-***
```

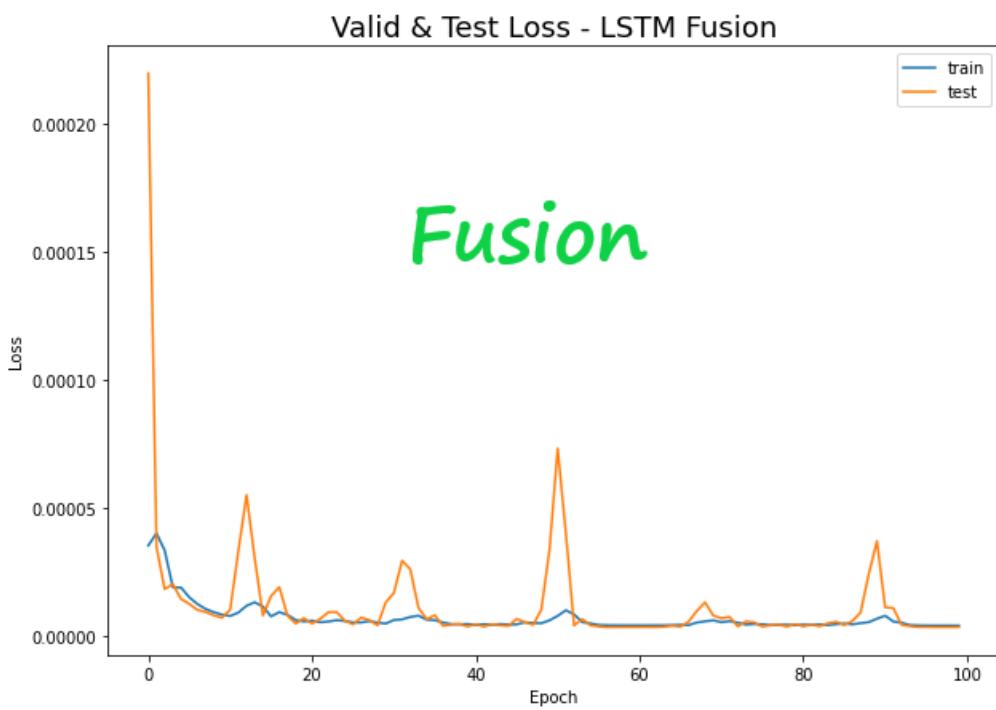
شکل 449: مقادیر مربوط به 5 ایپاک آخر آموزش

سپس زمان مربوط به آموزش را می‌توانید در ادامه مشاهده کنید:

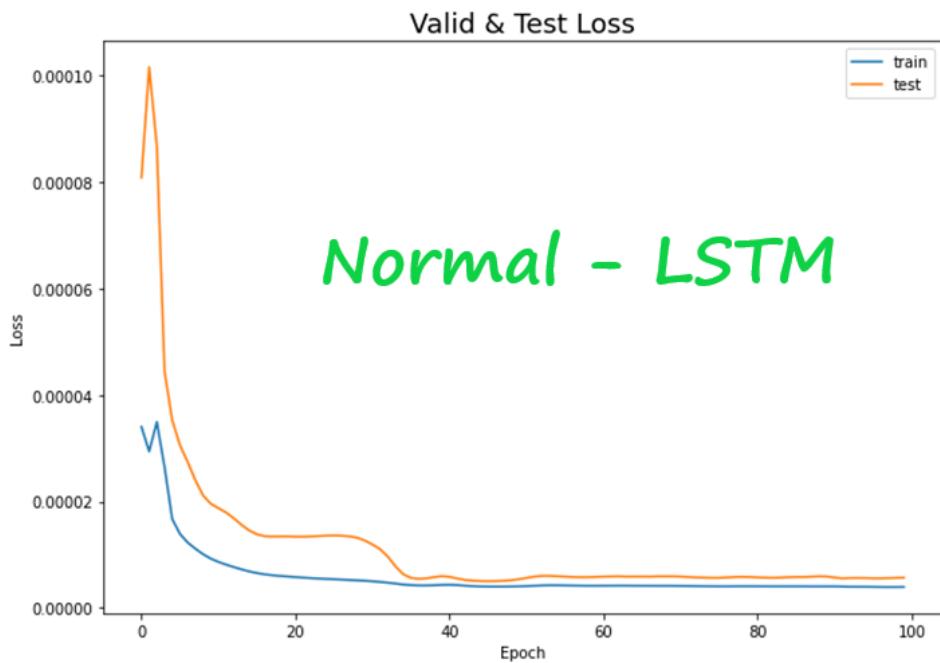
```
=====
*****  
The total Training Time is Equal with ==> : 2501.6557297706604 Sec.  
*****  
=====
```

شکل 450: زمان آموزش مدل دارای لایه Fusion

تا همین جای کار مشاهده می‌کنیم که زمان اجرای این حالت در مقایسه با حالتی که از LSTM تنها استفاده می‌کردیم حدود 3.5 برابری است.

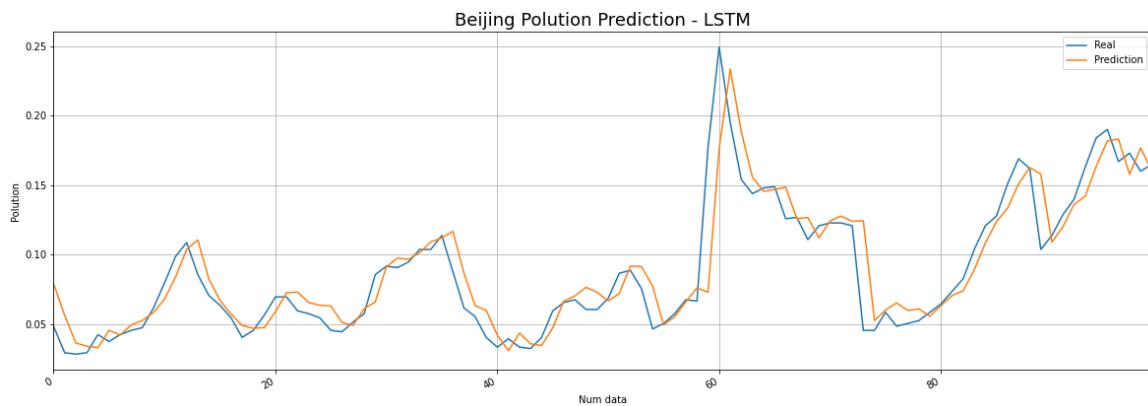


شکل 451: نمودار مربوط به Loss به دست آمده در حالت Fusion



شکل 452: نمودار مربوط به Loss به دست آمده در حالت LSTM معمولی

سپس در ادامه نیز می‌توانید نمودار حالت پیش‌بینی شده و واقعی را مشاهده نمایید:



شکل 453: نمودار مقادیر واقعی و پیش‌بینی شده در حالت Fusion

سپس در ادامه نیز می‌توانید مقدار Loss به دست آمده برای حالت استفاده از Fusion را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 1.8115157727152109e-06
>>>-----*****-----<<<
>>Fusion<<<
#####
```

شکل 454: مقدار Loss به دست آمده در حالت Fusion بر روی داده‌های Test

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.854833362818075e-06
>>>-----*****-----<<<
>>>Normal LSTM<<<
#####
```

شکل 455: مقدار Loss به دست آمده در حالت LSTM معمولی بر روی داده‌های Test

همان‌طور که در دو شکل بالا مشخص است استفاده از لایه Fusion باعث شده است که ما بتوانیم مقدار Loss را به حدود $\frac{1}{4}$ کاهش دهیم و این نشان می‌دهد که دقیق‌ترین مدل ما در حالت Fusion به مقدار خوبی افزایش می‌یابد.

✓ لازم به ذکر است که هر دو حالت Adam Optimizer آن‌ها است و همچنین

تابع Loss هر دو MSE است.

➤ نتیجه‌گیری نهایی: وقتی که ما از Fusion استفاده می‌کنیم مدل ما از نظر زمان آموزش افزایش چشمگیری پیدا می‌کند.

➤ اما در عوض از نظر دقیقت می‌توانیم ما بهبودهای خوبی را در حالتی که از این لایه استفاده می‌کنیم، مشاهده کنیم.

➤ دو فایل مربوط به دو پیاده سازی مربوط به این سوال وجود دارد به نامهای MiniProj_LSTM_Adam_MSE_Q1_PartF_Pytorch_AVGPool.ipynb و نیز MiniProj_LSTM_Adam_MSE_Q1_PartF_Pytorch_Normal.ipynb قرار دارد.

❖ قسمت G:

در این قسمت از ما خواسته شده است که از میان 7 ویژگی‌ای که برای آموزش شبکه استفاده می‌کردیم (به غیر از خود آلوگی) دو ویژگی را انتخاب کنیم و با این دو ویژگی عملکرد شبکه را بسنجیم.

خب در این حالت معقول است که بیاییم و دو تا از مهمترین ویژگی‌ها را انتخاب کنیم که بیشترین اثر را در نتیجه خروجی ما دارد. و ویژگی‌هایی را که کمترین اثر را در خروجی ما دارند را حذف کنیم.

برای انجام این کار روش‌های متفاوتی وجود دارد.

یکی از این روش‌ها این است که بیاییم و اثر هر کدام از ویژگی‌ها را بر روی خروجی و پیش‌بینی نهایی شبکه را بسنجیم. به این صورت که تک‌تک ویژگی‌ها را حذف کنیم و ببینیم که اثر حذف هر کدام از این ویژگی‌ها بر روی خروجی و دقیقت نهایی شبکه چی هست و در نهایت آن ویژگی‌هایی که حذف آن‌ها بیشترین اثر را بر روی دقیقت خروجی داشته باشد از همه مهم‌تر هستند.

این روش معرفی شده خب یکی از ساده‌ترین روش‌ها است. اما روش پیچیده‌تر و دقیق‌تری را می‌خواهم که در ادامه معرفی کنم:

اسم این روش Gradient Importance یا اهمیت گرادیانی هست. ایده اصلی این روش این است که بیاییم و تاثیر هر کدام از ویژگی‌های ورودی را بر روی پیش‌بینی نهایی مشخص کنیم. اثر گذاری و یا مشارکت هر ویژگی در خروجی در این حالت به وسیله مقدار گرادیان‌هایی که از عملیات‌های تفکیک شده داده‌های ورودی به دست می‌آید، محاسبه می‌شود.

برای انجام و پیاده سازی این روش باید که سه مرحله را طی کنیم:

1. مرحله اول: گرادیان ورودی‌ها را محاسبه می‌کنیم.
2. مرحله دوم: گرادیان ورودی‌ها را به ازای هر کدام از ویژگی‌ها محاسبه می‌کنیم.
3. مرحله سوم: در این قسمت اثر هر کدام از ویژگی‌ها در طول زمان به وسیله گرادیان‌ها محاسبه می‌کنیم و ویژگی که بیشتری اثر را دارد به این صورت مشخص می‌شود.

نتیجه به دست آمده به راحتی تحلیل می‌شود و همبستگی و Correlation هر کدام از ویژگی‌ها را با مقادیر Target را نشان می‌دهد. هر چه که مقدار آن بیشتر باشد نیز نشان دهنده اثر بیشتر آن ویژگی بر روی پیش‌بینی Target ما هست.

سپس در ادامه می‌توانید محاسبات برای به دست آوردن اهمیت گرادیانی را مشاهده نمایید:

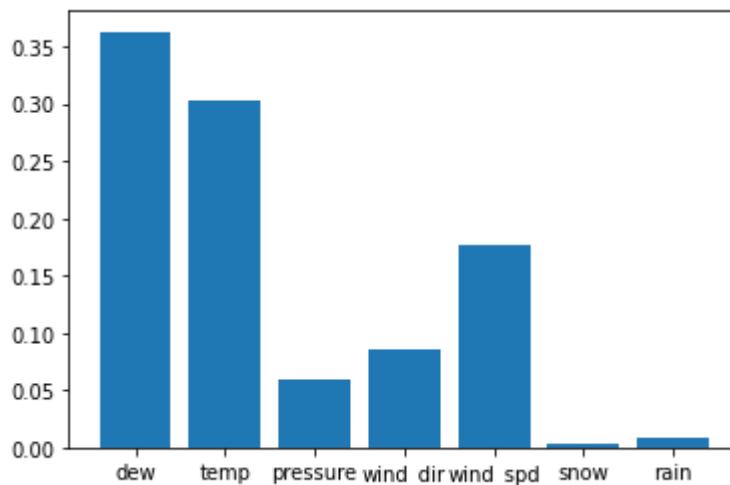
```
1  ### PREDICTION ERROR ON TEST DATA ###
2  gb = GradientBoostingRegressor(n_estimators=300)
3  gb.fit(seq_x, seq_y.ravel())
4  # mean_absolute_error(seq_y, gb.predict(seq_x))

GradientBoostingRegressor(alpha=0.9, ccp_alpha=0.0, criterion='friedman_mse',
                           init=None, learning_rate=0.1, loss='ls', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=300,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=None, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0, warm_start=False)

1  ### FEATURE IMPORTANCES REPORT ###
2  print(seq_x.shape[1])
3  plt.bar(range(seq_x.shape[1]), gb.feature_importances_)
4  plt.xticks(range(seq_x.shape[1]), ['dew','temp','pressure', 'wind_dir', 'wind_spd', 'snow', 'rain'])
5  np.set_printoptions(False)
```

شکل 456: نحوه محاسبه و به دست آوردن اهمیت گرادیانی

در ادامه نیز می‌توانید نتیجه محاسبات را مشاهده نمایید:



شکل 457: نتیجه اهمیت گرادیانی محاسبه شده

► از نمودار بالا به این نتیجه می‌رسیم که بیشترین اثر و وابستگی میان خروجی و Target ها و پارامترها در پارامترهای `dew` و `temp` هست.

روش دیگری نیز وجود دارد و آن نیز استفاده از ماتریس همبستگی هست. که اصول آن نیز به این صورت است که پیوندهای ارتباطی میان داده‌ها را به وسیله روش Pearson به ما نشان می‌دهد. همبستگی Pearson که به نام کارل پیرسون نامگذاری شده است می‌تواند برای خلاصه‌سازی قدرت ارتباط خطی میان دو نمونه داده استفاده شود.

ضریب همبستگی پیرسون به وسیله کواریانس دو نمونه داده تقسیم بر حاصل ضرب standard deviation دو نمونه محاسبه می‌شود که نرمال شده کواریانس میان دو نمونه داده است تا این که به ما یک نتیجه قابل تفسیر بدهد. در ادامه می‌توانید نحوه محاسبه آن را برای دو داده مشاهده کنید:

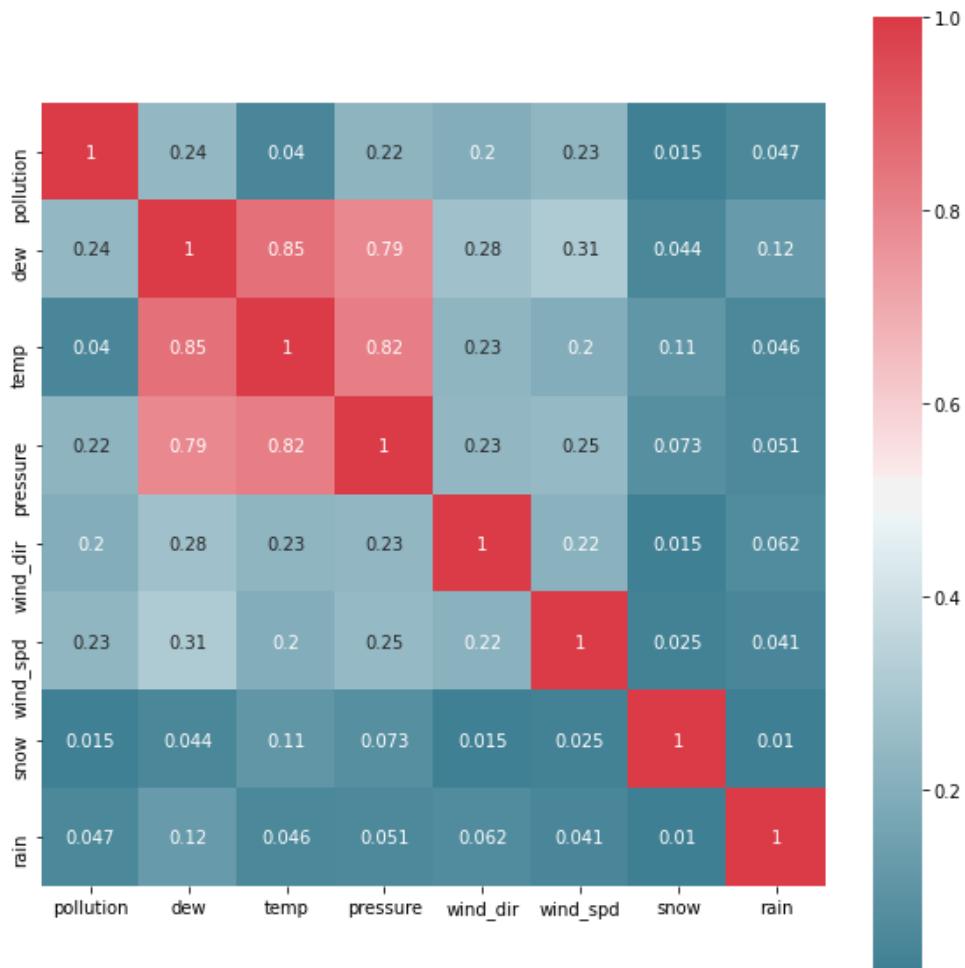
```
1 Pearson's correlation coefficient = covariance(X, Y) / (stdv(X) * stdv(Y))
```

شکل 458: محاسبه همبستگی پیرسون برای دو نمونه داده

نتیجه به دست آمده از این محاسبه می‌تواند به عنوان ضریب همبستگی برای درک بهتر روابط میان دو داده مورد استفاده قرار بگیرد.

این محاسبه مقداری بین 1 و -1- را به ما باز می‌گرداند که به ترتیب به معنی بیشترین همبستگی مثبت و بیشترین همبستگی منفی هست. و مقدار 0 نیز به این معنی است که همبستگی‌ای وجود ندارد.

در ادامه اقدام به به دست آوردن ماتریس مربوط به این همبستگی می‌کنم:



شکل 459: ماتریس همبستگی به دست آمده

در ادامه نیز می‌توانید پارامترهایی که بیشترین مقدار همبستگی را با پارامتر آلودگی هوا داشته‌اند را مشاهده کنید.



شکل ۴۶۰: پارامترهای دارای بیشتری میزان همبستگی با پارامتر آلودگی

► از مشاهده شکل بالا به این نتیجه می‌رسیم که پارامتر dew و wind speed از بقیه همبستگی بیشتری با آلودگی یا pollution دارند. و پس از این دو پارامتر، wind_dir بیشترین همبستگی را دارد.

در قسمت بعدی سوال من به پیاده سازی این دو روش یعنی در حقیقت دو حالتی که از دو روش پیرسون و نیز روش اهمیت گرادیانی به دست آوردهام می‌پردازم.

:H قسمت ❖

✓ پیاده‌سازی حالت ویژگی‌های به دست آمده از اهمیت گرادیانی Gradient) (Important

```
6 # split a multivariate sequence into samples
7 def split_sequences(sequences, n_steps=11, n_samples=12000, start_from=0):
8     X, y = list(), list()
9     for i in range(start_from, (start_from + n_samples)):
10         # find the end of this pattern
11         end_ix = i + n_steps
12         # check if we are beyond the dataset
13         # gather input and output parts of the pattern
14         seq_x = sequences[i:end_ix, [0, 1, 2]] [0, 1, 2]
15         seq_y = sequences[end_ix, 0]
16         y.append(seq_y)
17         X.append(seq_x)
18
19     return array(X), array(y)
```

شکل 461: تغییر کد برای انتخاب 3 ویژگی به دست آمده

حالت LSTM همراه با MSE و RMSprop: حالت

در ابتدا می‌توانید که تعریف مدل به همراه Optimizer و نیز تابع Loss را مشاهده نمایید:

```
1 torch.manual_seed(13)
2 model = LSTM(n_features=3, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 462: تعریف مدل و Loss و تابع Optimizer در مدل LSTM با سه ویژگی

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 3.312053974562635e-05 , Valid Loss : => 4.529436879480878e-05
***->>>-----<<<-***  

step : 1 Train loss : 1.7189804592635482e-05 , Valid Loss : => 3.385713541259368e-05
***->>>-----<<<-***  

step : 2 Train loss : 1.4253318034267674e-05 , Valid Loss : => 2.614968013949692e-05
***->>>-----<<<-***  

step : 3 Train loss : 1.2197561562061309e-05 , Valid Loss : => 2.1367261341462533e-05
***->>>-----<<<-***  

step : 4 Train loss : 1.069613077561371e-05 , Valid Loss : => 1.8516625782164433e-05
***->>>-----<<<-***  

step : 5 Train loss : 9.621567707896853e-06 , Valid Loss : => 1.652843637081484e-05
***->>>-----<<<-***
```

شکل 463: ایپاک اول مدل LSTM

```

***->>-----<<-***  

step : 93 Train loss : 3.902943892656671e-06 , Valid Loss : => 5.156441077512379e-06  

***->>-----<<-***  

step : 94 Train loss : 3.898833130369895e-06 , Valid Loss : => 5.133705834547678e-06  

***->>-----<<-***  

step : 95 Train loss : 3.895346712670289e-06 , Valid Loss : => 5.068781436420977e-06  

***->>-----<<-***  

step : 96 Train loss : 3.910857618999823e-06 , Valid Loss : => 5.0382103654555975e-06  

***->>-----<<-***  

step : 97 Train loss : 3.902833626489155e-06 , Valid Loss : => 5.0791570538422095e-06  

***->>-----<<-***  

step : 98 Train loss : 3.8851131719032614e-06 , Valid Loss : => 5.041123882013684e-06  

***->>-----<<-***  

step : 99 Train loss : 3.885382599158523e-06 , Valid Loss : => 5.042325210524723e-06  

***->>-----<<-***
```

شکل 464: ایپاک آخر مدل LSTM

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****  

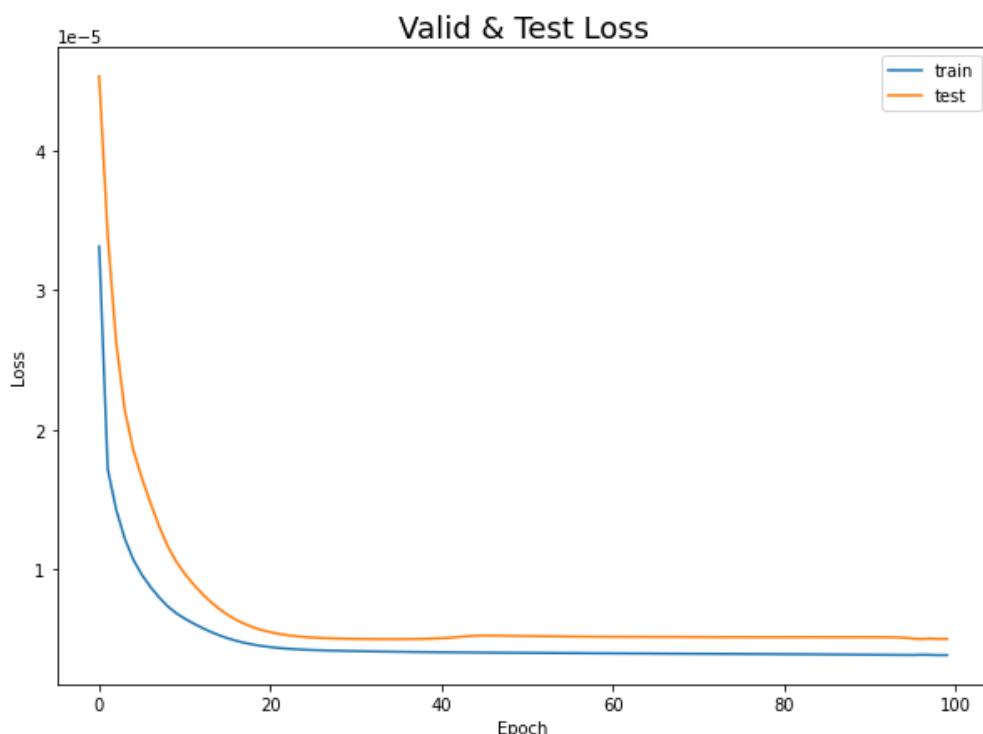
The total Training Time is Equal with ==> : 702.0366792678833 Sec.  

*****  

=====
```

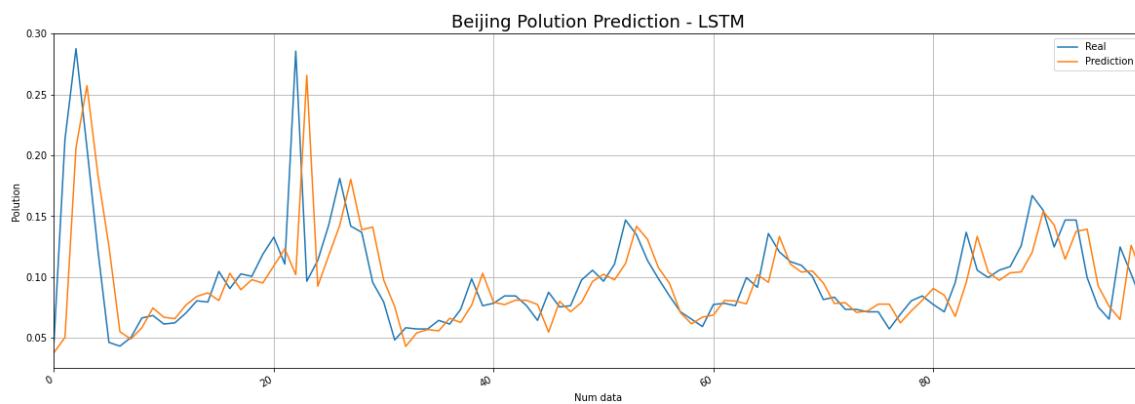
شکل 465: زمان اجرای آموزش در حالت مدل LSTM و با RMSprop و MSE و 3 ویژگی

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل 466: نمودار Loss در حالت مدل LSTM با RMSprop و MSE

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌های تست (100 عدد) مشاهده نمایید:



شکل 467: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت MSE و Adam

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.083063062959506e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 468: مقدار Loss به دست آمده برای داده‌های تست

حالت RNN همراه با MSE و RMSprop: حالت

در ابتدا می‌توانید که تعریف مدل به همراه Optimizer و نیز تابع Loss را مشاهده نمایید:

```
1 torch.manual_seed(13)
2 model = RNN(n_features=3, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 469: تعریف مدل و Optimizer و تابع Loss در مدل RNN با سه ویژگی

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 5.019151442684233e-05 , Valid Loss : => 3.065400756895542e-05
***->>>-----<<<-***  

step : 1 Train loss : 1.5912337348951646e-05 , Valid Loss : => 2.2624627454206348e-05
***->>>-----<<<-***  

step : 2 Train loss : 1.2669423507759347e-05 , Valid Loss : => 1.785806706175208e-05
***->>>-----<<<-***  

step : 3 Train loss : 1.1107911788470422e-05 , Valid Loss : => 1.543480084122469e-05
***->>>-----<<<-***  

step : 4 Train loss : 9.837962545376892e-06 , Valid Loss : => 1.3600790388106058e-05
***->>>-----<<<-***  

step : 5 Train loss : 8.945745540161927e-06 , Valid Loss : => 1.2146917385204384e-05
***->>>-----<<<-***
```

شکل ۱۵: ایپاک اول مدل RNN

```

***->>>-----<<<-***  

step : 93 Train loss : 4.151734813543347e-06 , Valid Loss : => 5.101292306790128e-06
***->>>-----<<<-***  

step : 94 Train loss : 4.155105233076029e-06 , Valid Loss : => 5.095096198298658e-06
***->>>-----<<<-***  

step : 95 Train loss : 4.171987859687457e-06 , Valid Loss : => 5.175628165792053e-06
***->>>-----<<<-***  

step : 96 Train loss : 4.135453594305242e-06 , Valid Loss : => 5.080710349526877e-06
***->>>-----<<<-***  

step : 97 Train loss : 4.149831394897774e-06 , Valid Loss : => 5.06982896088933e-06
***->>>-----<<<-***  

step : 98 Train loss : 4.152597136756716e-06 , Valid Loss : => 5.052162596257403e-06
***->>>-----<<<-***  

step : 99 Train loss : 4.158347968283731e-06 , Valid Loss : => 5.054888286394998e-06
***->>>-----<<<-***
```

شکل ۱۵: ایپاک آخر مدل RNN

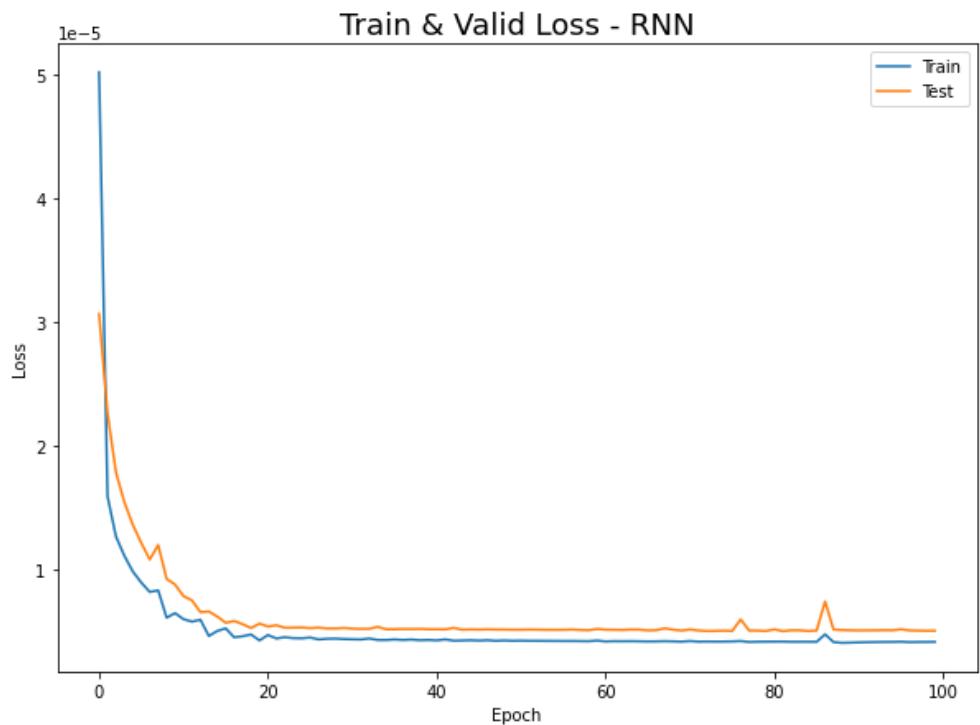
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 168.8154809474945 Sec.
*****
=====
```

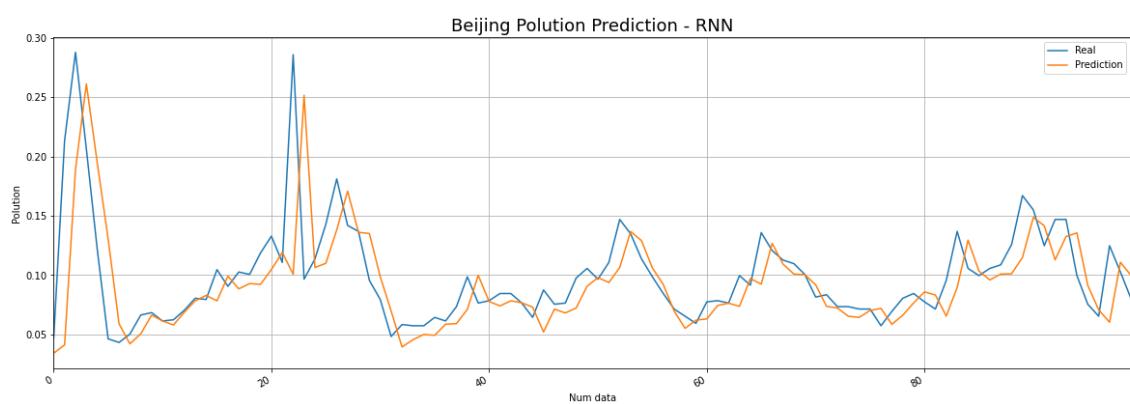
شکل ۱۶: زمان اجرای آموزش در حالت مدل RNN و با RMSprop و MSE و یزگی

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 473: نمودار Loss در حالت مدل RNN با MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌های تست (100 عدد) مشاهده نمایید:



شکل 474: نمودار مقدار واقعی و پیش‌بینی شده در حالت LSTM در مدل

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>>> 4.1481101143290285e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 475: مقدار Loss به دست آمده برای داده‌های تست

حالت GRU همراه با RMSprop و MSE

در ابتدا می‌توانید که تعریف مدل به همراه Optimizer و نیزتابع Loss را مشاهده نمایید:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=3, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 476: تعریف مدل و Optimizer و تابع Loss در مدل GRU با سه ویژگی

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```
step : 0 Train loss : 3.66969908393609e-05 , Valid Loss : => 3.501068560096125e-05
***->>>-----<<<-***  
step : 1 Train loss : 1.4917409936121354e-05 , Valid Loss : => 2.5604788951265315e-05
***->>>-----<<<-***  
step : 2 Train loss : 1.2194650624102602e-05 , Valid Loss : => 2.4453951589142283e-05
***->>>-----<<<-***  
step : 3 Train loss : 1.1119879382507254e-05 , Valid Loss : => 1.7850684196067353e-05
***->>>-----<<<-***  
step : 4 Train loss : 9.474685825019454e-06 , Valid Loss : => 1.499311455215017e-05
***->>>-----<<<-***  
step : 5 Train loss : 8.660967634447539e-06 , Valid Loss : => 1.3157675721837828e-05
***->>>-----<<<-***
```

شکل 477: ایپاک اول مدل GRU

```
***->>>-----<<<-***  
step : 93 Train loss : 3.9468592236517e-06 , Valid Loss : => 4.956366775635009e-06
***->>>-----<<<-***  
step : 94 Train loss : 3.944179078098386e-06 , Valid Loss : => 4.9552584435635555e-06
***->>>-----<<<-***  
step : 95 Train loss : 3.941502599627711e-06 , Valid Loss : => 4.95413871249184e-06
***->>>-----<<<-***  
step : 96 Train loss : 3.938842464898092e-06 , Valid Loss : => 4.953034086308132e-06
***->>>-----<<<-***  
step : 97 Train loss : 3.936201940329435e-06 , Valid Loss : => 4.951920292417829e-06
***->>>-----<<<-***  
step : 98 Train loss : 3.933578382323807e-06 , Valid Loss : => 4.950748387879381e-06
***->>>-----<<<-***  
step : 99 Train loss : 3.931013399657483e-06 , Valid Loss : => 4.949600979064902e-06
***->>>-----<<<-***
```

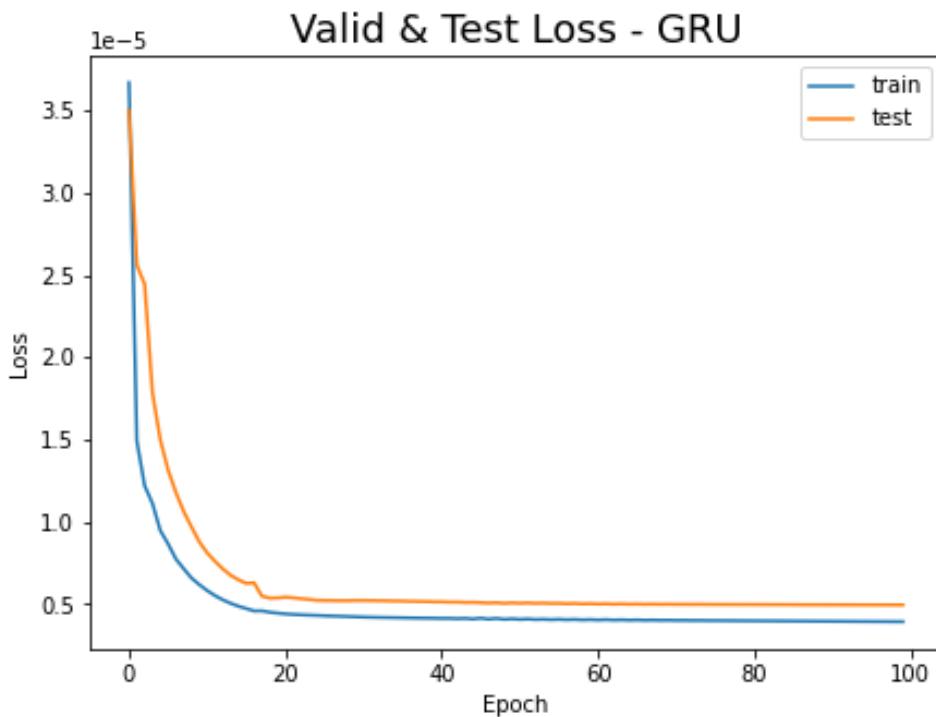
شکل 478: ایپاک آخر مدل GRU

همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```
=====
*****
The total Training Time is Equal with ==> : 577.2936375141144 Sec.
*****
=====
```

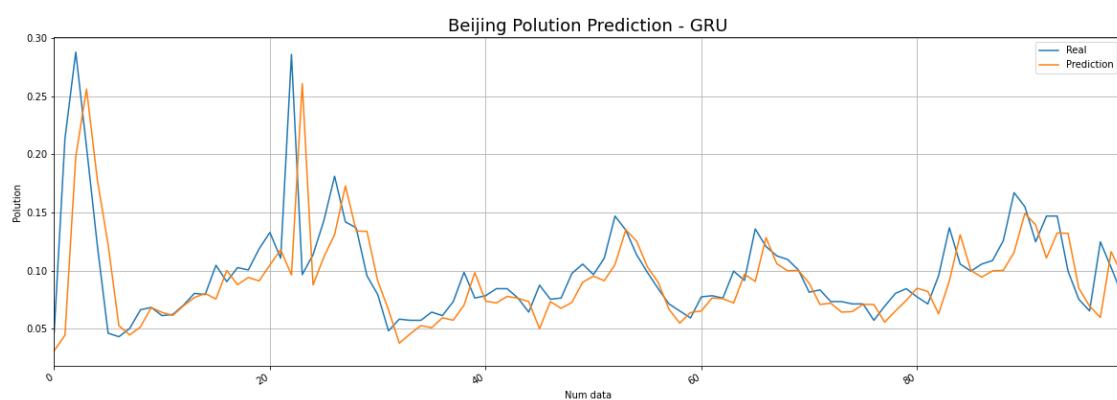
شکل 479: زمان اجرای آموزش در حالت مدل GRU و با RMSprop و MSE و 3 ویژگی

سپس در ادامه می‌توانید نمودار مربوط به این Loss‌ها را مشاهده نمایید:



شکل ۴۸۰: نمودار Loss در حالت مدل GRU با MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌های تست (100 عدد) مشاهده نمایید:



شکل ۴۸۱: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MSE و RMSprop

```

#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.261188812961336e-06
>>>-----*****-----<<<
>>>-----<<<
#####

```

شکل 482: مقدار Loss به دست آمده برای داده‌های تست

✓ پیاده‌سازی حالت ویژگی‌های به دست آمده از پیرسون (Pearson)

```

6 # split a multivariate sequence into samples
7 def split_sequences(sequences, n_steps=11, n_samples=12000, start_from=0):
8     X, y = list(), list()
9     for i in range(start_from, (start_from + n_samples)):
10         # find the end of this pattern
11         end_ix = i + n_steps
12         # check if we are beyond the dataset
13         # gather input and output parts of the pattern
14         seq_x = sequences[i:end_ix, [0, 1, 5]] # Change here
15         seq_y = sequences[end_ix, 0]
16         y.append(seq_y)
17         X.append(seq_x)
18
19     return array(X), array(y)

```

شکل 483: تغییر کد برای انتخاب 3 ویژگی به دست آمده در حالت Pearson

حالت LSTM همراه با MSE و RMSprop

در ابتدا می‌توانید که تعریف مدل به همراه Optimizer و نیز تابع Loss را مشاهده نمایید:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=3, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)

```

شکل 484: تعریف مدل و Optimizer و تابع Loss در مدل LSTM با سه ویژگی

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 3.0911578971426936e-05 , Valid Loss : => 4.265025677159429e-05
***->>-----<<<-***  

step : 1 Train loss : 1.667477124137804e-05 , Valid Loss : => 3.182447305880487e-05
***->>-----<<<-***  

step : 2 Train loss : 1.3903106209666779e-05 , Valid Loss : => 2.432524257649978e-05
***->>-----<<<-***  

step : 3 Train loss : 1.1939093402664487e-05 , Valid Loss : => 1.9611054643367728e-05
***->>-----<<<-***  

step : 4 Train loss : 1.0468045168090612e-05 , Valid Loss : => 1.6881820823376377e-05
***->>-----<<<-***  

step : 5 Train loss : 9.39964740925158e-06 , Valid Loss : => 1.5064209622020522e-05
***->>-----<<<-***
```

شکل 5: ایپاک اول مدل LSTM

```

***->>-----<<<-***  

step : 93 Train loss : 3.915358412389954e-06 , Valid Loss : => 5.2932692827501645e-06
***->>-----<<<-***  

step : 94 Train loss : 3.911600061595285e-06 , Valid Loss : => 5.294269901545097e-06
***->>-----<<<-***  

step : 95 Train loss : 3.907882267958484e-06 , Valid Loss : => 5.29455107365114e-06
***->>-----<<<-***  

step : 96 Train loss : 3.904204949503764e-06 , Valid Loss : => 5.294499191222712e-06
***->>-----<<<-***  

step : 97 Train loss : 3.900557207331682e-06 , Valid Loss : => 5.294370494084433e-06
***->>-----<<<-***  

step : 98 Train loss : 3.8969181613841405e-06 , Valid Loss : => 5.294200828454147e-06
***->>-----<<<-***  

step : 99 Train loss : 3.893268483807333e-06 , Valid Loss : => 5.293914582580328e-06
***->>-----<<<-***
```

شکل 5: ایپاک آخر مدل LSTM

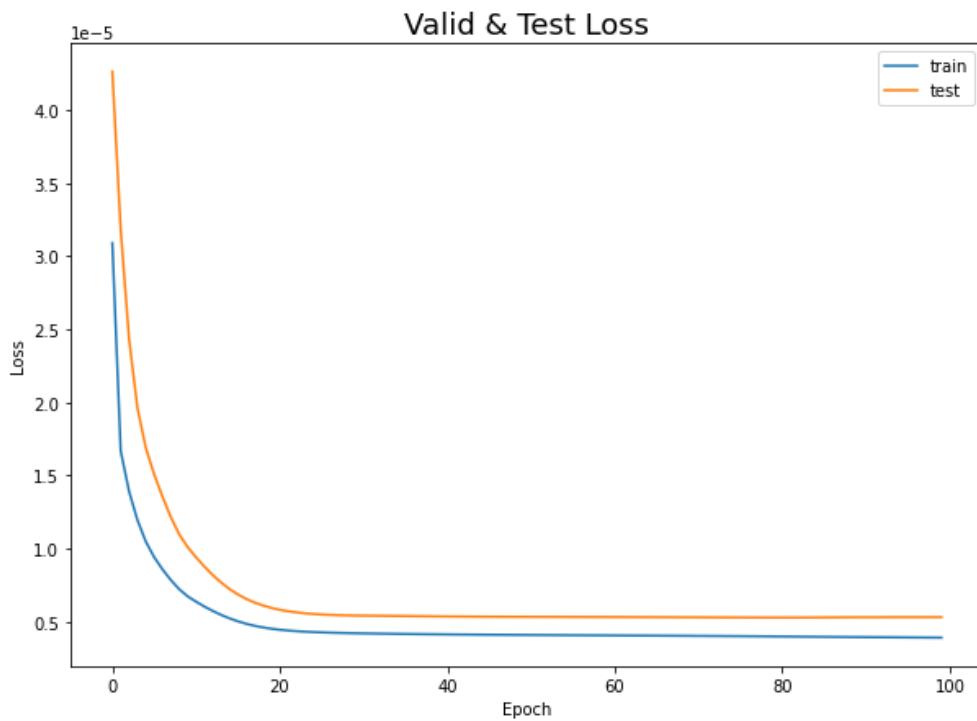
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 710.5631175041199 Sec.
*****
=====
```

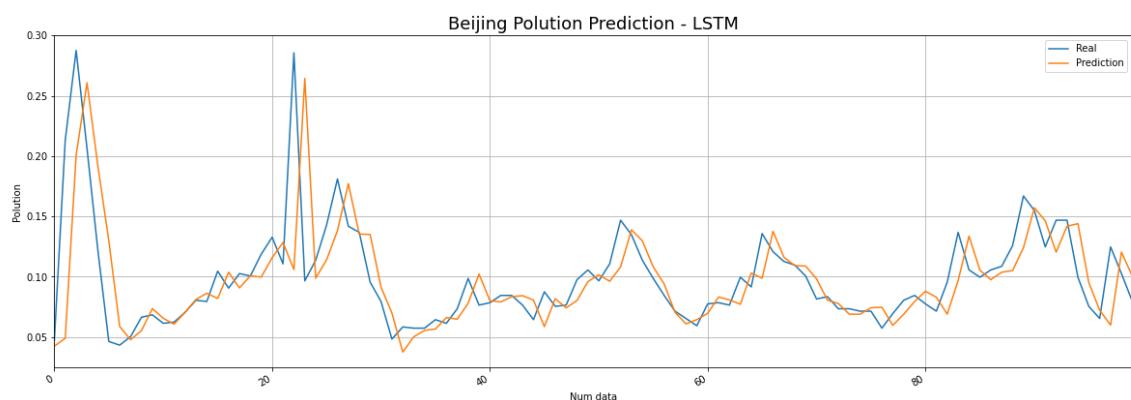
شکل 487: زمان اجرای آموزش در حالت مدل LSTM و با RMSprop و MSE و 3 ویژگی

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 488: نمودار Loss در حالت مدل LSTM با سه ویژگی MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌های تست (100 عدد) مشاهده نمایید:



شکل 489: نمودار مقدار واقعی و پیش‌بینی شده در مدل LSTM در حالت MSE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.094780160812661e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 490: مقدار Loss به دست آمده برای داده‌های تست

مقایسه حالت LSTM و RMSprop برای Pearson و MSE در حالت عادی

جدول: مقایسه حالت عادی پیرسون و گرادیان با سه ویژگی با حالت عادی 8 ویژگی

	حالت عادی	Pearson	Gradient
زمان آموزش	708.1412	710.56 Sec.	702.036
بر Loss روی داده‌های تست	4.0137430812661e-06	4.094780160812661e-06	4.083063062959506e-06

- از مقایسه جدول بالا به این نتیجه می‌رسیم که با استفاده از 3 ویژگی نسبت به حالت عادی کمی کاهش دقت داریم و مقدار Loss ما افزایش می‌یابد اما در حالت Gradient این موضوع کمتر است و اثر کمتری در دقت دارد و بنابراین این حالت بهتر است. در زمینه زمان اجرا تغییر محسوسی را نمیتوانیم مشاهده کنیم.

حالت RNN همراه با Pearson و RMSprop

در ابتدا می‌توانید که تعریف مدل به همراه Loss Optimizer و نیز تابع Pearson را مشاهده نمایید:

```

1 torch.manual_seed(13)
2 model = RNN(n_features=3, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 491: تعریف مدل و Optimizer و تابع Loss در مدل RNN با سه ویژگی

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 4.937637080826486e-05 , Valid Loss : => 3.0404661316424608e-05
***->>-----<<-***
step : 1 Train loss : 1.5826815704349427e-05 , Valid Loss : => 2.3403981739344695e-05
***->>-----<<-***
step : 2 Train loss : 1.2186225882032886e-05 , Valid Loss : => 1.7279648881716034e-05
***->>-----<<-***
step : 3 Train loss : 1.0901659707694004e-05 , Valid Loss : => 1.4922748475025098e-05
***->>-----<<-***
step : 4 Train loss : 9.65917957558607e-06 , Valid Loss : => 1.3340664736460894e-05
***->>-----<<-***
step : 5 Train loss : 8.852214705742274e-06 , Valid Loss : => 1.2225522620913884e-05
***->>-----<<-***

```

شکل 492: ایپاک اول مدل RNN

```

***->>-----<<-***
step : 93 Train loss : 4.198140840162523e-06 , Valid Loss : => 5.372765735955909e-06
***->>-----<<-***
step : 94 Train loss : 4.164677978648494e-06 , Valid Loss : => 5.336833836433167e-06
***->>-----<<-***
step : 95 Train loss : 4.156126463203691e-06 , Valid Loss : => 5.311291257385165e-06
***->>-----<<-***
step : 96 Train loss : 4.167410459679862e-06 , Valid Loss : => 5.325755715603009e-06
***->>-----<<-***
step : 97 Train loss : 4.164455454641332e-06 , Valid Loss : => 5.3163565365442385e-06
***->>-----<<-***
step : 98 Train loss : 4.187820203757534e-06 , Valid Loss : => 5.344654035676891e-06
***->>-----<<-***
step : 99 Train loss : 4.1566437042395895e-06 , Valid Loss : => 5.320888371594871e-06
***->>-----<<-***

```

شکل 493: ایپاک آخر مدل RNN

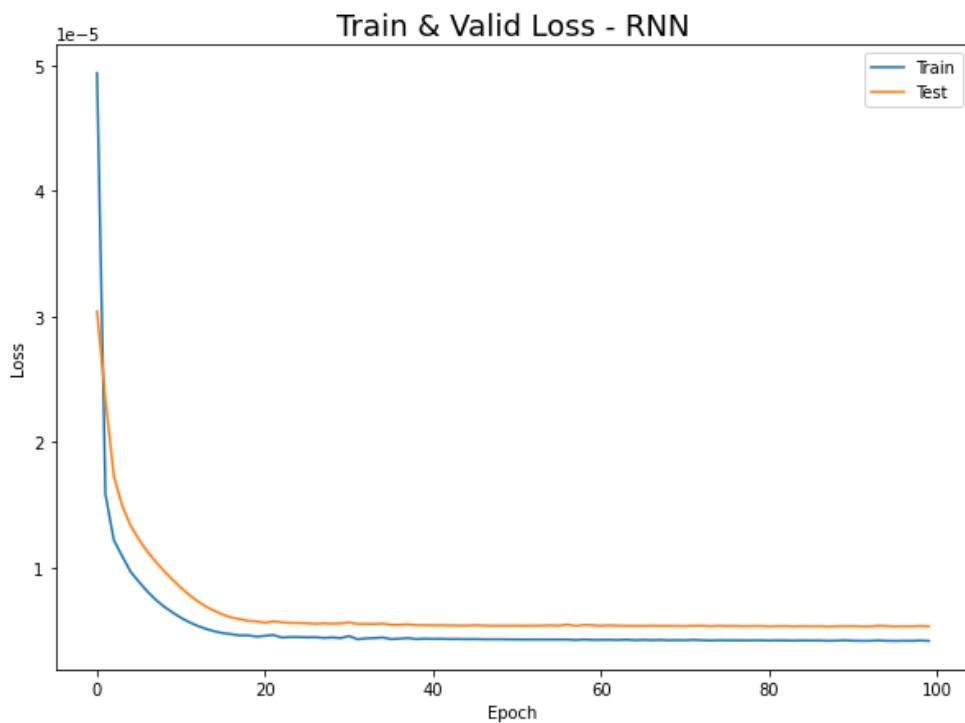
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 162.02086305618286 Sec.
*****
=====
```

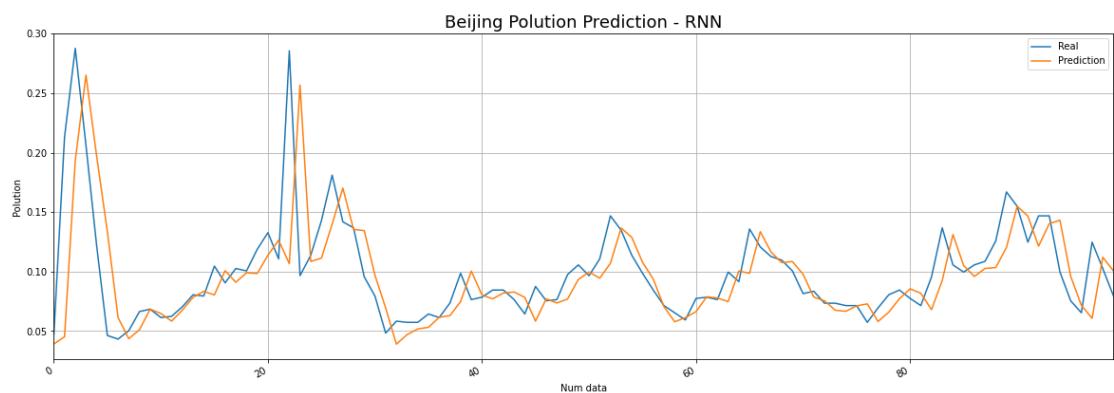
شکل 494: زمان اجرای آموزش در حالت مدل RNN و با RMSprop و 3 ویژگی MSE

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 495: نمودار Loss در حالت مدل RNN با سه ویژگی MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌های تست (100) مشاهده نمایید:



شکل 496: نمودار مقدار واقعی و پیش‌بینی شده در مدل RNN در حالت MSE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :==>> 4.288532429200131e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 497: مقدار Loss به دست آمده برای داده‌های تست

مقایسه حالت LSTM و RMSprop برای Pearson و MSE در حالت عادی

جدول 9: مقایسه حالت عادی پیرسون و گوادیان با سه ویژگی با حالت عادی 8 ویژگی

	حالت عادی	Pearson	Gradient
زمان آموزش	176.534 Sec.	162.020 Sec.	168.815 Sec.
بر روی داده‌ها	4.03028430812661e -06	4.288532429200131e -06	4.1481101143290285e -06

- از مقایسه جدول بالا به این نتیجه می‌رسیم که با استفاده از 3 ویژگی نسبت به حالت عادی کمی کاهش دقت داریم و مقدار Loss ما افزایش می‌یابد اما در حالت Gradient این موضوع کمتر است و اثر کمتری در دقت دارد و بنابراین این حالت بهتر است. در زمینه زمان اجرا تغییر محسوسی را نمیتوانیم مشاهده کنیم.

حالت GRU همراه با Pearson و RMSprop

در ابتداء می‌توانید که تعریف مدل به همراه Loss و نیز تابع Optimizer را مشاهده نمایید:

```
[8] 1 torch.manual_seed(13)
2 model = GRU(n_features=3, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.RMSprop(model.parameters(), lr=0.0003)
```

شکل 498: تعریف مدل و Loss در مدل GRU با سه ویژگی

سپس در ادامه به بررسی مقادیر Loss در این حالت می‌پردازیم:

```

step : 0 Train loss : 3.515452337063228e-05 , Valid Loss : => 3.5470758254329365e-05
***->>>-----<<<-***  

step : 1 Train loss : 1.4759601097709189e-05 , Valid Loss : => 2.5914453241663676e-05
***->>>-----<<<-***  

step : 2 Train loss : 1.1862809148927529e-05 , Valid Loss : => 2.1098574235414465e-05
***->>>-----<<<-***  

step : 3 Train loss : 1.059584496736837e-05 , Valid Loss : => 1.659059691398094e-05
***->>>-----<<<-***  

step : 4 Train loss : 9.263422580746313e-06 , Valid Loss : => 1.4009230304509402e-05
***->>>-----<<<-***  

step : 5 Train loss : 8.388775653050591e-06 , Valid Loss : => 1.2457932287361473e-05
***->>>-----<<<-***
```

شکل 499: 5 ایپاک اول مدل GRU

```

***->>>-----<<<-***  

step : 93 Train loss : 4.0066565580976505e-06 , Valid Loss : => 5.294012002802143e-06
***->>>-----<<<-***  

step : 94 Train loss : 4.005072483172019e-06 , Valid Loss : => 5.294289013060431e-06
***->>>-----<<<-***  

step : 95 Train loss : 4.0022527226634945e-06 , Valid Loss : => 5.294153301899011e-06
***->>>-----<<<-***  

step : 96 Train loss : 3.997364211439465e-06 , Valid Loss : => 5.284314276650548e-06
***->>>-----<<<-***  

step : 97 Train loss : 3.996971346593152e-06 , Valid Loss : => 5.2883342238298305e-06
***->>>-----<<<-***  

step : 98 Train loss : 3.9950237289303915e-06 , Valid Loss : => 5.289728559243183e-06
***->>>-----<<<-***  

step : 99 Train loss : 3.991070933019122e-06 , Valid Loss : => 5.2827679804371045e-06
***->>>-----<<<-***
```

شکل 500: 5 ایپاک آخر مدل GRU

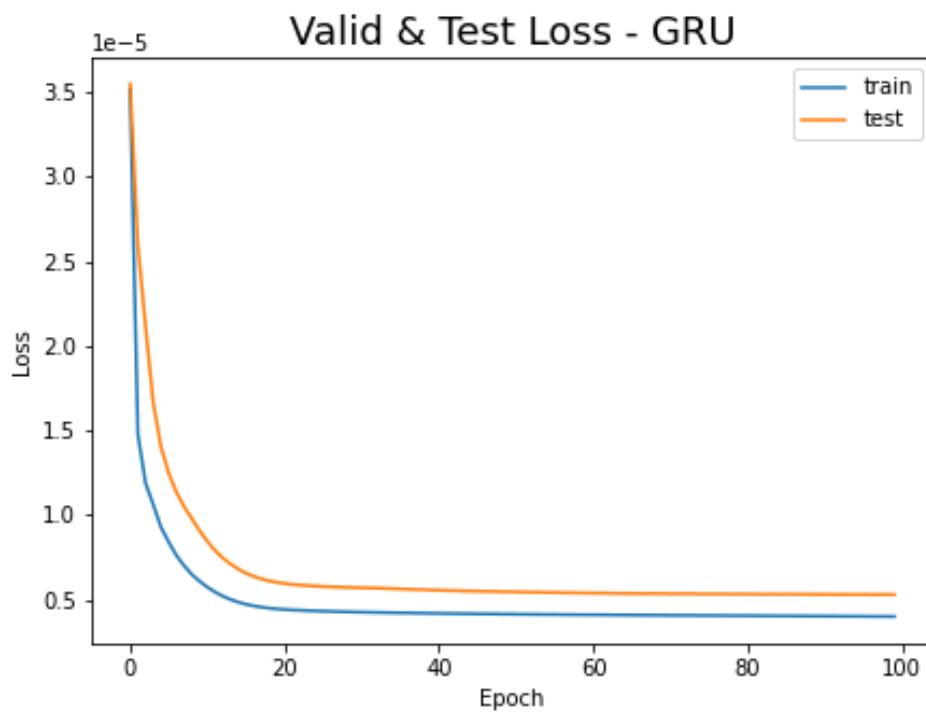
همچنین زمان اجرای این آموزش را نیز می‌توانید که در ادامه مشاهده نمایید:

```

=====
*****
The total Training Time is Equal with ==> : 546.1224629878998 Sec.
*****
=====
```

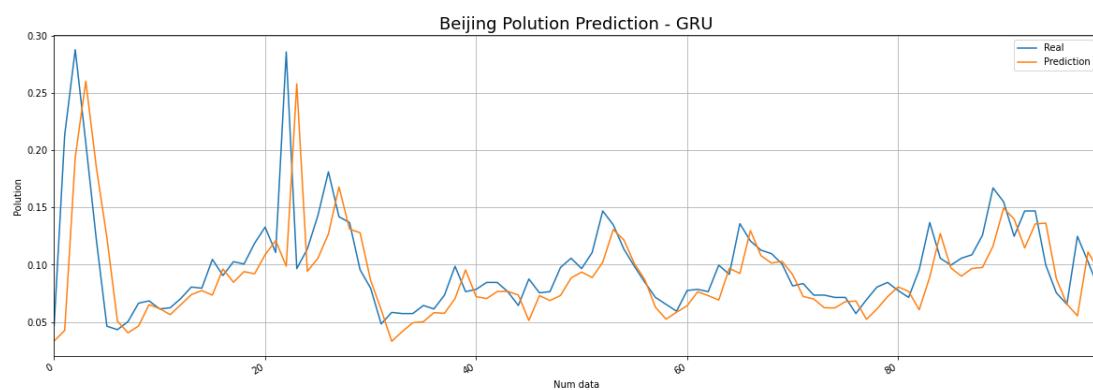
شکل 501: زمان اجرای آموزش در حالت مدل GRU و با RMSprop و 3 ویژگی MSE

سپس در ادامه می‌توانید نمودار مربوط به این Loss ها را مشاهده نمایید:



شکل 502: نمودار Loss در حالت مدل GRU با سه ویژگی MSE و RMSprop

سپس در ادامه می‌توانید نمودار مقدار واقعی و پیش‌بینی شده را برای تعدادی از داده‌های تست (100 عدد) مشاهده نمایید:



شکل 503: نمودار مقدار واقعی و پیش‌بینی شده در مدل GRU در حالت MSE و RMSprop

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>>> 4.385090154149414e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 504: مقدار Loss به دست آمده برای داده‌های تست

مقایسه حالت LSTM و RMSprop برای Pearson و MSE در حالت عادی

جدول 10: مقایسه حالت عادی پیرسون و گرادیان با سه ویژگی با حالت عادی 8 ویژگی

حالت عادی		Pearson	Gradient
زمان آموزش	570.3360 Sec.	546.122 Sec.	577.2936 Sec.
بر Loss روی داده‌های تست	4.2099830812661e-06	4.385090154149414e-06	4.261188812961336e-06

- از مقایسه جدول بالا به این نتیجه می‌رسیم که با استفاده از 3 ویژگی نسبت به حالت عادی کمی کاهش دقت داریم و مقدار Loss ما افزایش می‌یابد اما در حالت Gradient این موضوع کمتر است و اثر کمتری در دقت دارد و بنابراین این حالت بهتر است. در زمینه زمان اجرا تغییر محسوسی را نمیتوانیم مشاهده کنیم.

+ نتیجه نهایی: در نهایت با بررسی تمامی حالات بالا به این نتیجه می‌رسیم که استفاده از 3 ویژگی در کل کمی کاهش در دقت مدل ما دارد و مقدار کمی زمان اجرای ما را کاهش می‌دهد در مواردی.

+ اما نکته مهم این است که در دو حالتی که من امتحان کردم یعنی Gradient و Pearson حالت گرادیان کاهش دقت کمتری داشت و شبکه بهتر عملکردش را حفظ می‌کرد.

تمامی کدهای این قسمت در پوشه Q1 PartG&H – Codes در فایل آپلود شده قرار دارد.

سوال ۲ – نقصان دادگان

در این سوال از ما خواسته شده است که تاثیر Missing Data را بر روی شبکه بررسی کنیم.

داده‌ها در دنیای واقعی به ندرت پیش می‌آید که همگن باشند و اصطلاحاً تمیز باشند یعنی کم و کاست و داده‌های پرت نداشته باشند. داده‌ها ممکن است که در طول یا جمع آوری و یا در طول بهره‌برداری از آن‌ها برایشان مشکل پیش بیاید. داده‌های از دست رفته باید handle بشوند به دلیل این که وجود آن‌ها به شدت روی Performance ما تاثیر دارد. همچنین ممکن است که وجود این داده‌ها باعث شود که در مسئله کلاس‌بندی یا clustering کلان‌نتیجه اشتباهی را به دست بیاوریم.

در این سوال ما قصد داریم که به بررسی دقیقاً همین مشکل بپردازیم.

❖ قسمت A و B :

در این قسمت از ما خواسته شده است که به صورت تصادفی و Random 20 درصد از داده‌های موجود در هر ستون را حذف کنیم و یا به عبارت دیگر مقدار آن‌ها را به Nan تغییر دهیم که به این معنی است که مقدار ندارد.

برای انجام این کار باید ابتدا بنویسیم که 20 درصد از داده‌های یک ستون را که به آن می‌دهیم را به صورت تصادفی انتخاب کند که برای انجام این کار به صورت زیر عمل می‌کنیم:

```
7 def select_random_rows(seq_start=0, seq_end=12000, rand_percent=20):
8     rand_num = int((20*(seq_end-seq_start))/100)
9     rand_gen = random.sample(range(seq_start, seq_end), rand_num)
10    return rand_gen
```

شکل 505: تابع انتخاب داده‌های مورد نظر برای حذف یا Nan کردن

در ادامه یک تابع می‌نویسیم که باید و این 20 درصد را برای هر کدام از ستون‌ها تولید کند و سپس مقدار این داده‌های انتخاب شده را حذف کند و به Nan تبدیل کند:

```

12 def apply_nan(sequence, seq_start=0, seq_end=12000, rand_percent=20):
13     for i in range(0, 8):
14         rand_r = select_random_rows(seq_start=seq_start, seq_end=seq_end, rand_percent=rand_percent)
15         np.put(sequence[seq_start:seq_end, i], rand_r, np.nan)
16     return sequence

```

شکل 506: تابع برای حذف کردن 20 درصد از داده‌های هر کدام از 8 ستون دیتابست

در ادامه یک تست هم انجام می‌دهم تا از کارکرد درست این توابع مطمئن شوم و به این منظور تعداد حالت‌هایی که مقدار آن‌ها در یک ستون Nan است را می‌شمارم:

```

49 n = apply_nan(data, seq_start=0, seq_end=12000, rand_percent=20)
50 occurrences = np.count_nonzero(np.isnan(data[:, 1]))
51 occurrences

```

2400

شکل 507: تست عملکرد برنامه نوشته شده

همان‌طور که در شکل بالا می‌توانید مشاهده نمایید در یک ستون توانسته است به همان تعداد مورد نظر Nan ایجاد کند و درست کار می‌کند.

:C قسمت ❖



شکل 508: برطرف کردن مشکل Missing Data

در این قسمت از ما خواسته شده است که 3 روش را برای برطرف کردن مشکل این نقصان داده پیدا کنیم.

1. حذف یا Drop کردن Missing Value ها:

این روش سریع‌ترین و نیز راحت‌ترین روش و راه برای برطرف کردن Missing Value ها هست. هرچند که استفاده از این روش خیلی توصیه نمی‌شود. نحوه عملکرد این روش به این صورت است که آن Row و سطrix که Missing Data درون آن قرار دارد را به کلی حذف می‌کنیم.

این روش به دلیل این که سایز دیتاست ما را کاهش می‌دهد که علت آن هم همان حذف کردن دیتاهای شده هست، کیفیت و دقت مدل ما را کاهش می‌دهد.

2. پر کردن و پوشش دادن Missing Value ها:

این روش معمول‌ترین روش برای رسیدگی و بررسی Missing Value ها است. در این روش در هر کجا که Missing Value وجود داشته باشد، مقدار آن با یک معیار آماری مانند میانگین، میانه یا مد داده‌های آن ویژگی، جایگزین می‌شود. برای انجام این کار می‌توانیم که از backward-fill یا forward-fill استفاده کنیم و به این صورت که بر روی مقادیر بعدی propagate کنیم و مقدار مقادیر از دست رفته را به دست بیاوریم.

یکی از روش‌های خوب انجام این کار نیز استفاده از pipeline هست که مقادیری که بیشتر از همه در یک ویژگی frequent هستند را به جای مقادیری که از بین رفته‌اند قرار می‌دهد.

3. یک مدل پیش‌بینی کننده با قابلیت رسیدگی به Missing Data

این روش با فاصله خیلی زیاد یکی از بهترین و کارآمدترین روش‌ها برای مدیریت این داده‌های از دست رفته هست. بسته به نوع داده‌ها که Miss شده‌اند می‌توانیم از مدل classification یا Regression برای پیش‌بینی داده از دست رفته استفاده کنیم. این روش با تبدیل ویژگی‌های Miss شده به labelها و سپس استفاده از ستون‌های بدون Missing Value برای پیش‌بینی ستون‌های دارای Missing Value انجام می‌شود.

❖ قسمت D ❖

در این قسمت از ما خواسته شده است که با یک روش دلخواه داده‌هایی که در مرحله قبل از دستدادیم را پیش‌بینی کنیم.

من ابتدا از یک روش آسان می‌روم که در آن مقادیر از دست رفته را با میانگین داده‌های موجود جایگزین می‌کند. و این کار را به صورت زیر انجام می‌دهم:

```
12 import numpy.ma as ma  
13 data = np.where(np.isnan(data), ma.array(data, mask=np.isnan(data)).mean(axis=0), data)
```

شکل 509: جایگزین کردن مقادیر از دست رفته با میانگین آن ستون

سپس برای این که مطمئن شویم دیگر مقدار Nan در داده‌های ما وجود ندارد، به صورت زیر عمل می‌کنم:

```
16 occurrences = np.count_nonzero(np.isnan(data1[:, :]))  
17 print("Number of Nan in Dataset : => ", occurrences)
```

```
Number of Nan in Dataset : => 0
```

شکل 510: محاسبه تعداد مقادیر Nan

همان‌طور که در شکل بالا می‌توانید مشاهده کنید، دیگر مقدار Nan ای در دیتابست ما وجود ندارد.

❖ قسمت E ❖

در این قسمت از ما خواسته شده است که با استفاده از MSE Loss بیاییم و مقدار خطای به دست آمده در هر ستون را نسبت به حالت دیتابست عادی گزارش دهیم.

برای انجام این کار از دستورات زیر استفاده می‌کنم:

```
1. criterion = nn.MSELoss()  
2. loss1 = criterion(torch.tensor(data1[:, 0], dtype=torch.float32), torch.tensor(da  
ta[:, 0], dtype=torch.float32))  
3. loss2 = criterion(torch.tensor(data1[:, 1], dtype=torch.float32), torch.tensor(da  
ta[:, 1], dtype=torch.float32))  
4. loss3 = criterion(torch.tensor(data1[:, 2], dtype=torch.float32), torch.tensor(da  
ta[:, 2], dtype=torch.float32))  
5. loss4 = criterion(torch.tensor(data1[:, 3], dtype=torch.float32), torch.tensor(da  
ta[:, 3], dtype=torch.float32))
```

```

6. loss5 = criterion(torch.tensor(data1[:, 4], dtype=torch.float32), torch.tensor(da
ta[:, 4], dtype=torch.float32))
7. loss6 = criterion(torch.tensor(data1[:, 5], dtype=torch.float32), torch.tensor(da
ta[:, 5], dtype=torch.float32))
8. loss7 = criterion(torch.tensor(data1[:, 6], dtype=torch.float32), torch.tensor(da
ta[:, 6], dtype=torch.float32))
9. loss8 = criterion(torch.tensor(data1[:, 7], dtype=torch.float32), torch.tensor(da
ta[:, 7], dtype=torch.float32))
10. print("MSE Loss Column 1 :=> {:.5f}".format(loss1.item()))
11. print("MSE Loss Column 2 :=> {:.5f}".format(loss2.item()))
12. print("MSE Loss Column 3 :=> {:.5f}".format(loss3.item()))
13. print("MSE Loss Column 4 :=> {:.5f}".format(loss4.item()))
14. print("MSE Loss Column 5 :=> {:.5f}".format(loss5.item()))
15. print("MSE Loss Column 6 :=> {:.5f}".format(loss6.item()))
16. print("MSE Loss Column 7 :=> {:.5f}".format(loss7.item()))
17. print("MSE Loss Column 8 :=> {:.5f}".format(loss8.item()))

```

سپس در ادامه پس از اجرای این دستورات خروجی به صورت زیر را به دست می‌آورم:

```

MSE Loss Column 1 :=> 0.00050
MSE Loss Column 2 :=> 0.00338
MSE Loss Column 3 :=> 0.00291
MSE Loss Column 4 :=> 0.00253
MSE Loss Column 5 :=> 0.00639
MSE Loss Column 6 :=> 0.00073
MSE Loss Column 7 :=> 0.00005
MSE Loss Column 8 :=> 0.00014

```

شکل 511: مقدار Loss های به دست آمده برای داده‌های هر ستون از دیتابیس پس از جایگزین کردن مقدارهای میانگین با NaNها

❖ قسمت F ❖

در این قسمت از ما خواسته شده است که برای دو شبکه LSTM و GRU بیاییم و این داده‌های دارای مقادیر از دست رفته که ما با استفاده از میانگین آن‌ها را جایگزین کرده‌ایم را بررسی کنیم و عملکرد آن‌ها را بسنجیم.

به این منظور من دیتابیسی دارای مقادیر Miss که آن‌ها را من پر کرده‌ام را به شبکه می‌دهم تا آموزش ببیند و مقادیر هدف را نیز مقادیر هدف واقعی را می‌دهم:

► بررسی شبکه LSTM در حالت MSE و Adam

ابتدا مدل طراحی شده به همراه Loss را در ادامه قرار می‌دهم:

```

1 torch.manual_seed(13)
2 model = LSTM(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)

```

شکل 512: طراحی مدل مورد استفاده در LSTM

سپس مقادیر Loss را در اینجا قرار می‌دهم:

```

step : 0 Train loss : 3.63441220484674e-05 , Valid Loss : => 6.634043746938308e-05
***->>>-----<<<-***  

step : 1 Train loss : 2.800491606661429e-05 , Valid Loss : => 0.0001011171592399478
***->>>-----<<<-***  

step : 2 Train loss : 3.391080225507418e-05 , Valid Loss : => 9.24733771632115e-05
***->>>-----<<<-***  

step : 3 Train loss : 3.4154599571290117e-05 , Valid Loss : => 7.093132364874085e-05
***->>>-----<<<-***  

step : 4 Train loss : 2.4434200678175937e-05 , Valid Loss : => 4.5199648477137086e-05
***->>>-----<<<-***  

step : 5 Train loss : 1.8717230080316463e-05 , Valid Loss : => 4.6953033190220594e-05
***->>>-----<<<-***  


```

شکل 513: مقادیر 5 لاس اول آموزش مدل

```

***->>>-----<<<-***  

step : 93 Train loss : 7.338094158330933e-06 , Valid Loss : => 1.1955835390836e-05
***->>>-----<<<-***  

step : 94 Train loss : 7.31902732901896e-06 , Valid Loss : => 1.1977895473440488e-05
***->>>-----<<<-***  

step : 95 Train loss : 7.308694487437606e-06 , Valid Loss : => 1.2052602988357346e-05
***->>>-----<<<-***  

step : 96 Train loss : 7.304275006754324e-06 , Valid Loss : => 1.2144561080882946e-05
***->>>-----<<<-***  

step : 97 Train loss : 7.302552583860234e-06 , Valid Loss : => 1.2228443287312985e-05
***->>>-----<<<-***  

step : 98 Train loss : 7.300933330164602e-06 , Valid Loss : => 1.2292213078277806e-05
***->>>-----<<<-***  

step : 99 Train loss : 7.298008164313311e-06 , Valid Loss : => 1.2336374648536244e-05
***->>>-----<<<-***  


```

شکل 514: مقادیر 5 لاس آخر آموزش مدل

در ادامه نیز می‌توانید زمان صرف شده برای آموزش این مدل را مشاهده نمایید:

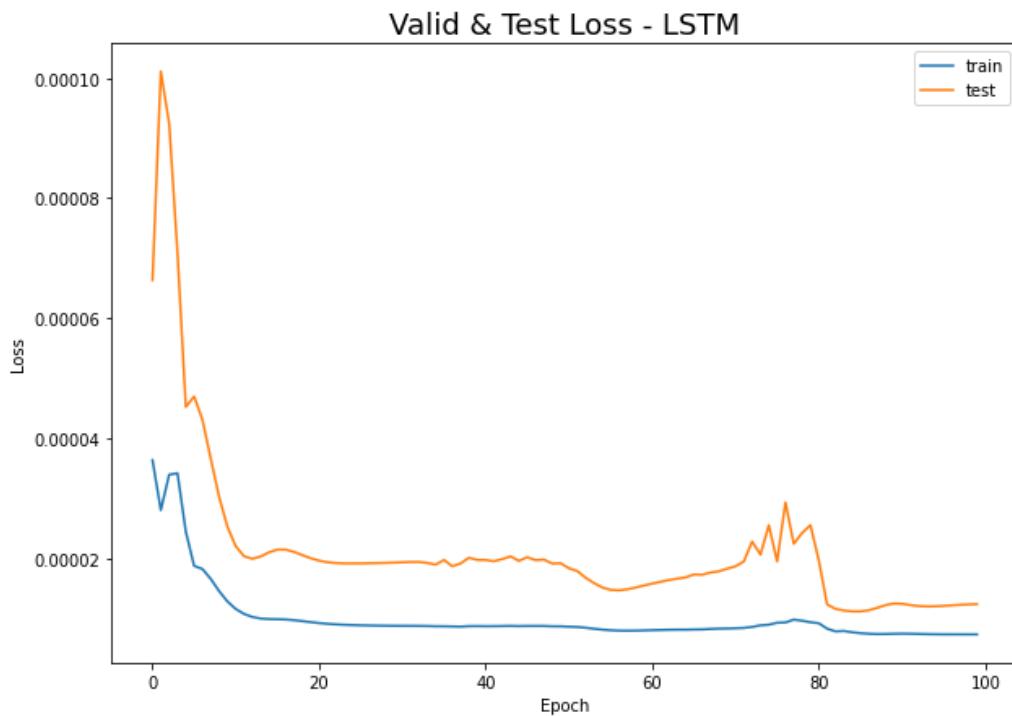
```

=====
*****
The total Training Time is Equal with ==> : 764.349057674408 Sec.
*****
=====
```

شکل 515: زمان آموزش این مدل

مشاهده می‌کنیم که زمان آموزش در این حالت کمی افزایش پیدا کرده است اما این افزایش چشمگیر نیست شاید هم به دلایل دیگری رخ داده باشد.

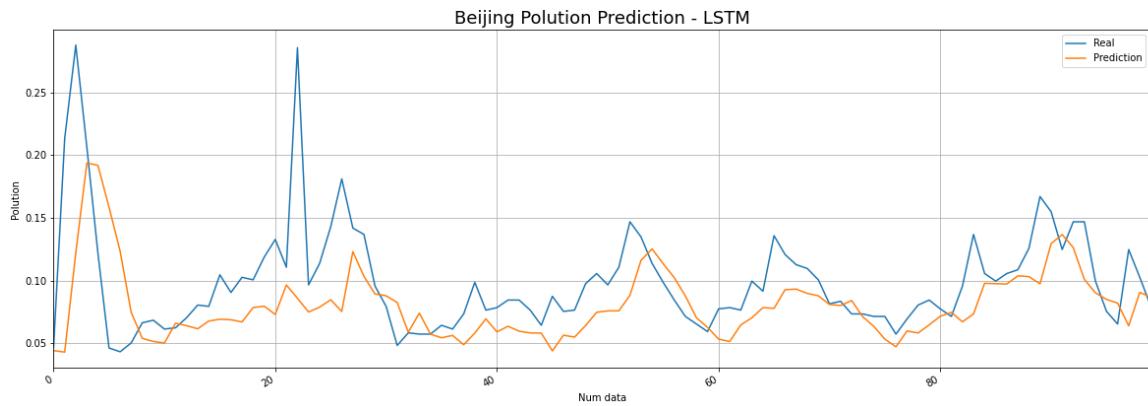
سپس در ادامه نیز می‌توانید نمودار Loss را مشاهده کنید:



شکل 516: نمودار Loss به دست آمده برای Train و Valid در آموزش این مدل

نمودار Loss به دست آمده در این حالت نیز کاهش دقیق را نشان می‌دهد در حالی که نمودار دقیقی که پیش‌تر در حالت بدون داشتن داده‌های Miss داشتیم نوسانات کمتری داشت و نیز با نرخ بهتری کاهش پیدا می‌کرد و منظم‌تر بود اما در اینجا اینطور نیست.

سپس در ادامه نیز می‌توانید نمودار مقایسه میان حالت واقعی و پیش‌بینی را به ازای حدود 100 داده را مشاهده نمایید:



شکل 517: نمودار حالت واقعی و پیش‌بینی شده به دست آمده برای این مدل

در این حالت نیز با مقایسه نمودار مقادیر پیش‌بینی شده و مقادیر واقعی با حالت به دست آمده در سوال 1 قسمت A و B به این نتیجه می‌رسیم که این شبکه در این حالت دقیق و کارایی‌اش کاهش چشمگیری پیدا کرده است و این دستکاری در داده‌ها اثر زیادی در دقت شبکه ما داشته است.

در ادامه نیز می‌توانید مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>>> 7.73549423320219e-06
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 518: مقدار Loss به دست آمده برای داده‌های تست در این مدل

- **نتیجه:** با مقایسه مقدار Loss به دست آمده در این حالت با مقدار Loss به دست آمده در سوال 1 قسمت A و B به این نتیجه می‌رسیم که مقدار Loss تقریباً دو برابر شده است و این به این معنی است که دقت در این حالت که ما داده‌های دارای مقادیر از دست رفته را به مدلمان می‌دهیم در مقایسه با حالت عادی کاهش نسبتاً زیادی پیدا کرده است.

► بررسی شبکه GRU در حالت MSE و Adam

ابتدا مدل طراحی شده به همراه Loss را در ادامه قرار می‌دهم:

```
1 torch.manual_seed(13)
2 model = GRU(n_features=8, n_output=1, seq_length=11, n_hidden_layers=233, n_layers=1)
3 criterion = nn.MSELoss()
4 optimizer = torch.optim.Adam(model.parameters(), lr=0.0003)
```

شکل 519: طراحی مدل مورد استفاده در LSTM

سپس مقادیر Loss را در اینجا قرار می‌دهم:

```
step : 0 Train loss : 3.324034200826039e-05 , Valid Loss : => 7.925129293774565e-05
***->>>-----<<<-***  
step : 1 Train loss : 2.3834512270210933e-05 , Valid Loss : => 8.588926711430152e-05
***->>>-----<<<-***  
step : 2 Train loss : 2.440852333093062e-05 , Valid Loss : => 8.926124311983585e-05
***->>>-----<<<-***  
step : 3 Train loss : 2.481215700584774e-05 , Valid Loss : => 7.135632665206988e-05
***->>>-----<<<-***  
step : 4 Train loss : 2.2777253168169407e-05 , Valid Loss : => 8.587889885529875e-05
***->>>-----<<<-***  
step : 5 Train loss : 2.4918310379143805e-05 , Valid Loss : => 7.356523256748914e-05
***->>>-----<<<-***
```

شکل 520: مقادیر 5 لاس اول آموزش مدل

```
***->>>-----<<<-***  
step : 93 Train loss : 8.475713311539343e-06 , Valid Loss : => 2.7161414579798778e-05
***->>>-----<<<-***  
step : 94 Train loss : 8.58205083059147e-06 , Valid Loss : => 2.4116553521404663e-05
***->>>-----<<<-***  
step : 95 Train loss : 8.60022590495646e-06 , Valid Loss : => 2.1949905203655362e-05
***->>>-----<<<-***  
step : 96 Train loss : 8.594769119129827e-06 , Valid Loss : => 2.0418477904361982e-05
***->>>-----<<<-***  
step : 97 Train loss : 8.344979774362098e-06 , Valid Loss : => 1.7901985828454297e-05
***->>>-----<<<-***  
step : 98 Train loss : 8.12098642733569e-06 , Valid Loss : => 2.0275041189355154e-05
***->>>-----<<<-***  
step : 99 Train loss : 8.187502964089314e-06 , Valid Loss : => 2.1726982396406433e-05
***->>>-----<<<-***
```

شکل 521: مقادیر 5 لاس آخر آموزش مدل

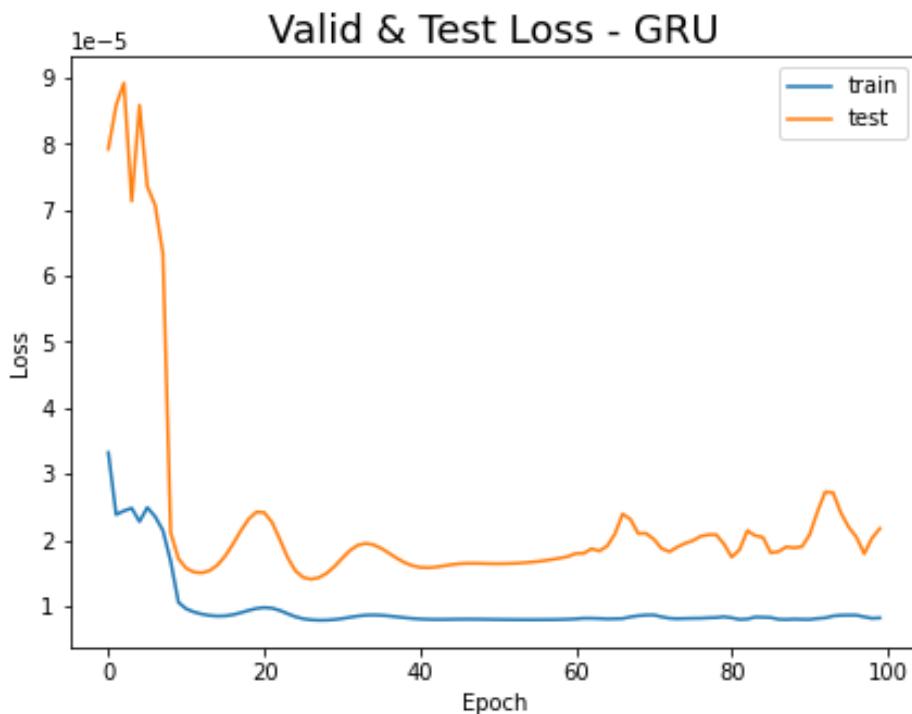
در ادامه نیز می‌توانید زمان صرف شده برای آموزش این مدل را مشاهده نمایید:

```
=====
*****  
The total Training Time is Equal with ==> : 547.3195297718048 Sec.  
*****  
=====
```

شکل 522: زمان آموزش این مدل

مشاهده می کنیم که زمان آموزش در این حالت کمی افزایش پیدا کرده است اما این افزایش چشمگیر نیست شاید هم به دلایل دیگری رخ داده باشد.

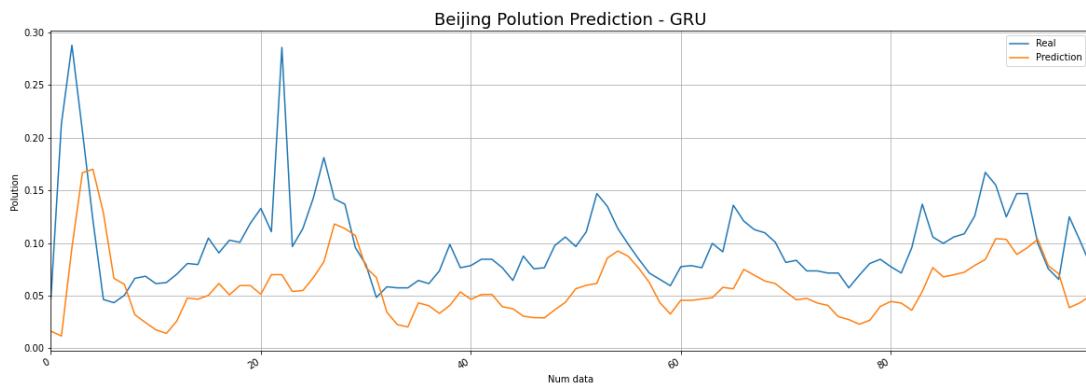
سپس در ادامه نیز می توانید نمودار Loss را مشاهده کنید:



شکل 523: نمودار Loss به دست آمده برای Train و Valid در آموزش این مدل

نمودار Loss به دست آمده در این حالت نیز کاهش دقیق را نشان می دهد در حالی که نمودار دقیقی که پیشتر در حالت بدون داشتن داده های Miss داشتیم نوسانات کمتری داشت و نیز با نرخ بهتری کاهش پیدا می کرد و منظم تر بود اما در اینجا اینطور نیست.

سپس در ادامه نیز می توانید نمودار مقایسه میان حالت واقعی و پیش بینی را به ازای حدود 100 داده را مشاهده نمایید:



شکل 524: نمودار حالت واقعی و پیش‌بینی شده به دست آمده برای این مدل

در این حالت نیز با مقایسه نمودار مقادیر پیش‌بینی شده و مقادیر واقعی با حالت به دست آمده در سوال 1 قسمت A و B به این نتیجه می‌رسیم که این شبکه در این حالت دقیق و کارایی‌اش کاهش چشمگیری پیدا کرده است و این دستکاری در داده‌ها اثر زیادی در دقت شبکه ما داشته است.

در ادامه نیز می‌توانید مقدار Loss به دست آمده برای داده‌های تست را مشاهده نمایید:

```
#####
>>>-----<<<
>>>-----*****-----<<<
**** Test Loss :=>> 1.5681098331697285e-05
>>>-----*****-----<<<
>>>-----<<<
#####
```

شکل 525: مقدار Loss به دست آمده برای داده‌های تست در این مدل

- **نتیجه:** با مقایسه مقدار Loss به دست آمده در این حالت با مقدار Loss به دست آمده در سوال 1 قسمت A و B به این نتیجه می‌رسیم که مقدار Loss تقریباً دو برابر شده است و این به این معنی است که دقت در این حالت که ما داده‌های دارای مقادیر از دست رفته را به مدلمان می‌دهیم در مقایسه با حالت عادی کاهش نسبتاً زیادی پیدا کرده است.

• باتشکر از زحمات شما

پرهام زیلوچیان مقدم