



به نام خدا



دانشگاه تهران

دانشکده مهندسی برق و کامپیووتر

شبکه های عصبی و یادگیری عمیق

مینی پروژه دوم

| | | |
|-------------|----------------|--------------------|
| میلاد دولتی | علیرضا فدایکار | نام و نام خانوادگی |
| 810195555 | | شماره دانشجویی |
| 1399/3/20 | | تاریخ ارسال گزارش |

فهرست گزارش سوالات

| | |
|---------|---|
| 3..... | قسمت اول) طراحی شبکه های عصبی |
| 7..... | بخش 1) نمودارهای مقادیر حقیقی و پیش بینی train و test |
| 8..... | بخش 2) مقایسه دقت و سرعت لایه های RNN, LSTM, GRU |
| 16..... | بخش 3) مقایسه دقت و سرعت با توابع بهینه سازی مختلف |
| 30..... | بخش 4) پیش بینی هفتگی و ماهانه |
| 37..... | بخش 5) تأثیر لایه Dropout |
| 43..... | بخش 6) لایه fusion و موازی کردن سه شبکه به کمک لایه Average |
| 50..... | بخش 7 و 8) پیش بینی با استفاده از سه ستون |
| 59..... | قسمت دوم) نقصان دادگان (بخش 1) |
| 60..... | بخش 2, 3, 4 و 5) نقصان دادگان (بخش 1) |
| 65..... | بخش 6) پیش بینی آلودگی پس از برطرف کردن missing values |

قسمت اول) طراحی شبکه های عصبی

در ابتدا مطابق شکل 1-0 کتابخانه های لازم را import می کنیم:

```
1 import numpy as np
2 import pandas as pd
3 import tensorflow as tf
4 import matplotlib.pyplot as plt
5 from tensorflow.keras.models import Sequential, Model
6 from tensorflow.keras.layers import Dense, Dropout, LSTM, GRU, SimpleRNN, Average, Input
7 from tensorflow.keras.callbacks import History
8 from sklearn.preprocessing import MinMaxScaler
9 from sklearn.metrics import mean_squared_error
10 import time
```

شکل 1-0: کتابخانه های لازم

سپس دیتاست را به کمک دستور load از کتابخانه numpy ، لود می کنیم که در شکل 2-0 نشان داده شده است. همانطور که مشاهده می شود به تعداد 43799 داده به همراه 8 ویژگی (یا فیچر) داریم.

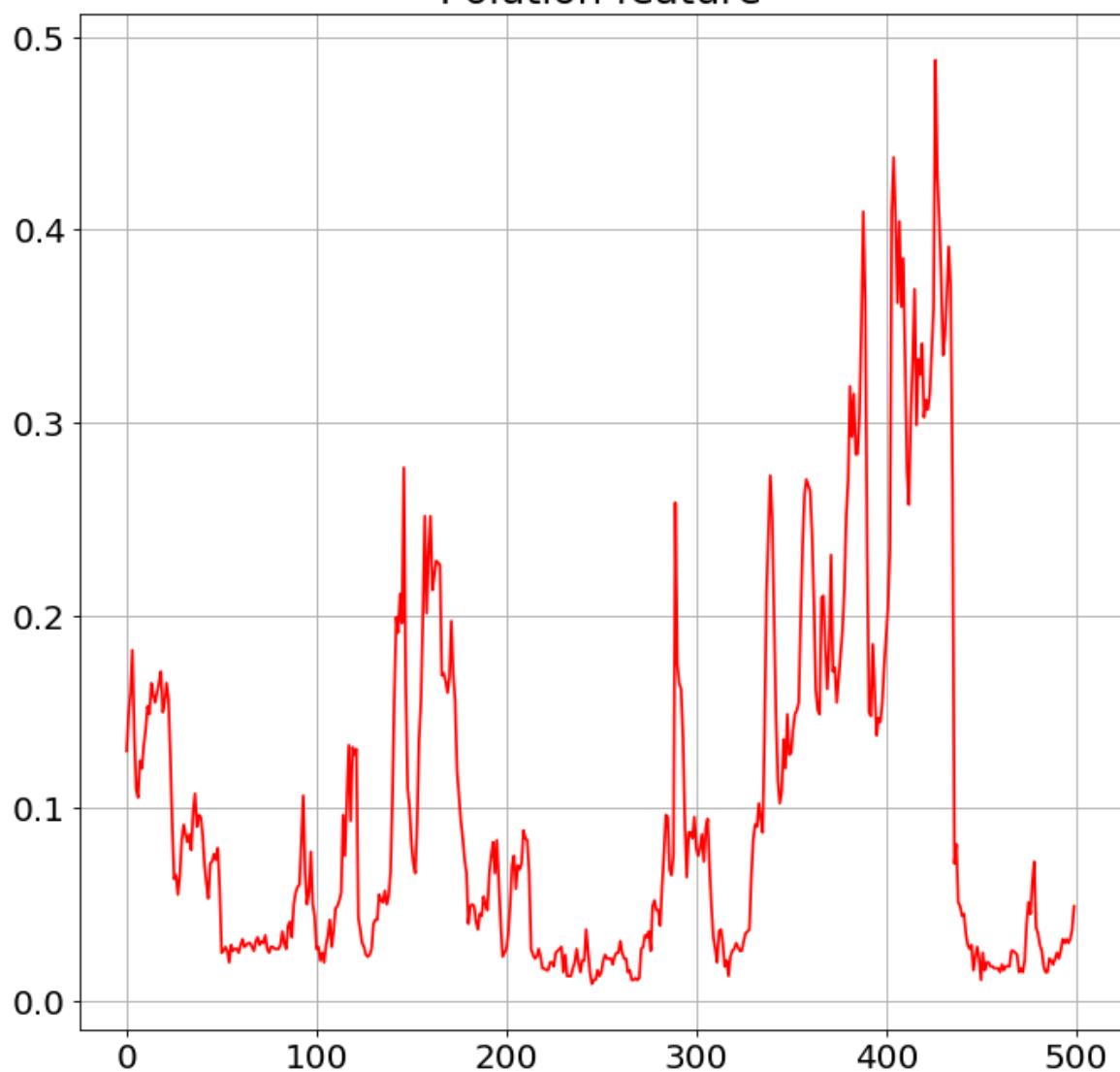
```
1 # Importing Dataset
2 dataset = np.load('polution_dataSet.npy')
3 Num_data, Num_features = dataset.shape
4 print(f"Dataset shape: {dataset.shape}")
```

Dataset shape: (43799, 8)

شکل 2-0: خواندن دیتاست

سپس فیچر آلودگی (pollution) که قرار است آن را پیش بینی کنیم را برای 500 داده نخست رسم می کنیم که در شکل 3-0 نشان داده شده است.

Polution feature



شکل ۰-۳: ویژگی آلودگی دیتاست

سپس به صورت شکل ۰-۴ ده درصد داده های دیتاست را به داده های test و ۹۰٪ را به داده های اختصاص می دهیم.

```

1 # Splitting dataset
2 Num_train_percentage = 0.9
3 Num_train = int(0.9*Num_data)
4
5 train = dataset[:Num_train,:]
6 test = dataset[Num_train:,:]
7
8 print(f"Train data shape = {train.shape}")
9 print(f"Test data shape = {test.shape}")

```

شکل ۰-۴: جدا کردن داده های `train` و `test`

توجه داشته باشید ابعاد ورودی لایه های recurrent می بایست به صورت :

$$(N, \text{window}, \text{NumFeatures})$$

باشد که N تعداد داده های `window` ، `train` برابر طول پنجره زمانی است که طبق صورت سوال برابر ۱۱ در نظر می گیریم و `NumFeatures` برابر تعداد فیچرها می باشد. برای این منظورتابع `createDataset` که در شکل ۰-۵ مشاهده می شود را پیاده سازی می کنیم.

```

1 # Creating dataset based on window
2 def createDataset(dataset, window=1, smpl=1, stride=1, predicting_feature=0):
3     dataX, dataY = [], []
4     N, N_feature = dataset.shape
5     Num = int(np.floor((N - window*smpl - 1)/stride))
6     index = np.arange(window) * smpl
7
8     if Num <= 0:
9         print("There is no enough data!")
10        return 0, 0
11
12    for i in range(Num):
13        a = dataset[index, :]
14        dataX.append(a)
15        dataY.append(dataset[i*stride + window*smpl, predicting_feature])
16        index += stride
17
18    return np.array(dataX), np.array(dataY)

```

شکل ۰-۵: تابع `createDataset`

توجه داشته باشید ورودی `smpl` این تابع مشخص کننده این است که داده ها با چه فاصله ای از هم دیگر انتخاب شوند و ورودی `stride` فاصله اولین داده دو سری زمانی متوالی است و ورودی آخر نیز مشخص می کند که کدام فیچر از دیتاست را قرار است پیش بینی کنیم که در این پروژه فیچر صفرم همان آلدگی می باشد. مثلا اگر طول پنجره ۱۱ و `smpl` و `stride` یک باشند سری زمانی اول داده ۱ تا ۱۱ ام و سری زمانی دوم داده ۲ تا ۱۲ خواهد بود.

حال با استفاده از تابع فوق دیتاست های ورودی شبکه را آماده می کنیم که این کار به صورت شکل ۰-۶ انجام می شود.

```
1 # Creating datasets based on window size to feed the network
2 window = 11
3 trainX, trainY = createDataset(train, window)
4 testX, testY = createDataset(test, window)
5 print(f"trainX shape = {trainX.shape} trainY shape = {trainY.shape}")
6 print(f"testX shape = {testX.shape} testY shape = {testY.shape}")
```

```
trainX shape = (39407, 11, 8) trainY shape = (39407,)
testX shape = (4368, 11, 8) testY shape = (4368,)
```

شکل ۰-۶: آماده کردن دیتاست ورودی شبکه

حال شبکه را پیاده سازی می کنیم. در ابتدا لایه recurrent که یکی از سه نوع لایه SimpleRNN، GRU و LSTM می باشد را به مدل اضافه می کنیم. تعداد واحدهای این لایه را $units = 50$ در نظر $Nneurons = 100$ (هر یک شامل ۱۰۰ fully connected) Dense لایه $L = 1$ باشد. سپس به اندازه $relu$ تابع فعالساز اضافه می کنیم و در نهایت لایه Dense با یک نورون و تابع فعالساز خطی (linear) را به عنوان لایه خروجی و پیش بینی آلدگی در ساعت ۱۲ ام (یک ساعت بعد از پنجره زمانی) به model اضافه می کنیم.

بخش ۱) نمودارهای مقادیر حقیقی و پیش بینی train و test

در قسمت های بعدی پروژه برای رسم این نمودارها پس از آموزش شبکه ابتدا با استفاده از متدهای predict، خروجی مدل برای داده های train و test را بدست آورده (شکل ۱-۱) و سپس آنها را به مقادیر واقعی شان (عمل عکس normalize) تبدیل کرده و به همراه داده های اصلی به صورت کد شکل ۱-۲ رسم می کنیم. لازم به ذکر است برای اینکه مقادیر مربوط به یک زمان خاص بر روی هم منطبق شوند مقادیر واقعی را به اندازه window به راست شیفت می دهیم.

```
1 # make predictions
2 trainPredict = model.predict(trainX)
3 testPredict = model.predict(testX)
```

شکل ۱-۱: بدست آوردن خروجی شبکه به ازای داده های train و test

```
1 # Train
2 if drop_use == True:
3     DROPUSE = "using DropOut Layers"
4 else:
5     DROPUSE = "without using DropOut Layers"
6
7 M = 100
8 plot_range = np.arange(M)*12
9
10 if Recurrent_Layer == 0:
11     RECURRENT = "RNN"
12 elif Recurrent_Layer == 1:
13     RECURRENT = "LSTM"
14 elif Recurrent_Layer == 2:
15     RECURRENT = "GRU"
16
17 plt.figure(num=None, figsize=(10, 10), dpi=80,
18             facecolor='w', edgecolor='w')
19 font = {'weight' : 'normal',
20         'size'   : 18}
21 plt.rc('font', **font)
22 plt.plot(trainPredict[plot_range], 'b-', label = 'Predicted')
23 plt.plot(trainY[plot_range], 'r-', label = 'Actual')
24 plt.grid('on')
25 plt.title(f"Predicted and Actual Train Data {DROPUSE}\n using {RECURRENT} Layer, Optimizer = {{")
26 plt.xlabel("hours 12 and 24")
27 plt.legend(loc="upper right")
```

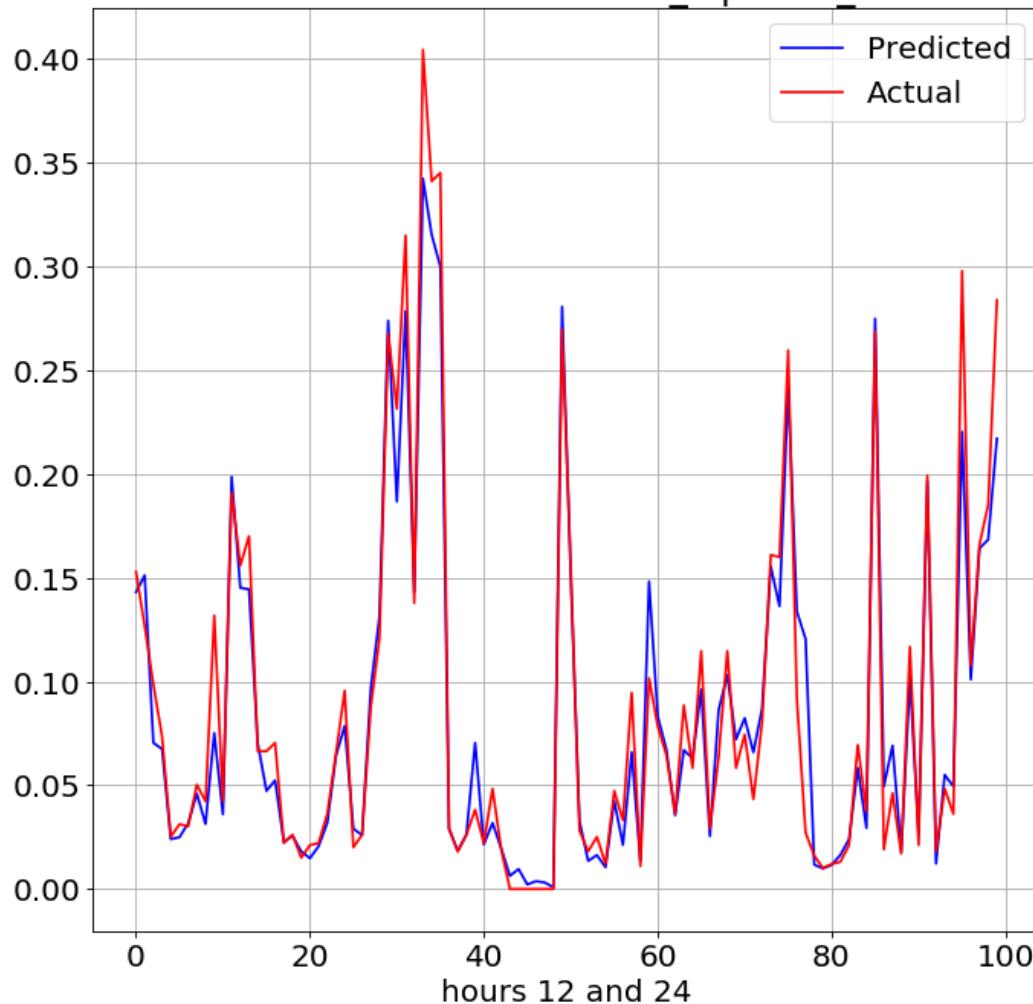
شکل ۱-۲: نحوه رسم مقادیر پیش بینی شده به همراه داده های اصلی

بخش 2) مقایسه دقت و سرعت لایه های RNN, LSTM, GRU

تعداد epoch ها را برابر 25 ، تابع loss را adam optimizer و MSE Batch size را 32 در نظر می گیریم و شبکه را جدآگانه با لایه های RNN و LSTM و GRU اجرا می کنیم و نمودارهای پیش بینی را برای هر حالت مطابق آنچه که در قسمت 1 پژوهه توضیح دادیم برای 200 داده رسم کرده و همچنین زمان اجرای کد برای آموزش شبکه را برای هر حالت اندازه گیری کرده و در نهایت نمودار Loss را بر حسب epoch برای هر سه شبکه در یک نمودار رسم می کنیم و سپس عملکردشان را مقایسه می کنیم.

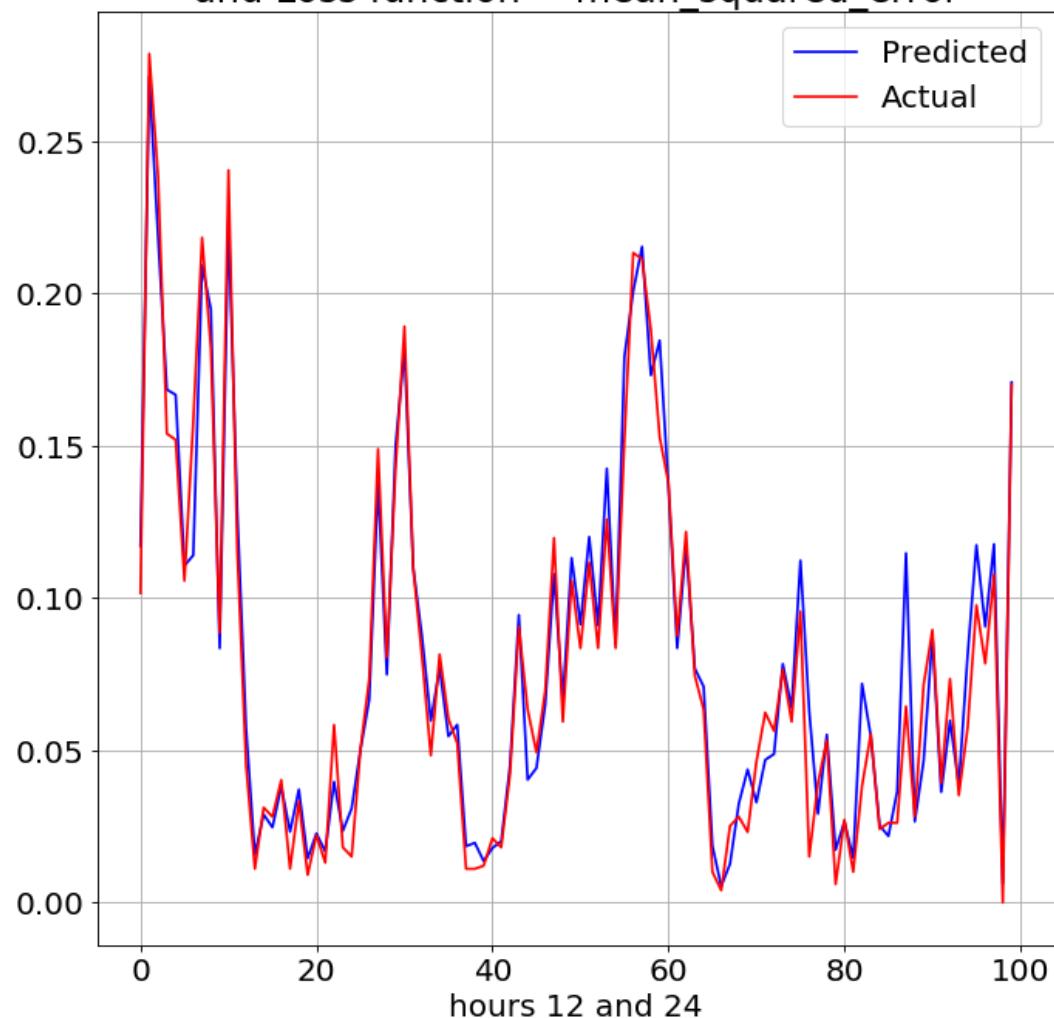
ابتدا لایه RNN را به عنوان لایه Recurrent در نظر می گیریم. نمودار پیش بینی داده های train و test به ترتیب در شکل های 2-1 و 2-2 رسم شده است.

Predicted and Actual Train Data without using DropOut Layers
using RNN Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 2-1: داده های train پیش بینی شده و واقعی با استفاده از لایه RNN

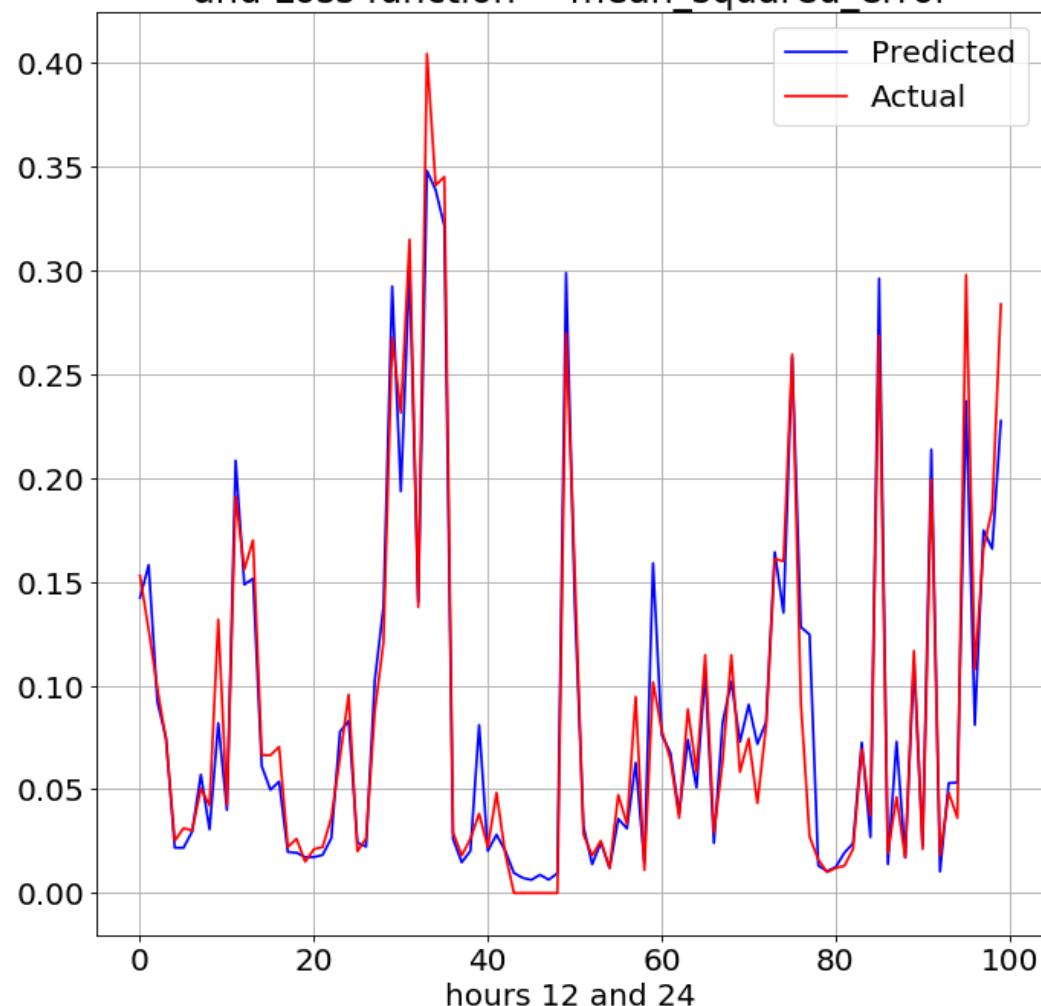
Predicted and Actual Test Data without using DropOut Layers
using RNN Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 2-2: داده های test پیش بینی شده و واقعی با استفاده از لایه RNN

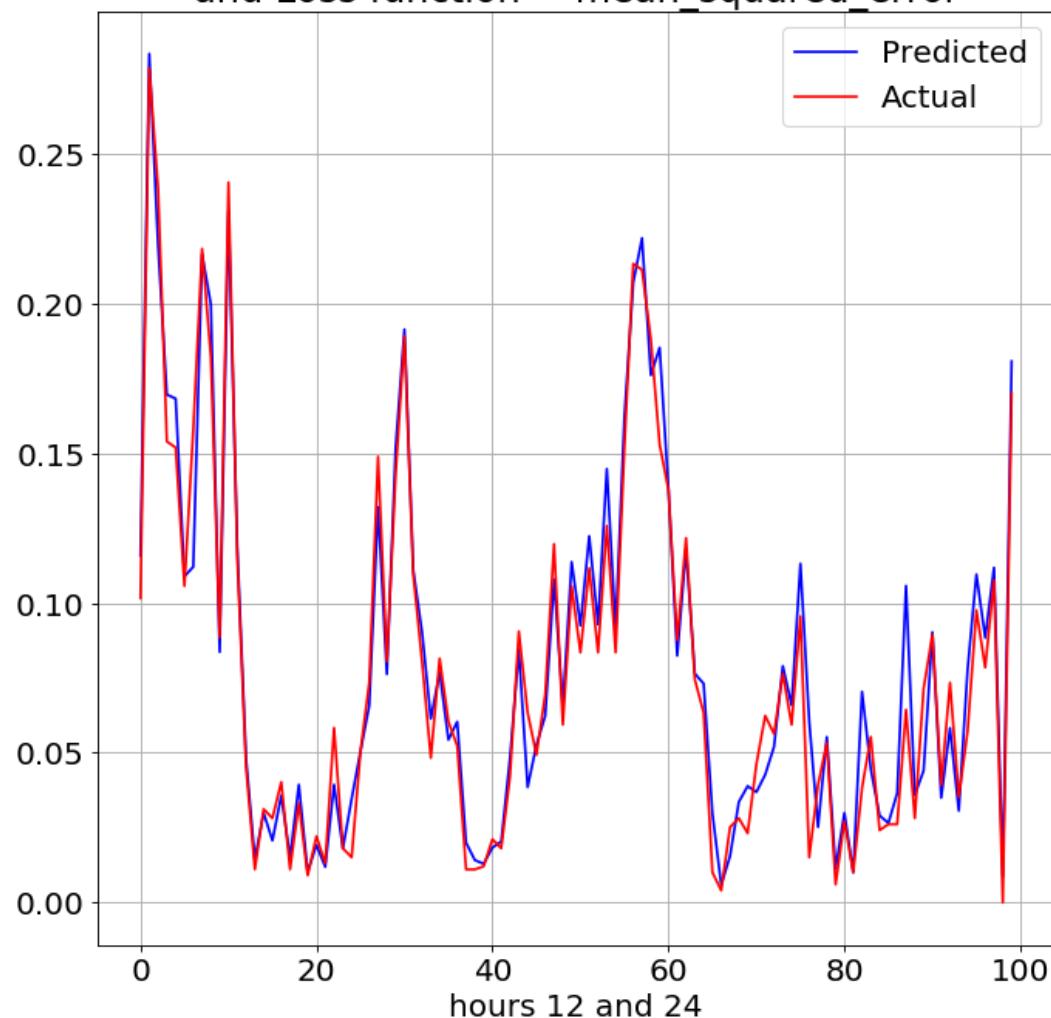
حال شبکه را به کمک لایه LSTM اجرا می کنیم. نمودارهای train و test در شکل های 2-3 و 2-4 رسم شده است.

Predicted and Actual Train Data without using DropOut Layers
using LSTM Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 2-3: داده های train پیش بینی شده و واقعی با استفاده از لایه LSTM

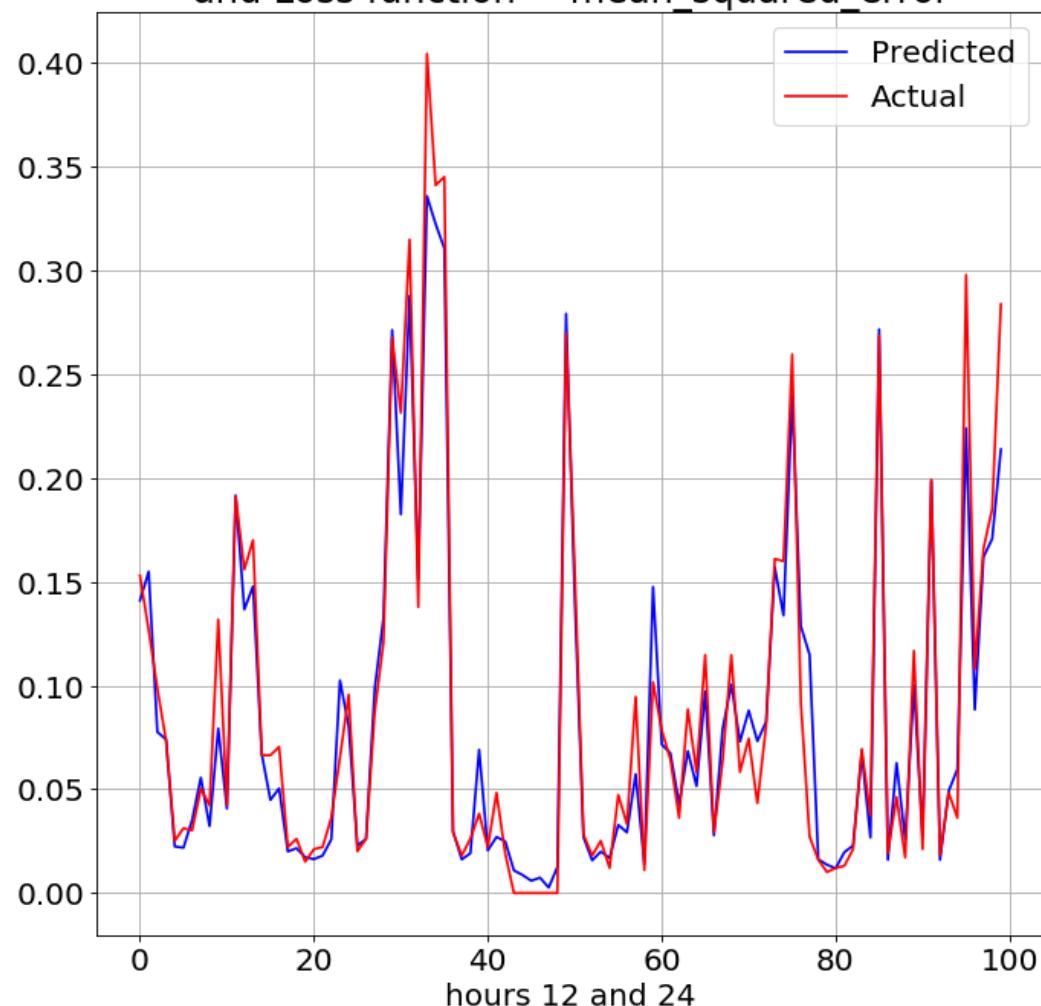
Predicted and Actual Test Data without using DropOut Layers
using LSTM Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل ۲-۴: داده های `test` پیش بینی شده و واقعی با استفاده از لایه LSTM

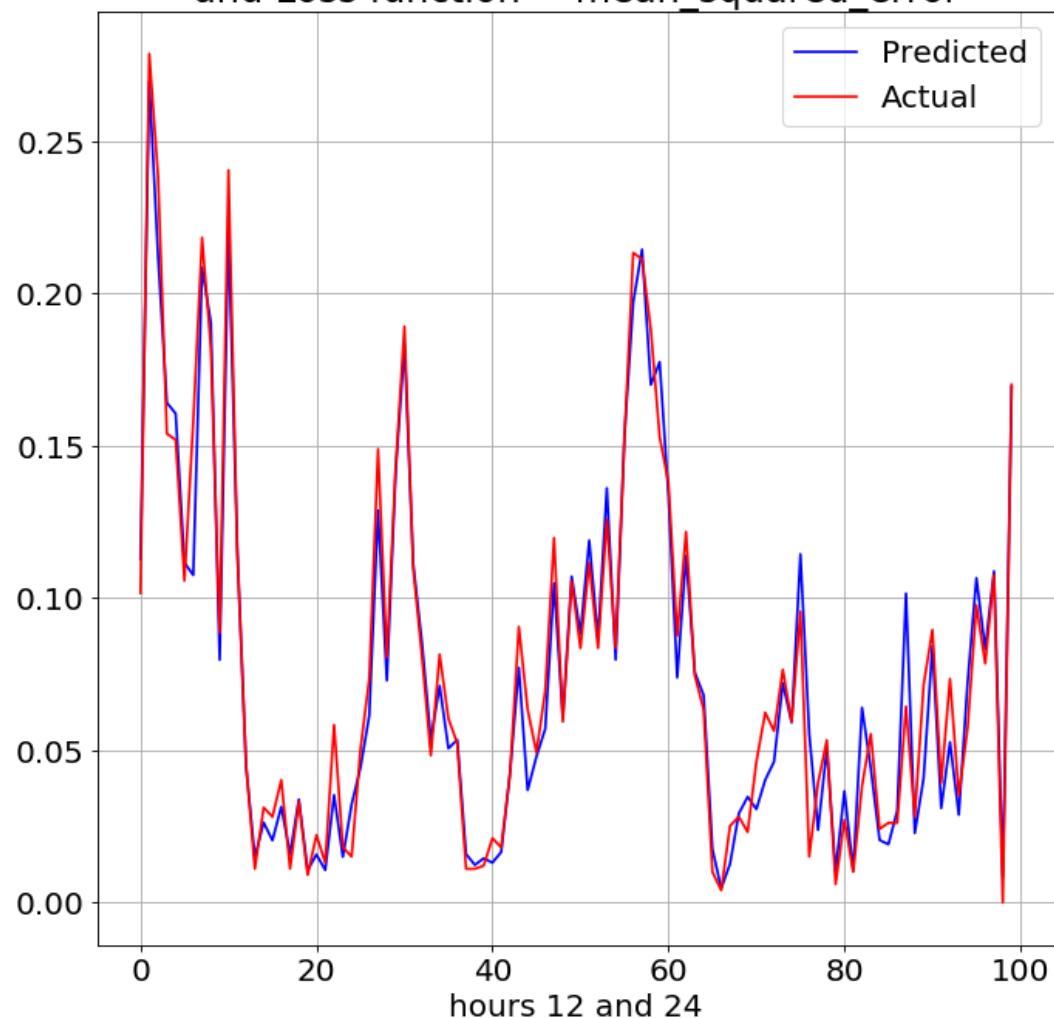
حال از لایه GRU استفاده می کنیم. نمودارهای مربوطه در شکل های ۲-۵ و ۲-۶ رسم شده اند.

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



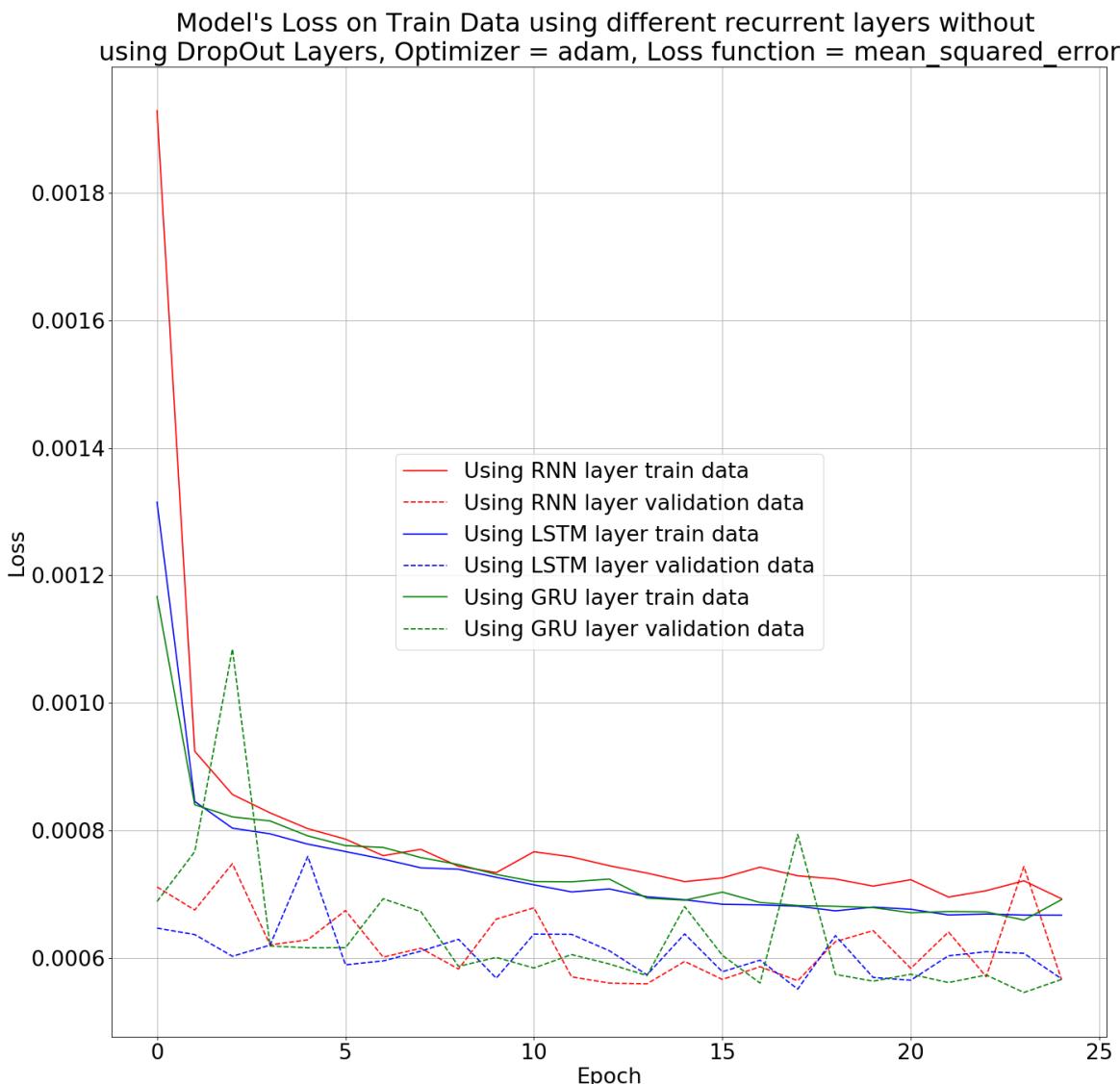
شکل ۲-۵: داده های train پیش بینی شده و واقعی با استفاده از لایه GRU

Predicted and Actual Test Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 6-6: داده های **test** پیش بینی شده و واقعی با استفاده از لایه **GRU**

نمودار Loss سه شبکه مذکور در شکل 7-2 بر حسب epoch رسم شده است (نمودار های خط چین مربوط به داده های validation هستند).



شکل 2-7: نمودار loss سه شبکه با لایه های recurrent

همچنین مقدار loss روی داده های test سه شبکه فوق و زمان اجرای آموزش سه شبکه مذکور بر حسب ثانیه در شکل 2-8 نشان داده شده است.

RNN layer:

Time taken = [102.5763590335846] seconds

Test evaluation:

MSE=0.0004968974855208737, MAE=0.0004968975554220378

LSTM layer:

Time taken = [195.58380937576294] seconds

Test evaluation:

MSE=0.0005318282384252645, MAE=0.0005318281473591924

GRU layer:

Time taken = [180.7485213279724] seconds

Test evaluation:

MSE=0.0004998963947630497, MAE=0.0004998964723199606

شکل 2-8: مقدار loss روی داده های test و زمان لازم برای آموزش شبکه برای لایه های Recurrent مختلف

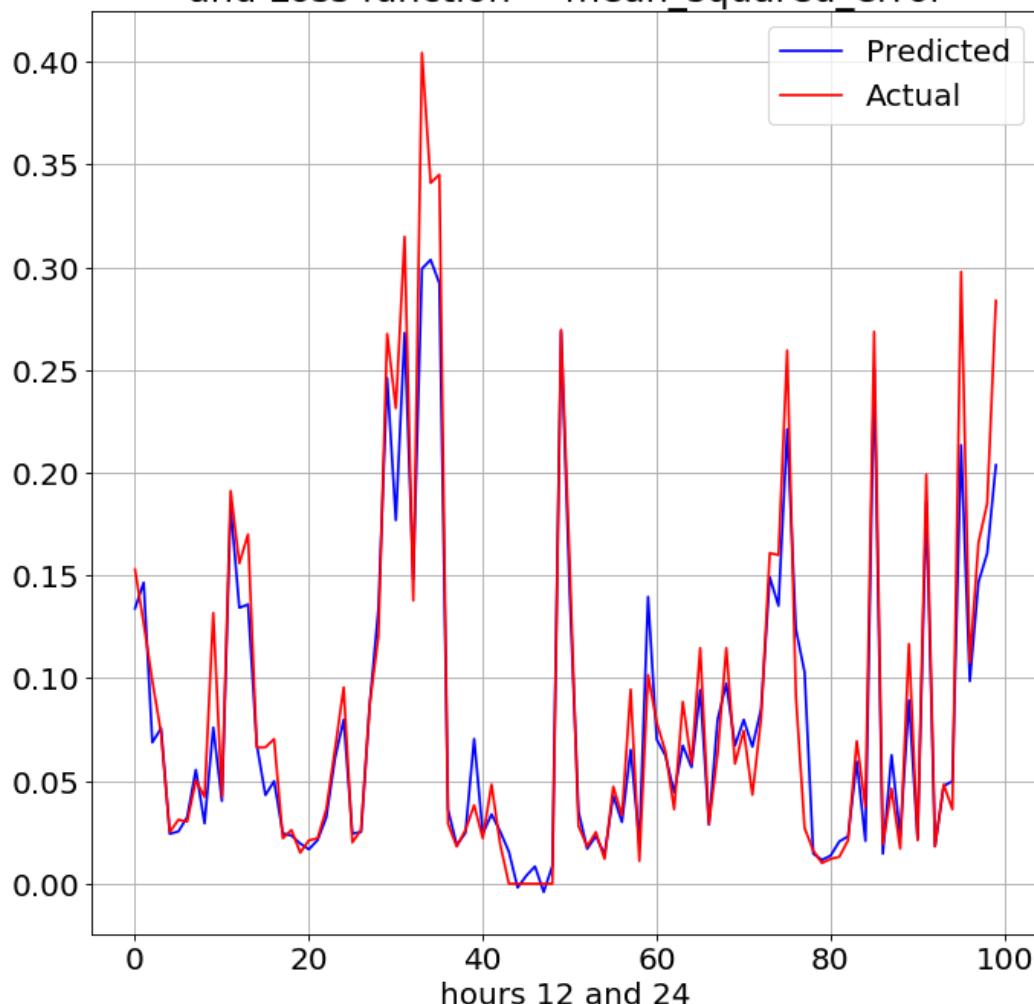
با توجه به شکل 2-7 مشاهده می شود که مقدار loss در صورت استفاده از لایه GRU در اکثر epoch ها کمتر از LSTM و RNN است و LSTM نیز کمتر از RNN است. زمان اجرای آموزش به کمک GRU نیز با توجه به شکل 2-8 از بقیه کمتر است. بنابرین نتیجه می گیریم عملکرد شبکه در صورت استفاده از لایه GRU از لحاظ زمانی کمتر از lstm و مقدار Loss آن نیز بهتر است. پس به ترتیب لایه های GRU و RNN و LSTM عملکرد بهتری دارند.

بخش 3) مقایسه دقت و سرعت با توابع بهینه سازی مختلف

در این قسمت مشابه قسمت قبل عمل می کنیم اما این بار تابع بهینه سازی را تغییر می دهیم و عملکرد شبکه در صورت استفاده از توابع بهینه سازی مختلف (RMSprop، Adagrad، Adam و GRU) را بررسی خواهیم کرد. تعداد epoch ها را مشابه قبل مقدار ثابت 25 ، تابع loss را MSE و لایه Recurrent را نیز GRU انتخاب می کنیم (چون در قسمت قبل عملکرد بهتری داشت) ، نتایج (روندهای نزولی بودن نمودارهای loss) با استفاده از دو لایه دیگر LSTM و RNN مشابه است.

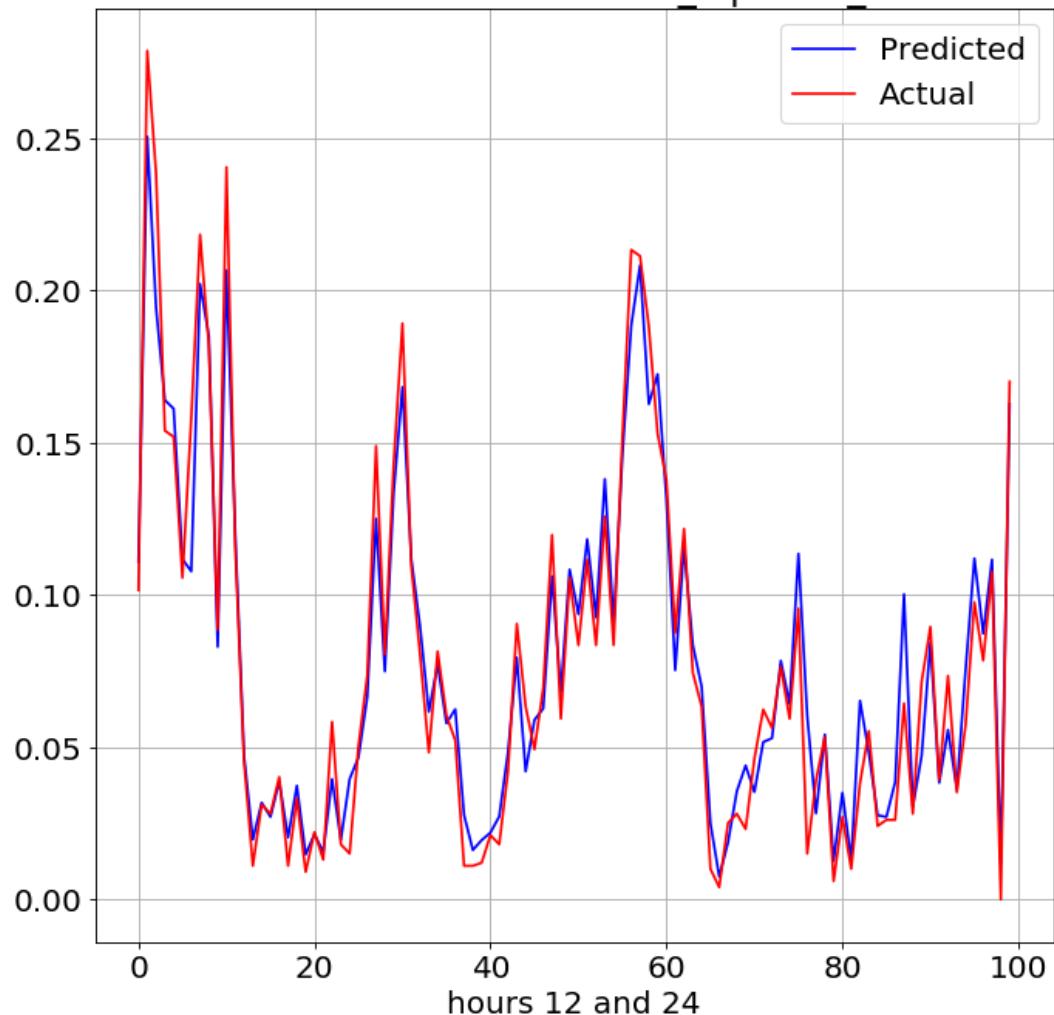
نمودار های پیش بینی در صورت استفاده از تابع بهینه سازی adam به صورت شکل های 3-1 و 3-2 خواهد بود:

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



3-1 شکل

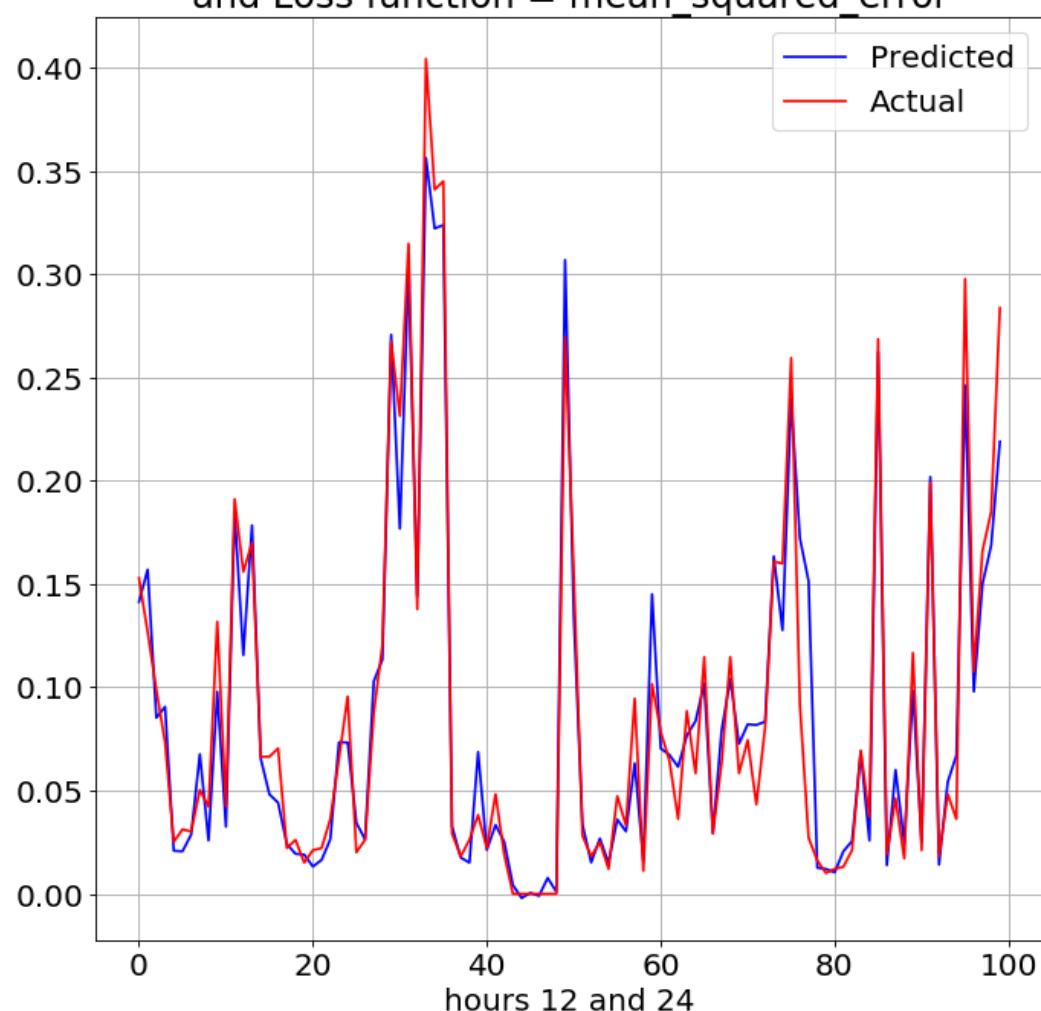
Predicted and Actual Test Data without using DropOut Layers
 using GRU Layer, Optimizer = adam
 and Loss function = mean_squared_error



شکل 3-2

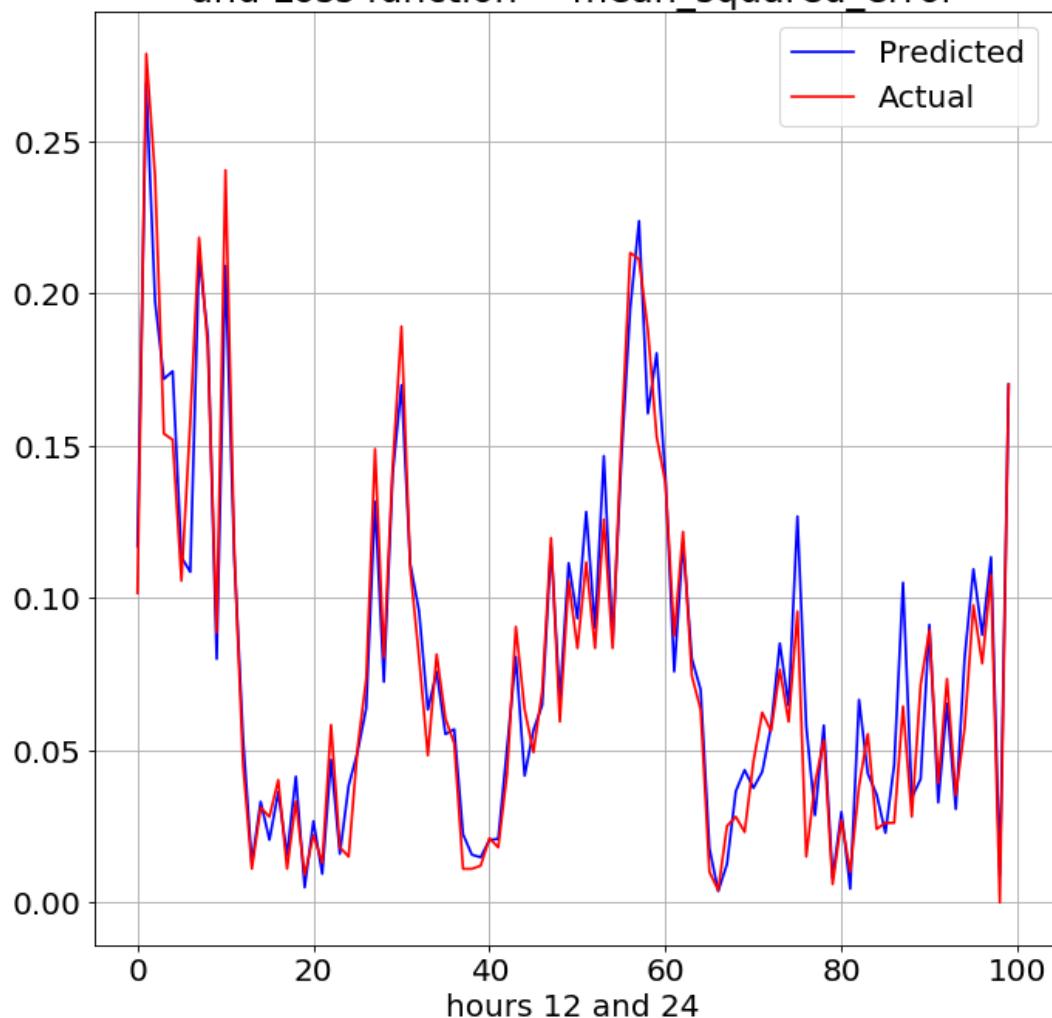
حال از تابع بهینه سازی Adagrad استفاده می کنیم. نمودارهای مربوطه در شکل های 3-3 و 3-4 قابل مشاهده هستند.

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = Adagrad
and Loss function = mean_squared_error



شكل 3-3

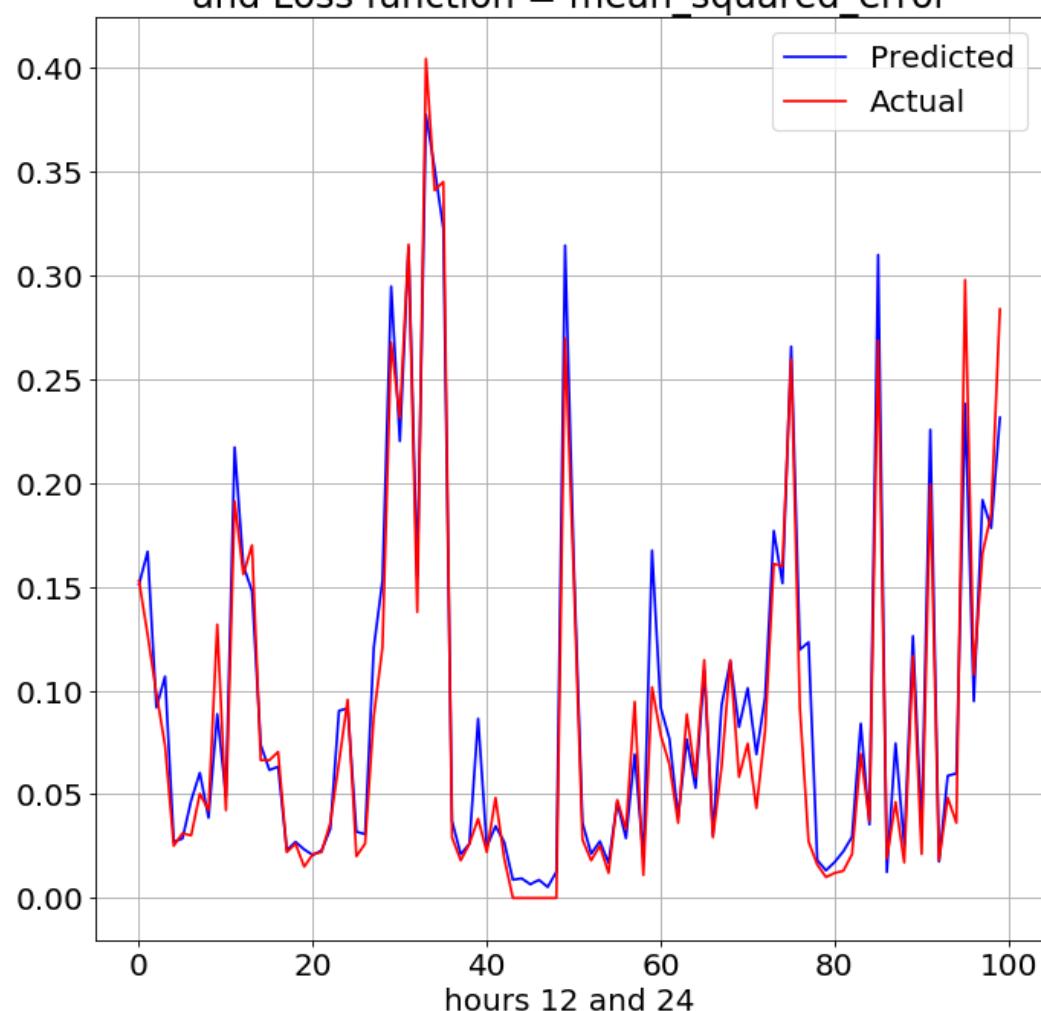
Predicted and Actual Test Data without using DropOut Layers
using GRU Layer, Optimizer = Adagrad
and Loss function = mean_squared_error



شکل 3-4

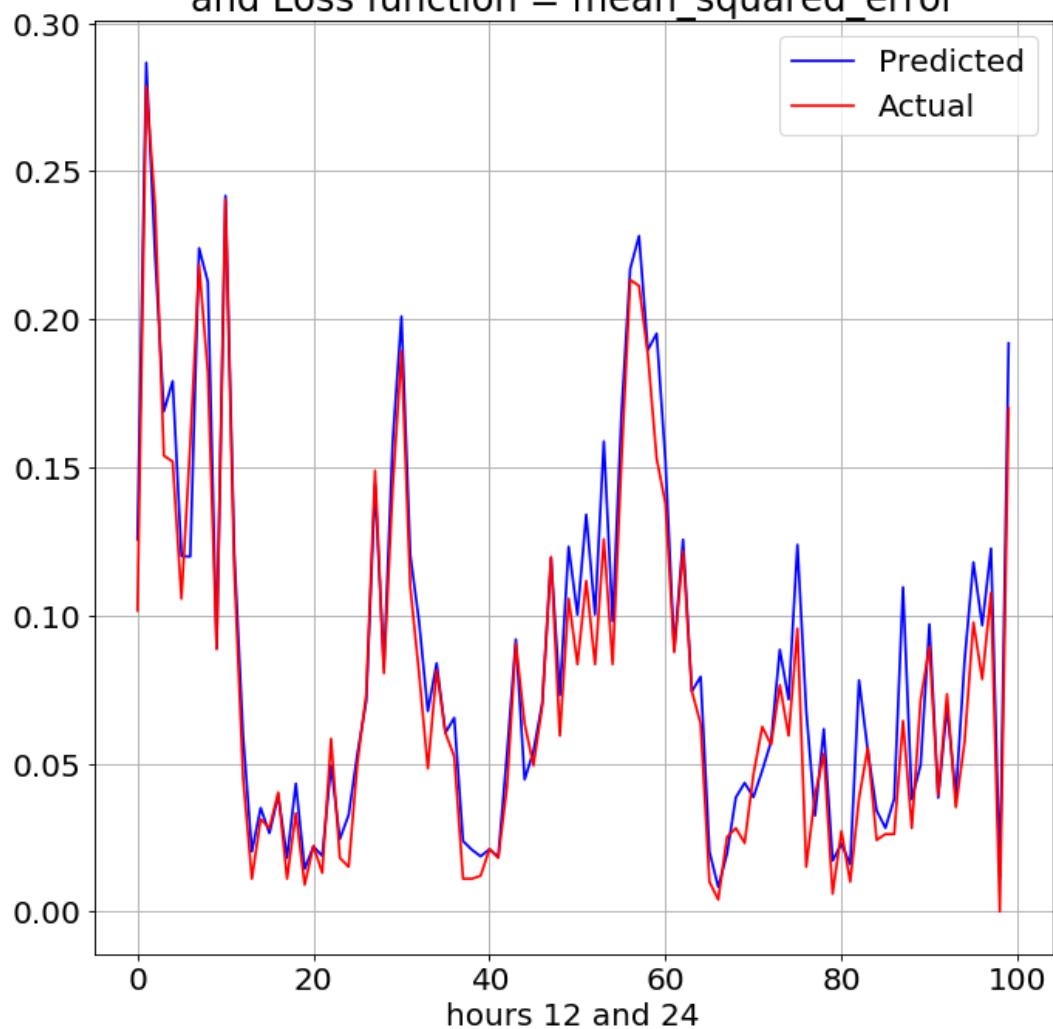
حال از تابع بهینه سازی RMSProp استفاده می کنیم. نمودارها در شکل های 3-5 و 3-6 قابل مشاهده هستند.

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = RMSprop
and Loss function = mean_squared_error



شكل 3-5

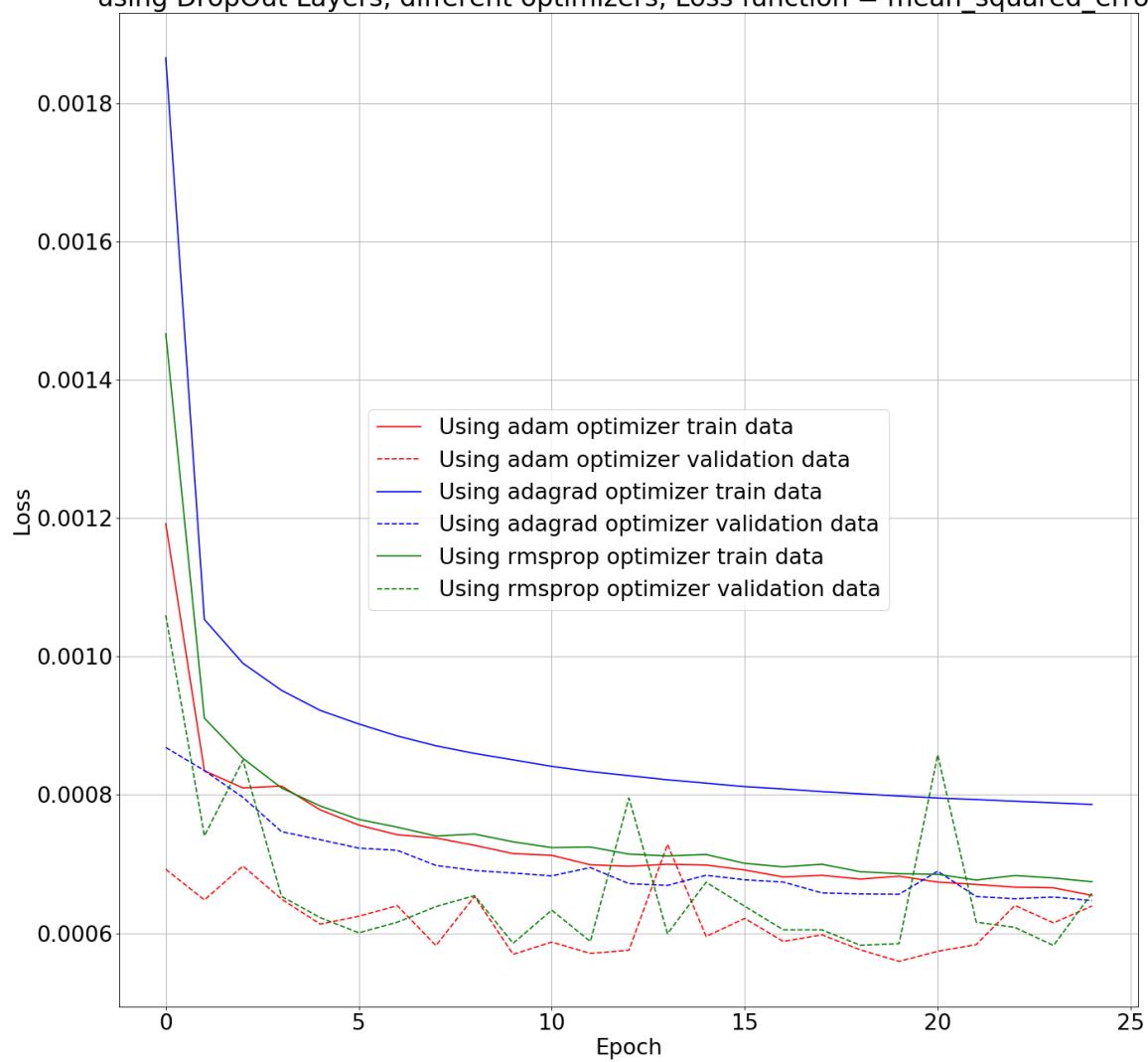
Predicted and Actual Test Data without using DropOut Layers
using GRU Layer, Optimizer = RMSprop
and Loss function = mean_squared_error



شکل 3-6

نمودار loss سه شبکه مذکور برای لایه GRU در شکل 3-7 قابل مشاهده است:

Model's Loss on Train Data using GRU recurrent layer, without using DropOut Layers, different optimizers, Loss function = mean_squared_error



شکل 7-3: نمودار loss سه شبکه با توابع بهینه سازی مختلف

همچنین زمان آموزش سه شبکه فوق بر حسب ثانیه و نتایج روی داده های تست در شکل 3-8 قابل مشاهده است:

GRU layer results:

Adam optimizer:

Time taken = 164.44748759269714 seconds

Test evaluation:

MSE = 0.0005907816742035141, MAE = 0.000590781681239605

Adagrad optimizer:

Time taken = 174.2288691997528 seconds

Test evaluation:

MSE = 0.0005181977082659384, MAE = 0.0005181977176107466

RMSprop optimizer:

Time taken = 164.38564944267273 seconds

Test evaluation:

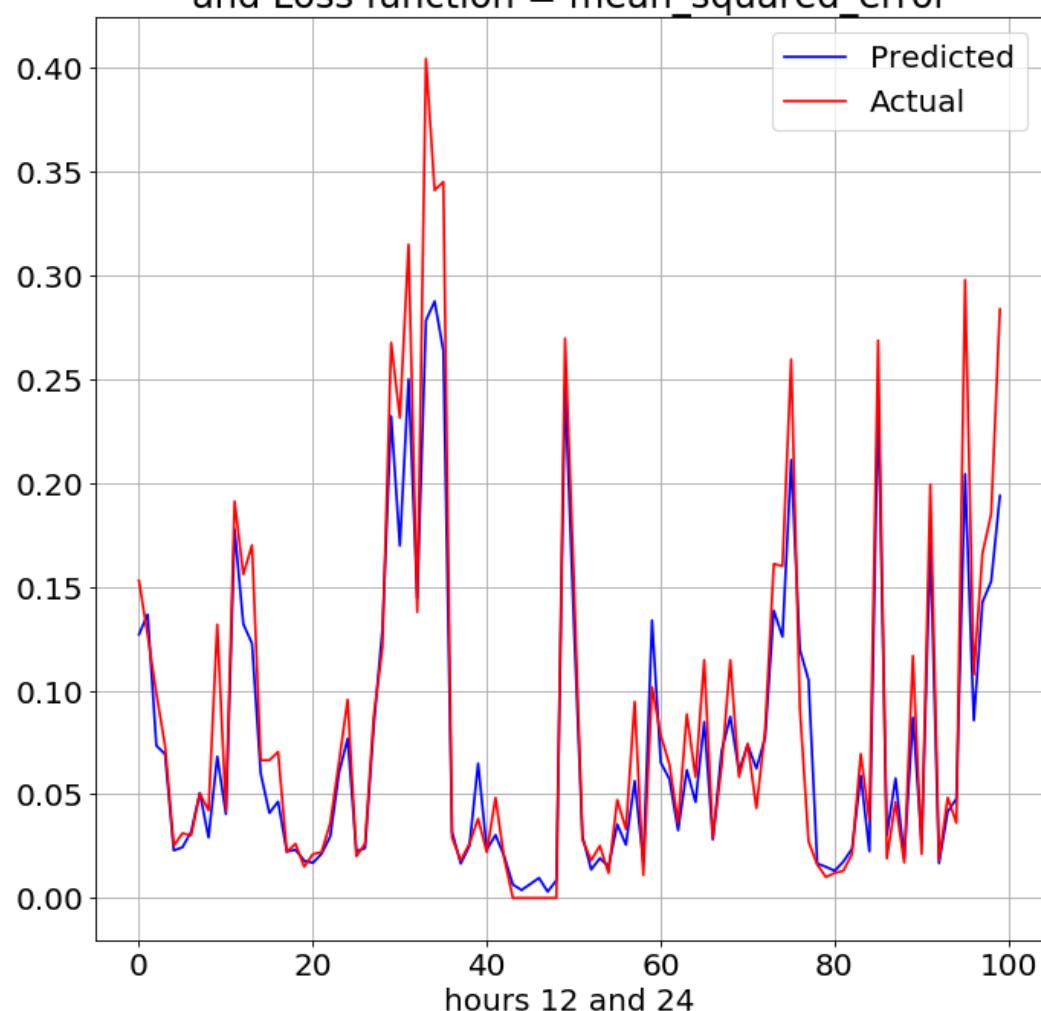
MSE = 0.0005775487009157007, MAE = 0.0005775486933998764

شکل 3-8: زمان اجرای شبکه در صورت استفاده از توابع بهینه سازی مختلف

با توجه به شکل 3-7 و 3-8 نتیجه می گیریم شبکه با تابع بهینه سازی adam شرایط بهتری دارد چون در تمام epoch ها نمودار loss آن طی آموزش پایین تر از rmsprop و adagrad می باشد. نتایج سه شبکه مذکور روی داده های تست طبق شکل 3-8 با تقریب 4 رقم برابر هستند. اما با تقریب 5 رقم و بیشتر شبکه adagrad وضعیت بهتری دارد.

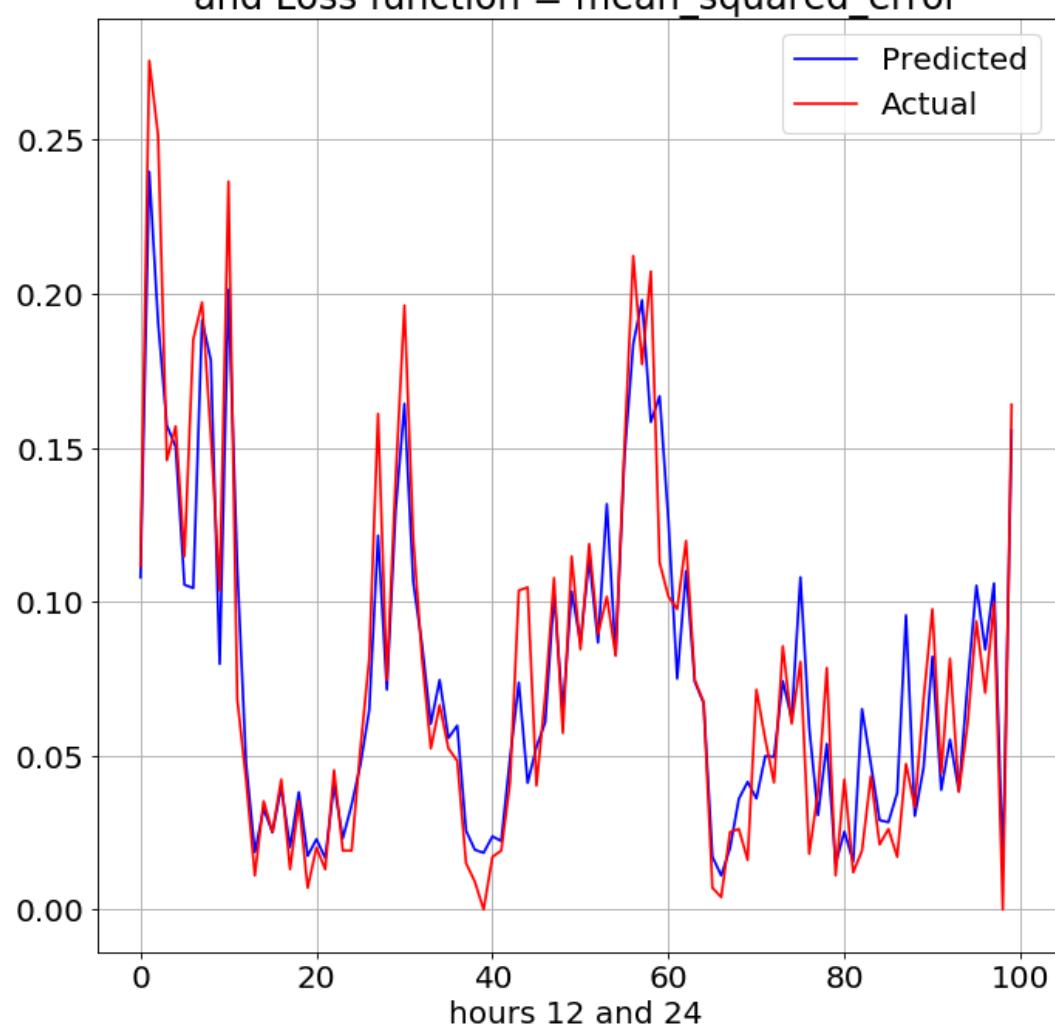
حال تابع loss را تغییر می دهیم (Mean Squared Error و Mean Absolute Error) و نتایج را مشاهده می کنیم. نمودارهای پیش بینی داده های train و test در شکل های 3-9 تا 3-12 قابل مشاهده هستند.

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



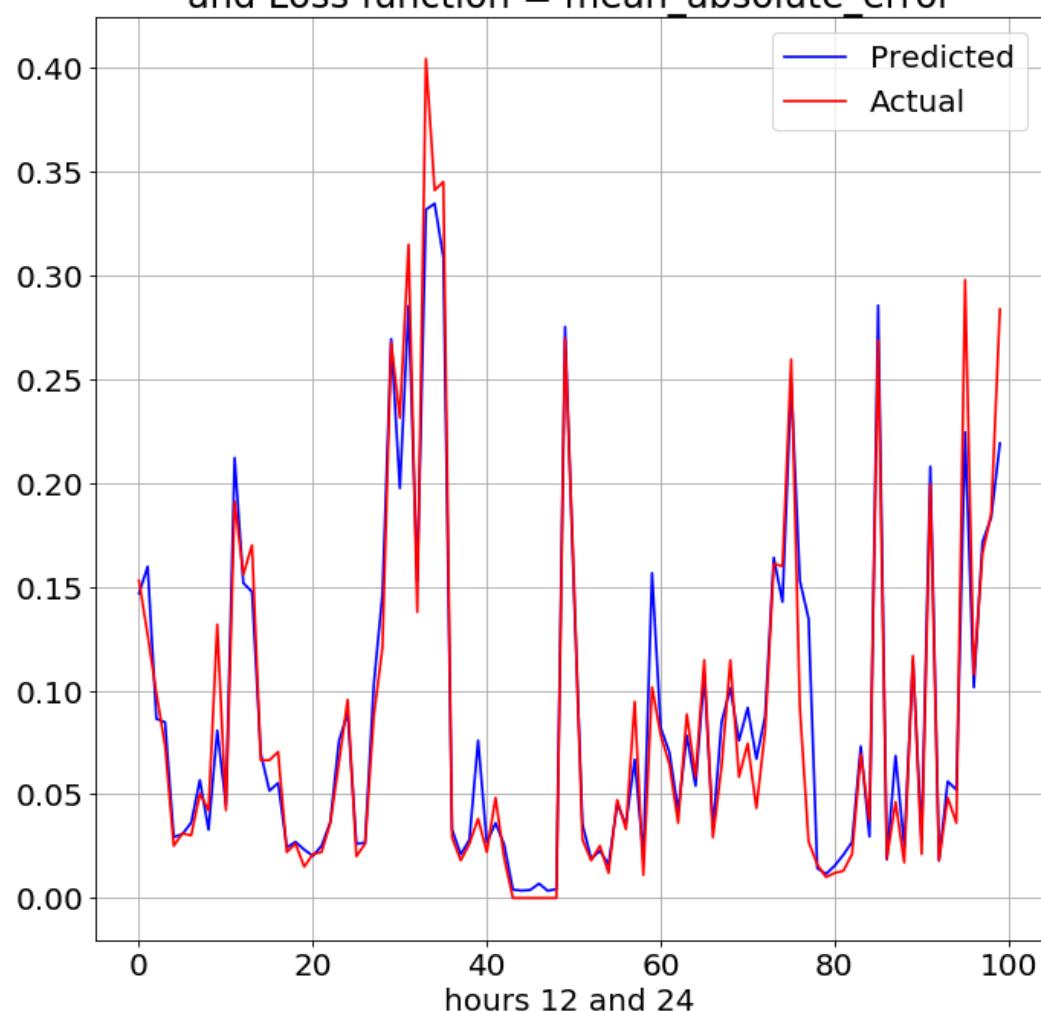
شكل 3-9

Predicted and Actual Test Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



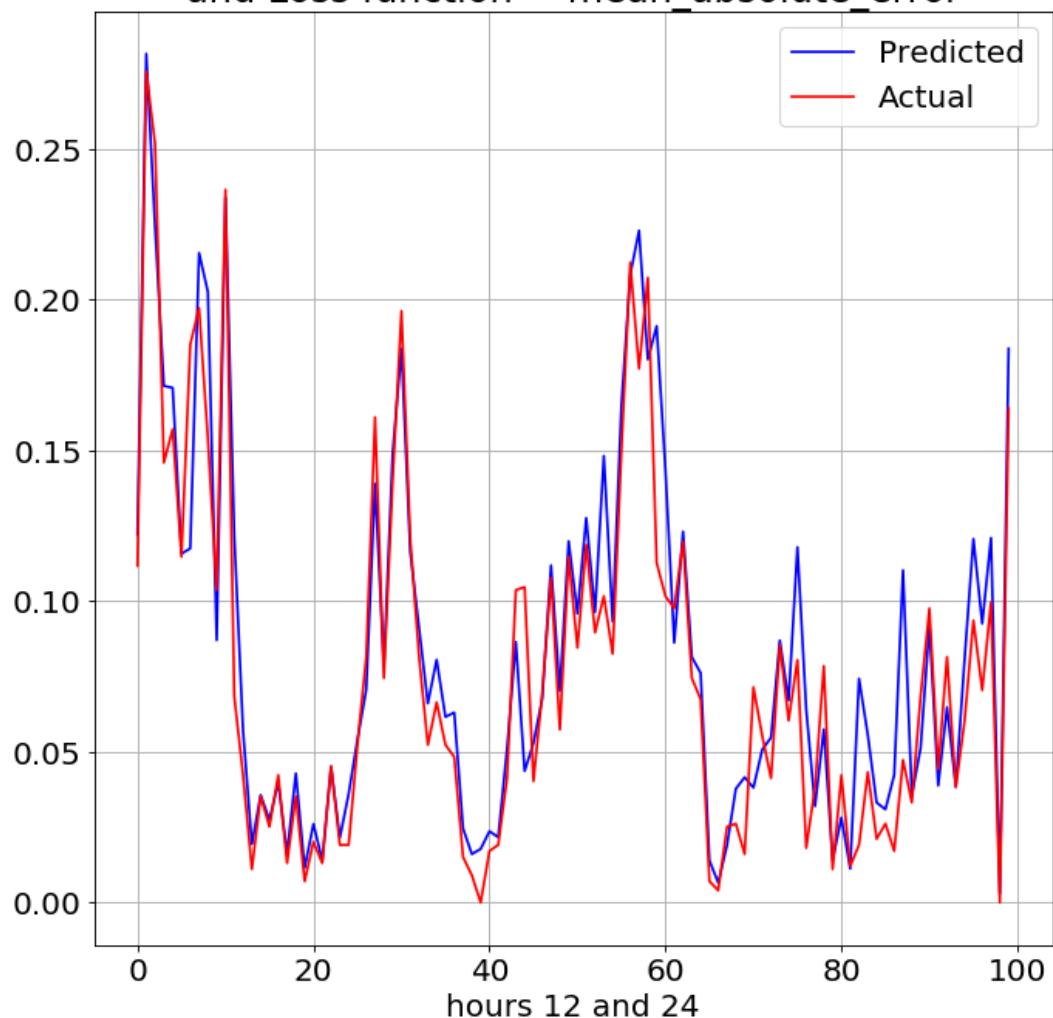
3-10 شکل

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_absolute_error



شكل 3-11

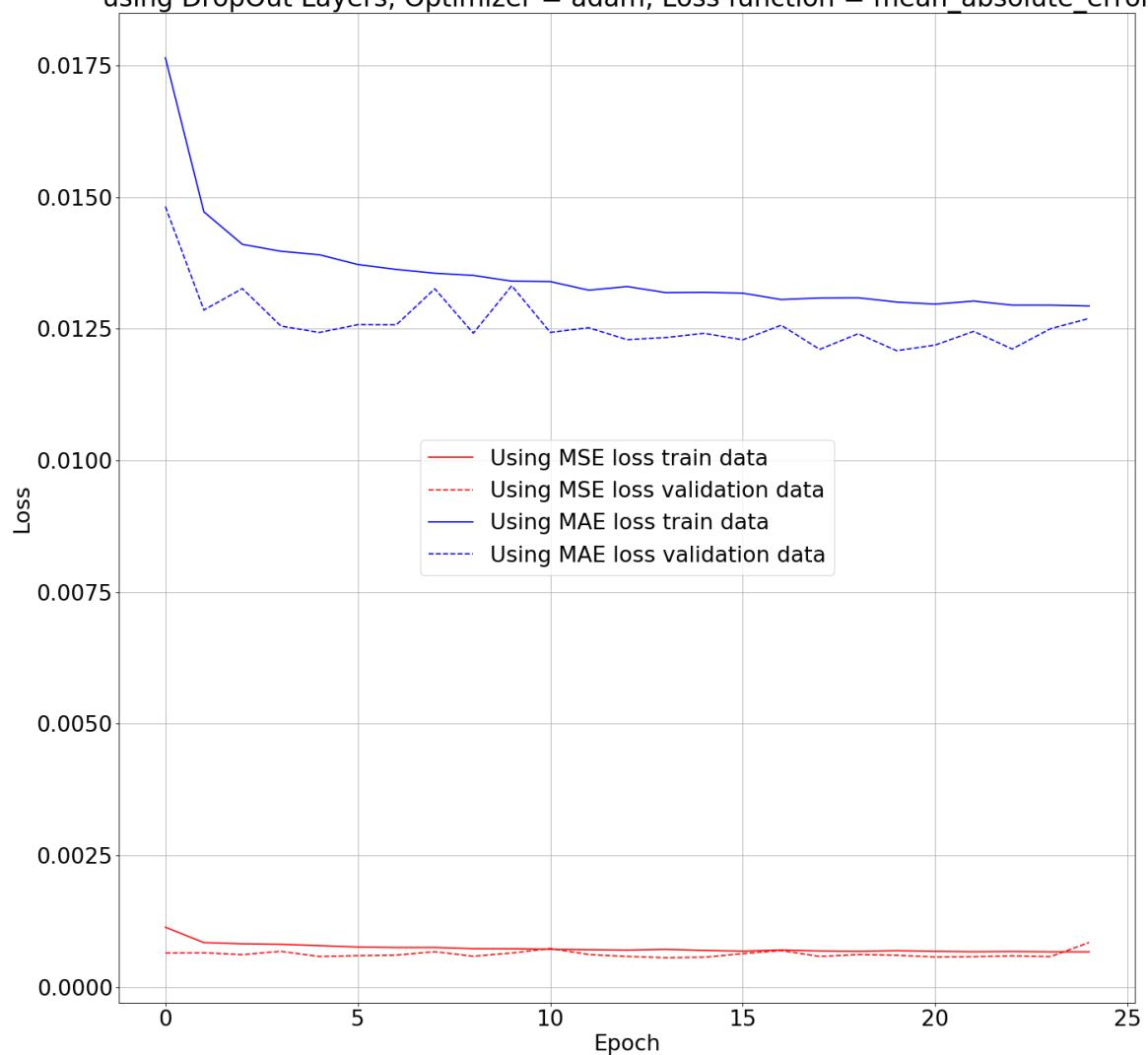
Predicted and Actual Test Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_absolute_error



شکل 3-12

نمودار loss برای دو شبکه فوق در شکل 3-13 قابل مشاهده است:

Model's Loss on Train Data using GRU recurrent layer, without using DropOut Layers, Optimizer = adam, Loss function = mean_absolute_error



شکل 3-13: نمودار loss دو شبکه با تابع loss مختلف

همچنین زمان آموزش دو شبکه و نتایج روی داده های تست در شکل 3-14 قابل مشاهده است:

GRU layer results:

MSE loss function:

Time taken = 157.4828884601593 seconds

Test evaluation:

MSE = 0.0006619819554204127, MAE = 0.0006619818741455674

MAE loss function:

Time taken = 171.610089302063 seconds

Test evaluation:

MSE = 0.011556239423406866, MAE = 0.0004769784864038229

شکل 3-14: زمان اجرای دو شبکه

با توجه به شکل 3-13 و 3-14 خطای MSE بسیار کمتر از MAE است و همچنین زمان اجرای MSE کمتر است در نتیجه تابع خطای MSE عملکرد بسیار بهتری از MAE دارد.

بخش 4) پیش بینی هفتگی و ماهانه

در این قسمت ابتدا یک ساعت رندوم برای پیش بینی هفتگی و یک ساعت و یک روز هفته تصادفی برای پیش بینی ماهانه انتخاب می کنیم که در شکل 4-1 قابل مشاهده است. لازم به ذکر است برای پیش بینی هفتگی فاصله بین داده ها باید یک روز باشد بنابرین ورودی smpl تابع createDataset را برابر 24 و برای پیش بینی ماهانه این فاصله برابر یک هفته است پس طول smpl را برای پیش بینی ماهانه 24×7 قرار می دهیم. همچنین برای اینکه تعداد داده ها به اندازه کافی زیاد باشد طول stride را در هر دو حالت برابر 1 فرض می کنیم.

```
1 # Preparing datasets
2
3 # Choosing random hours and day
4 randomHourWeek = np.random.randint(24, size = 1)
5 randomHourMonth = np.random.randint(24, size = 1)
6 randomDayMonth = np.random.randint(7, size = 1)
7
8 stride_week = 1
9 stride_month = 1
10
11 smpl_week = 24
12 smpl_month = 24 * 7
```

شکل 4-1: انتخاب ساعت و روز تصادفی و تعیین smpl و stride

سپس مطابق شکل 4-2 دیتابست را به دو قسمت train و test برای هر حالت پیش بینی هفتگی یا ماهانه تقسیم می کنیم و طول پنجره را نیز تعیین می کنیم.

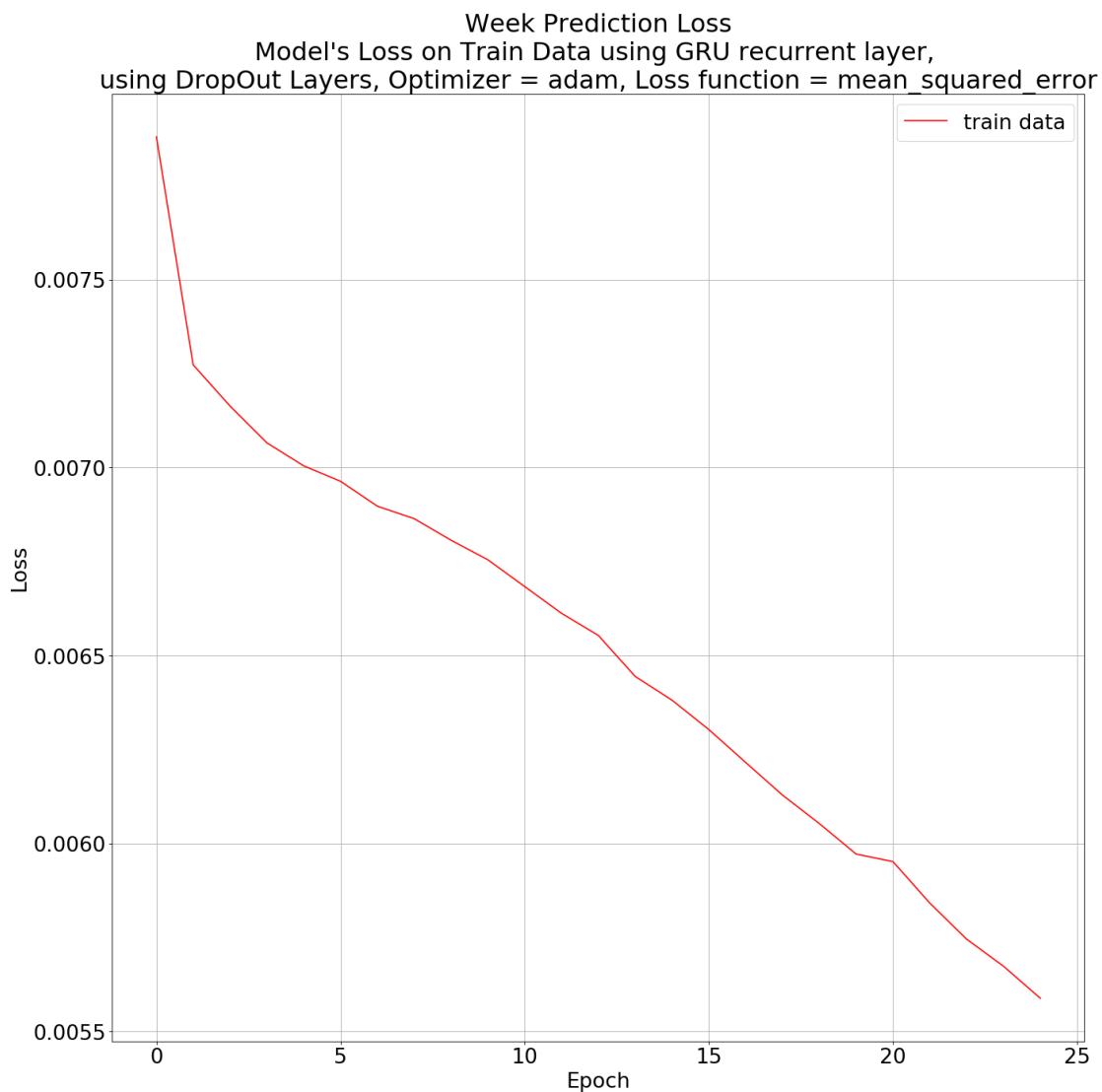
```
1 # Splitting Datasets
2 percentageWeek = 0.8
3 percentageMonth = 0.8
4
5 N_train_week = int(percentageWeek * Num_data)
6 N_train_month = int(percentageMonth * Num_data)
7
8 window_week = 7
9 window_month = 4
10
11 trainWeek = dataset[:N_train_week, :]
12 testWeek = dataset[N_train_week:, :]
13
14 trainMonth = dataset[:N_train_month, :]
15 testMonth= dataset[N_train_month:, :]
```

شکل 4-2 splitting dataset

در ابتدا پیش بینی هفتگی را انجام می دهیم. شبکه را مطابق قسمت های قبل پیاده سازی کرده و کنیم و تعداد epoch ها را مانند قبل 25 در نظر می گیریم. پس از آموزش شبکه مقدار MSE روی داده های تست به صورت شکل 4-3 خواهد بود. همچنین نمودار loss بر حسب epoch برای داده های test در شکل 4-4 قابل مشاهده است.

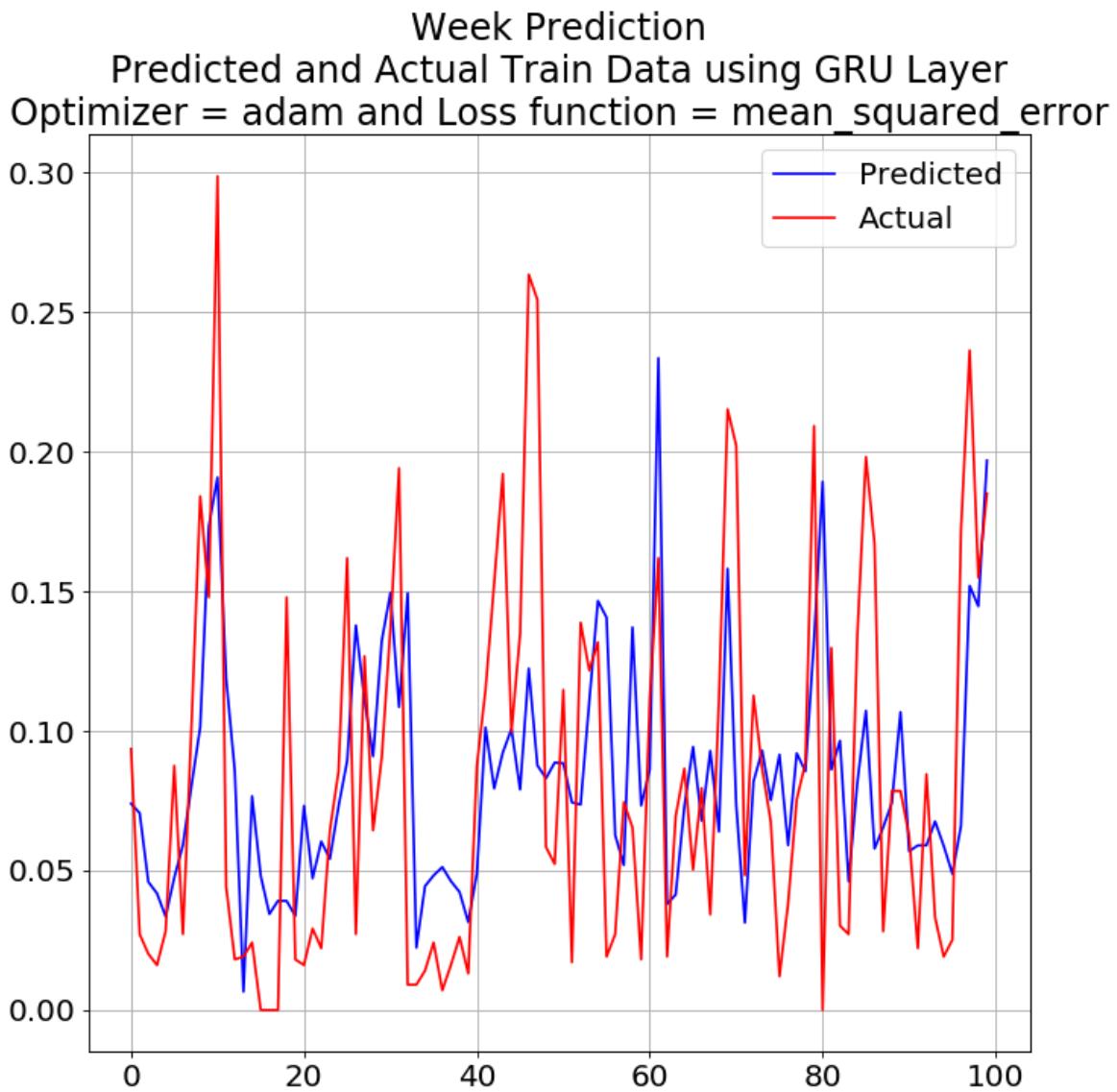
```
1 model.evaluate(testWeekX, testWeekY)
8591/8591 [=====] - 0s 45us/sample - loss: 0.008
3
0.00832090113046843
```

شکل 4-3 خطای MSE روی داده های test



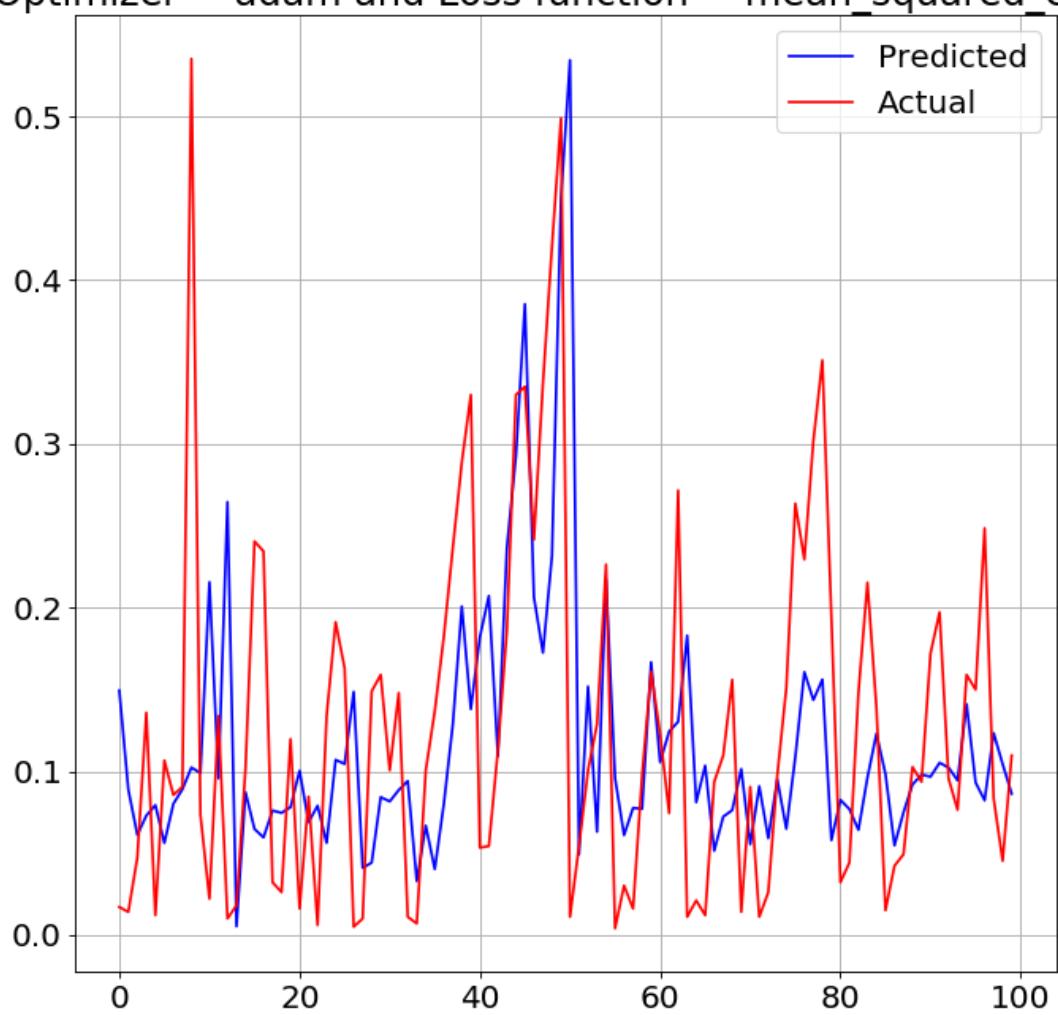
شکل 4-4: نمودار خطای MSE پیش بینی هفتگی بر حسب epoch

نمودار پیش بینی داده های train و test در شکل های 4-5 و 4-6 قابل مشاهده است.



شکل 4-5: پیش بینی داده های train

Week Prediction
Predicted and Actual Test Data using GRU Layer
Optimizer = adam and Loss function = mean_squared_error



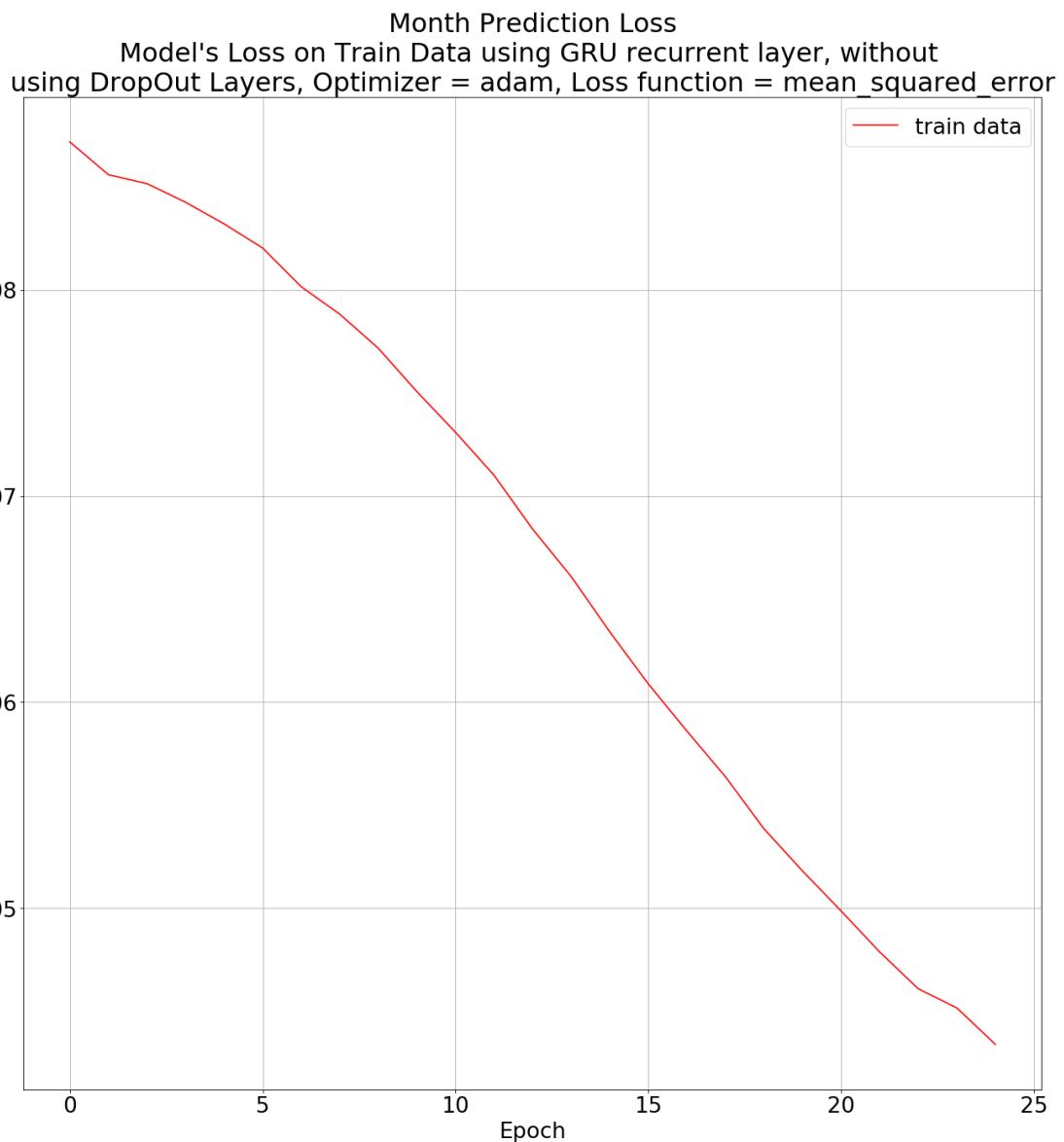
شکل 4-6: پیش بینی داده های test

حال پیش بینی ماهانه را انجام می دهیم. پس از آموزش شبکه مقدار loss برای داده های test و نمودار خطای داده های train بر حسب epoch به ترتیب به صورت شکل های 4-7 و 4-8 خواهد بود:

```
1 model.evaluate(testMonthX, testMonthY)
```

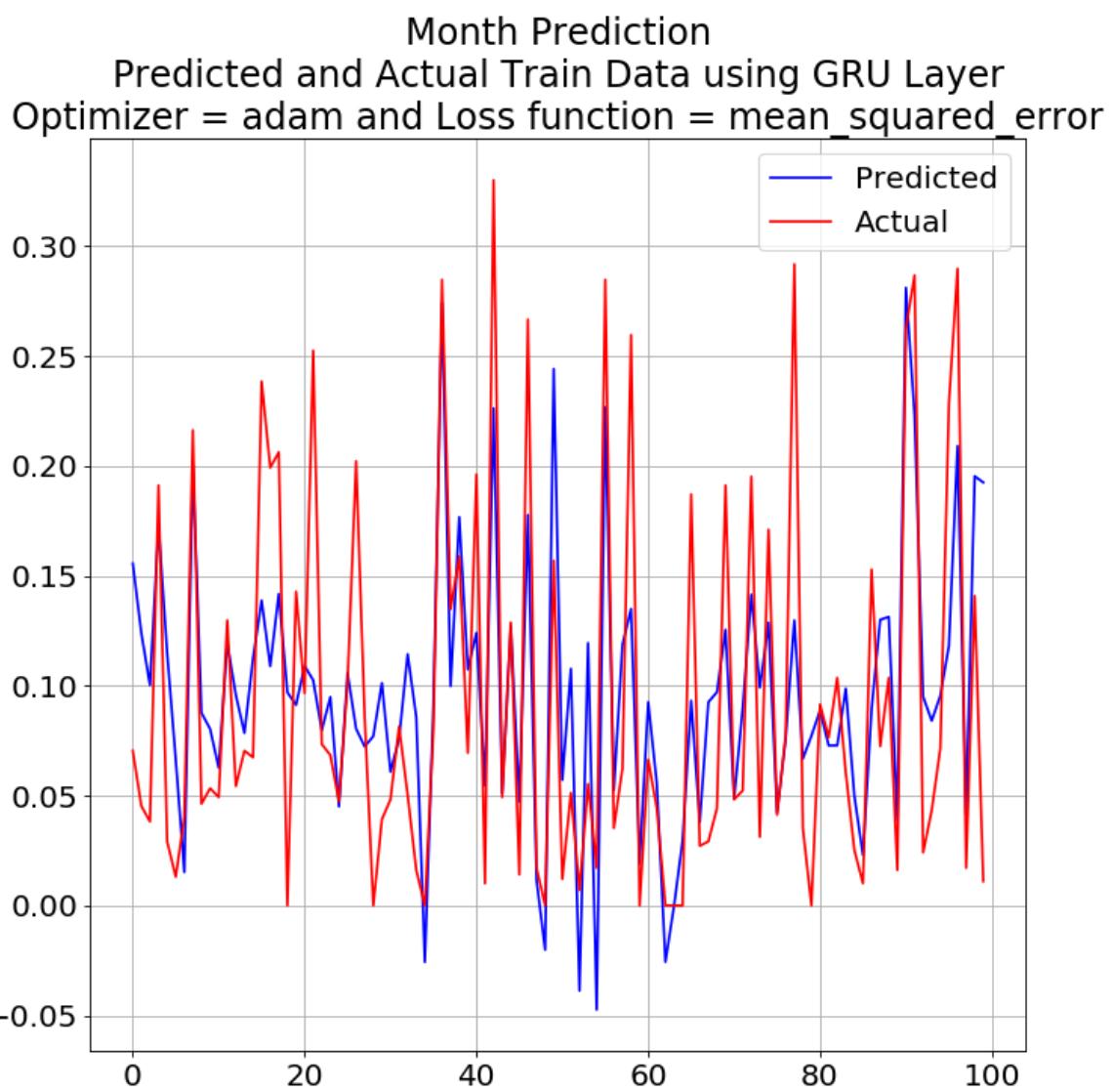
```
34366/34366 [=====] - 1s 35us/sample - loss: 0.004
6
0.004557538782522019
```

شکل 4-7: مقدار loss داده های test پیش بینی ماهانه

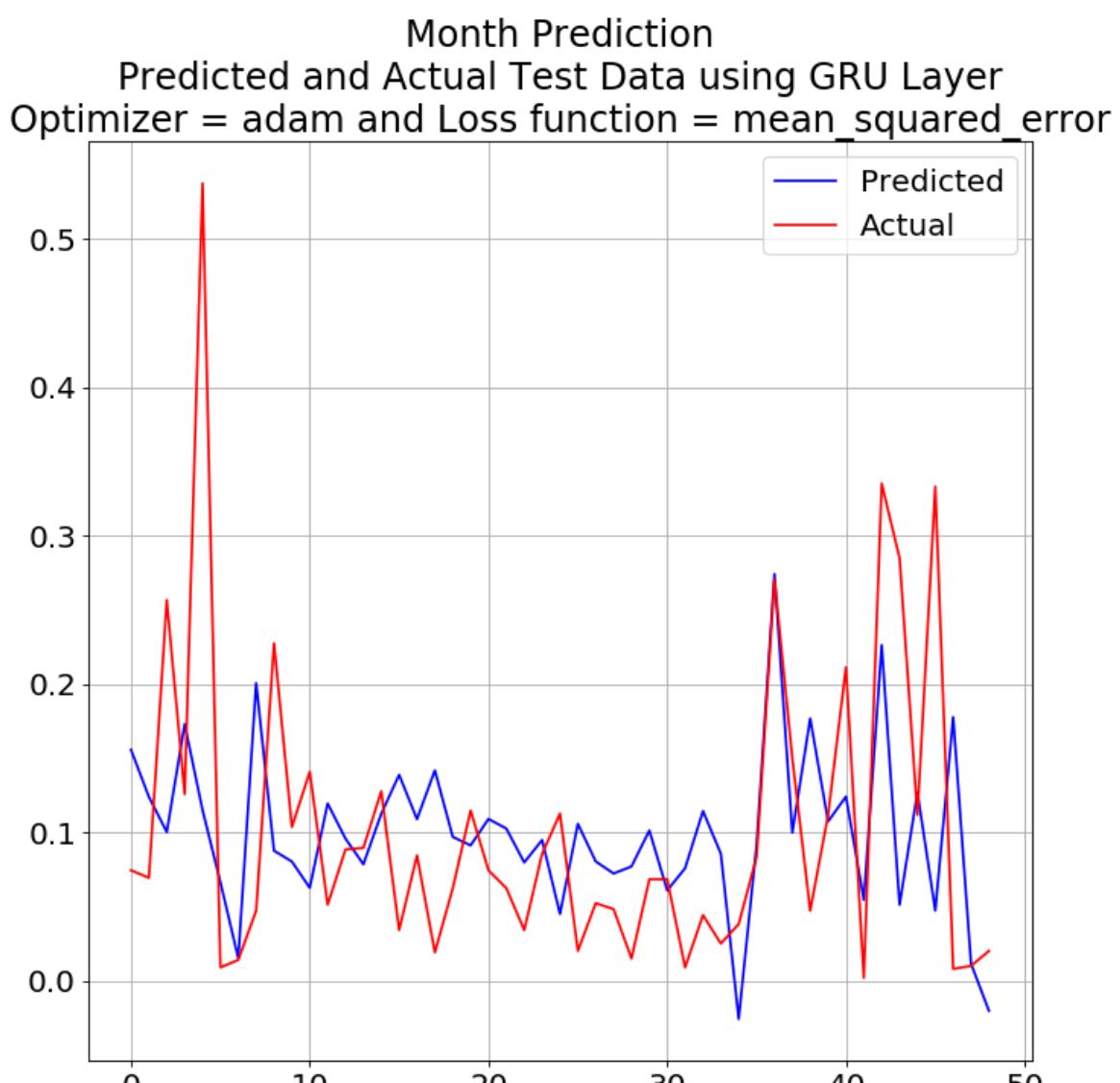


شکل 4-8: نمودار loss داده های train پیش بینی ماهانه

نمودار پیش بینی داده های train و test به ترتیب در 4-9 و 4-10 رسم شده است.



شکل ۴-۹: نمودار پیش بینی ماهانه روی داده های train

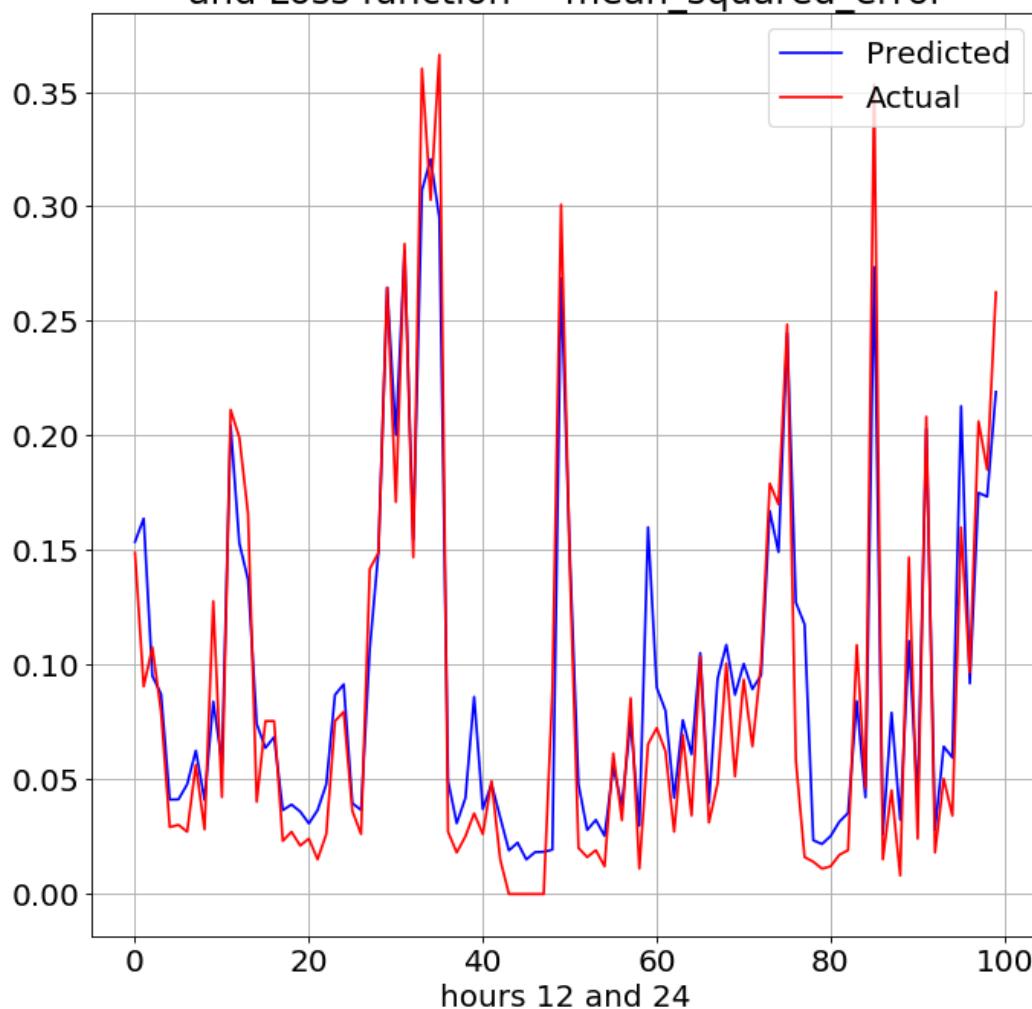


شکل ۱۰-۴: نمودار پیش بینی داده های test

بخش 5) تأثیر لایه Dropout

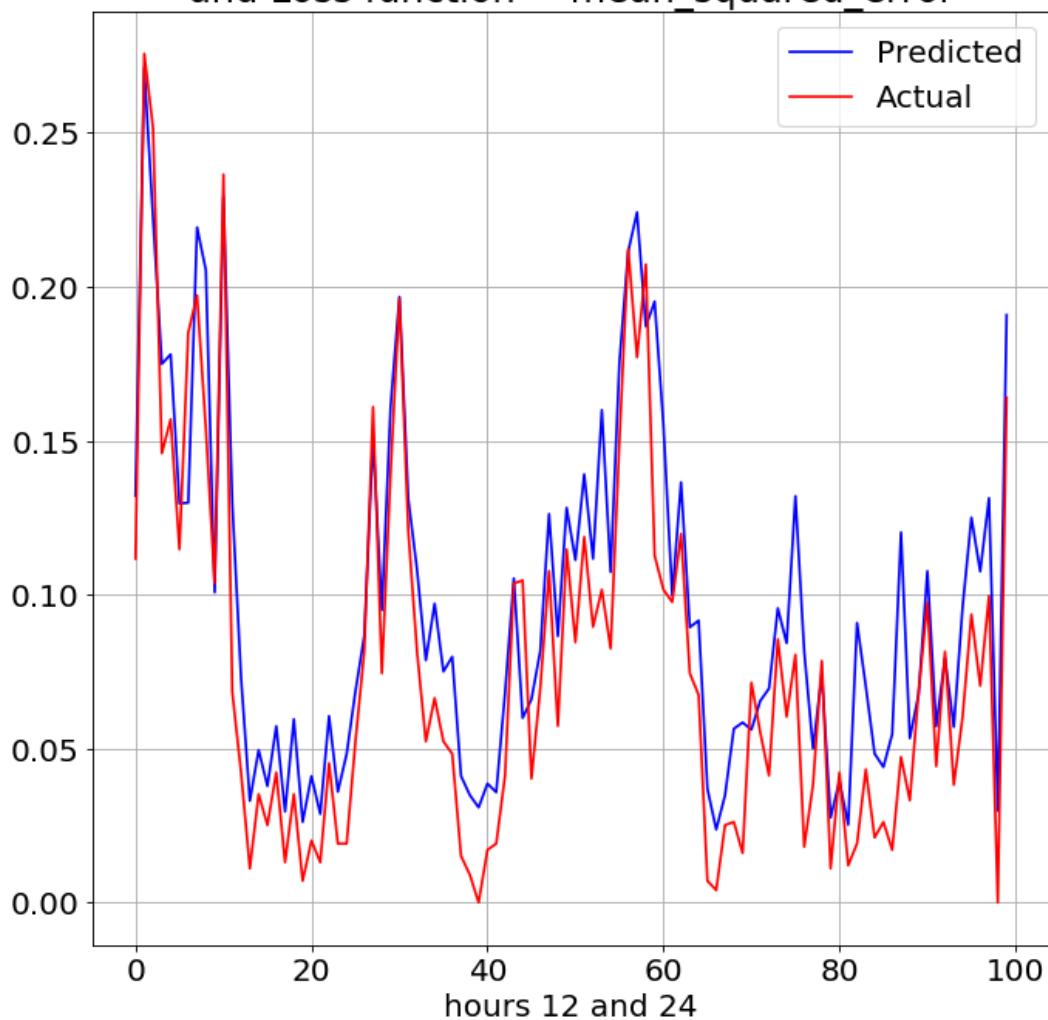
در این قسمت شبکه دلخواهی از قسمت های قبل را در نظر گرفته و تأثیر اضافه کردن لایه Dropout بر روی نمودارها و نتایج را بررسی می کنیم. adam optimizer را loss function را MSE و از لایه GRU به عنوان لایه Recurrent استفاده می کنیم. ابتدا شبکه را بدون لایه dropout آموزش می دهیم. نمودار های پیش بینی داده های train و test به ترتیب در شکل های 5-1 و 5-2 قابل مشاهده هستند.

Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 5-1: نمودار پیش بینی داده های train بدون استفاده از لایه dropout

Predicted and Actual Test Data without using DropOut Layers
 using GRU Layer, Optimizer = adam
 and Loss function = mean_squared_error



شکل 5-2: نمودار پیش بینی داده های test بدون لایه dropout

با اعمال این شبکه روی داده های test مقدار MSE به صورت شکل 5-3 خواهد شد:

```
1 model.evaluate(testX, testY)
```

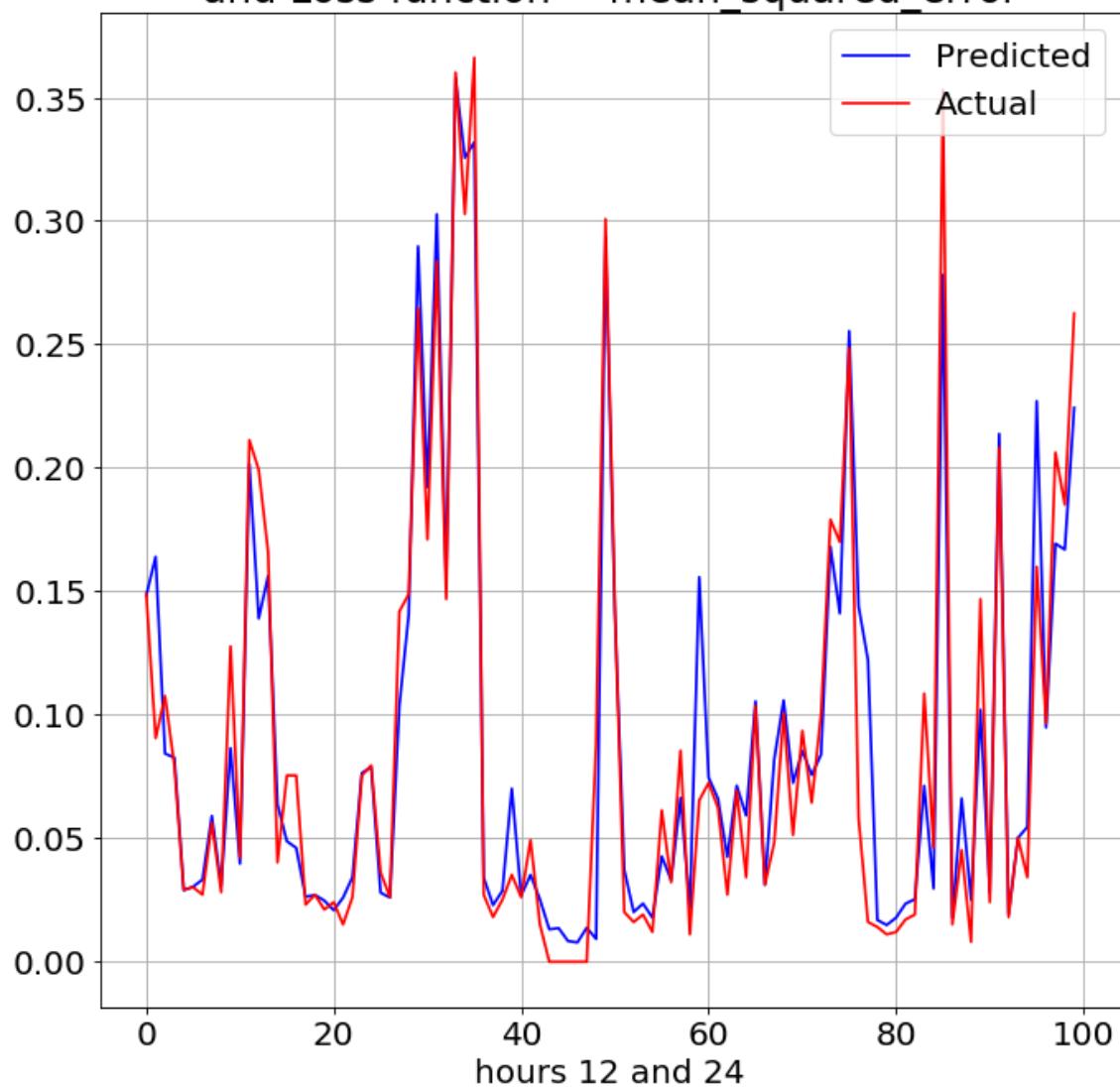
```
4368/4368 [=====] - 0s 58us/sample - loss: 7.3781e-04
```

```
0.0007378144143660973
```

شکل 5-3: مقدار MSE روی داده های تست

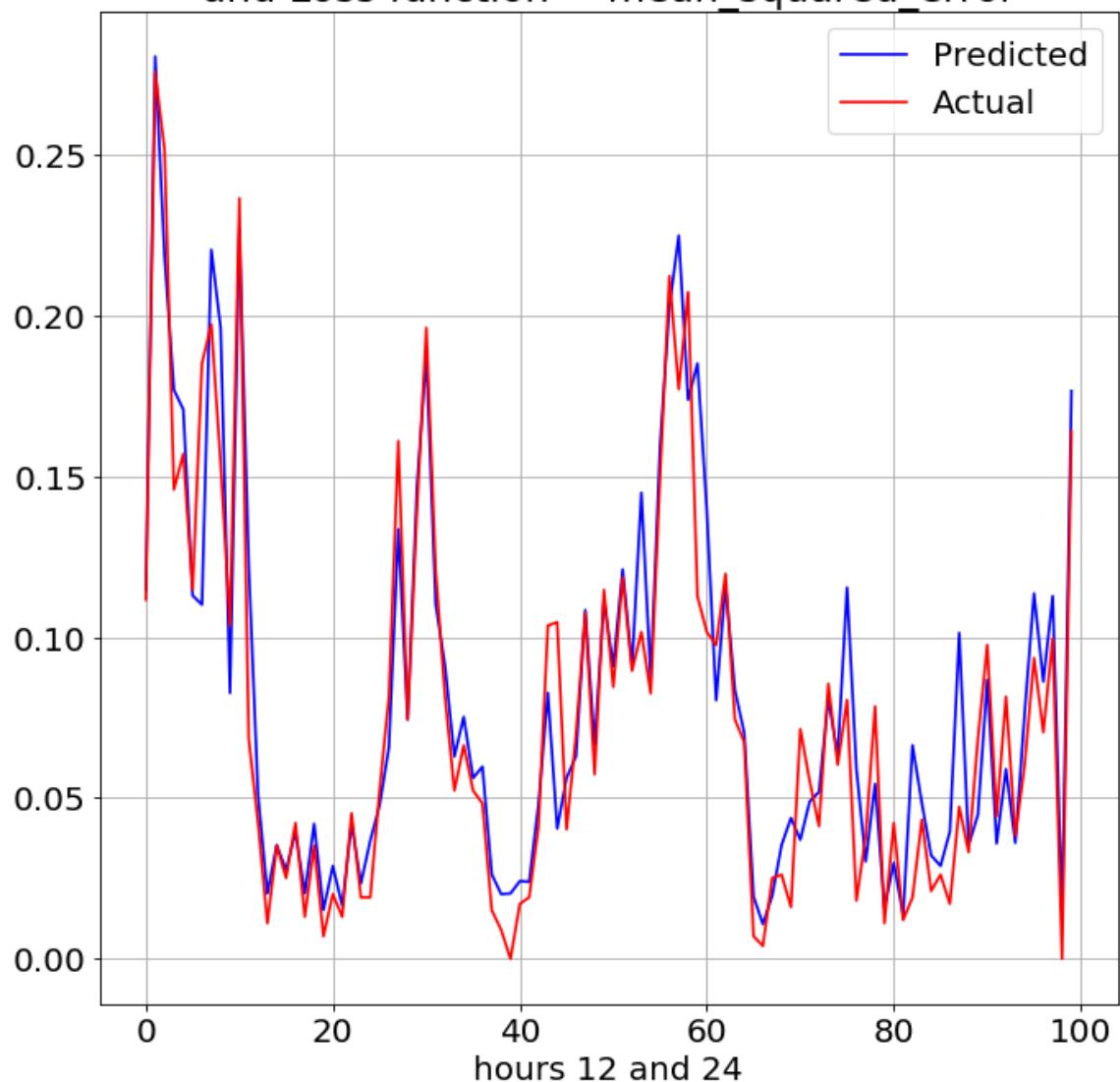
حال لایه Dropout را بعد از لایه recurrent و لایه dense با $Rate = 0.2$ قرار داده و دوباره شبکه را می کنیم. پس از آموزش شبکه نمودار پیش بینی داده های train و test به ترتیب به صورت شکل های 5-4 و 5-5 خواهد بود.

Predicted and Actual Train Data using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل ۴-۵. نمودار پیش بینی داده های train با استفاده از لایه dropout

Predicted and Actual Test Data using DropOut Layers
 using GRU Layer, Optimizer = adam
 and Loss function = mean_squared_error



شکل 5-5: پیش بینی داده های test در صورت استفاده از لایه dropout

با مقایسه شکل های 5-1 و 5-2 با 5-4 و 5-5 مشاهده می شود پیش بینی ها در صورت استفاده از لایه dropout بهتر شده است.

با اعمال این شبکه روی داده های test مقدار MSE به صورت شکل 6-5 خواهد شد.

```

1 model.evaluate(testX, testY)

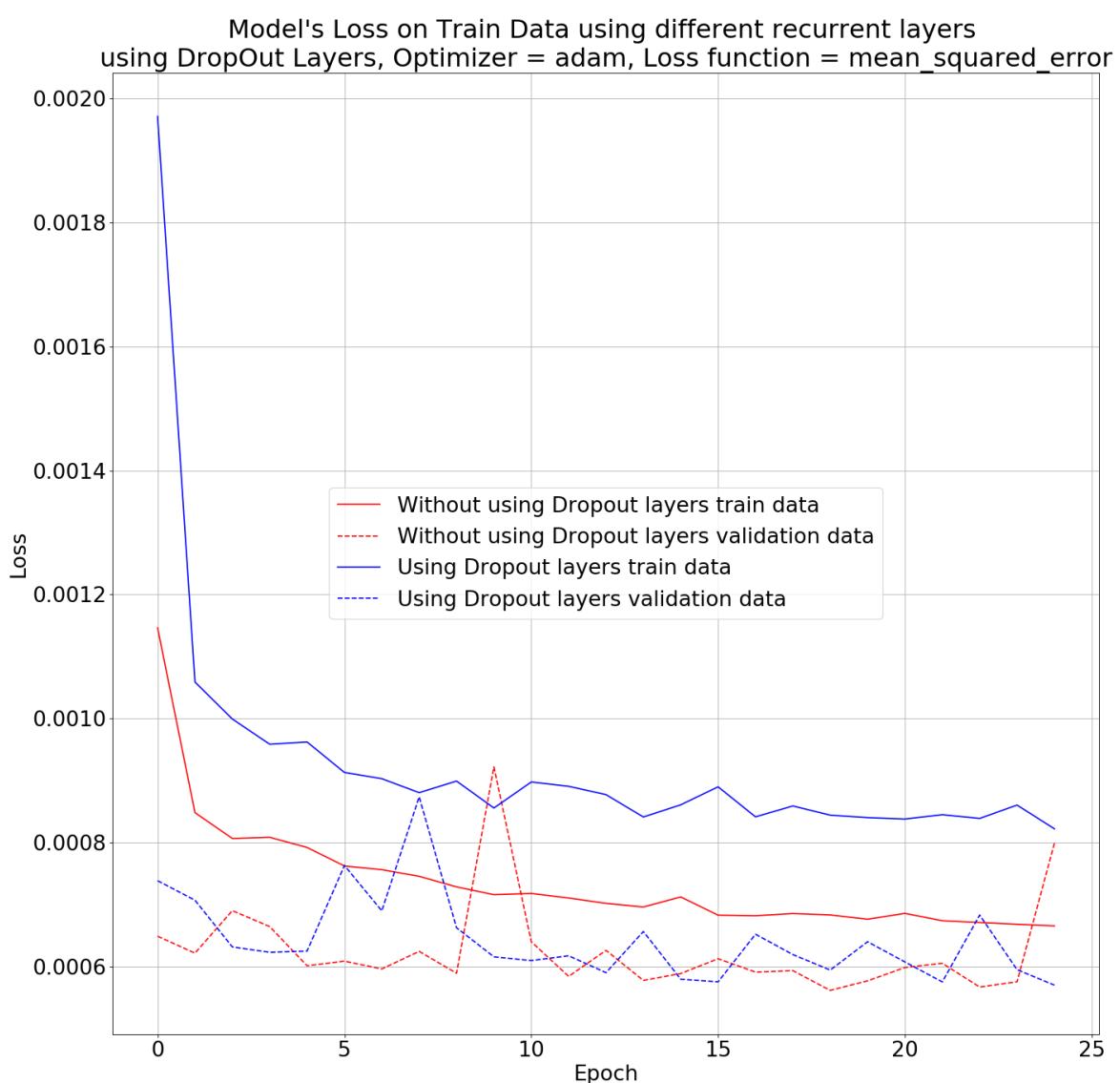
4368/4368 [=====] - 0s 61us/sample - loss: 5.0659e-04
0.0005065926272749786

```

شکل 5-6: نتیجه شبکه روی داده های test

با مقایسه شکل های 5-3 و 5-6 مشاهده می شود مقدار loss روی داده های test در صورت استفاده از لایه های dropout کمتر است.

شبکه 7 نمودار loss دو شبکه را بر حسب epoch برای داده های train و validation نشان می دهد.



شکل 5-7: نمودار loss داده های train بر حسب epoch

در نهایت شکل 5-8 مدت زمان لازم برای آموزش شبکه در صورت استفاده یا عدم استفاده از لایه های dropout را نشان می دهد.

```
1 print(f"Time taken without using dropout layer = {Time_taken_drp[0]} sec")
2 print(f"Time taken using dropout layer = {Time_taken_drp[1]} seconds")
```

```
Time taken without using dropout layer = 194.24439573287964 seconds
Time taken using dropout layer = 183.86869382858276 seconds
```

شكل 5-8: زمان آموزش شبکه

بخش 6) لایه fusion و موازی کردن سه شبکه به کمک لایه Average

بهترین شبکه بازگشتی در قسمتهای قبل شبکه طراحی شده در قسمت های 1 و 2 و 3 برای پیشビینی ساعت های 12 و 24 بود که در این قسمت از ساختار آن شبکه استفاده می کنیم. در اینجا سه شبکه مشابه را به صورت موازی پیاده سازی می کنیم در هر یک از این سه شبکه از لایه GRU با 50 یونیت به عنوان لایه recurrent و یک لایه dense با 100 نورون و یک لایه dense با یک نورون به عنوان لایه خروجی استفاده می کنیم و در آخر خروجی این سه شبکه را با استفاده از لایه Average کتابخانه keras به هم وصل می کنیم. برای ورودی این سه شبکه از سری زمانی های با طول پنجره های مختلف مثلا 5، 12 و 20 استفاده می کنیم. طبق شکل 6-1 ابتدا ورودی های train و test سه شبکه و خروجی شبکه کلی را آماده می کنیم.

```
1 train_percentage = 0.9
2 Num_train = int(Num_data*train_percentage)
3 train = dataset[:Num_train, :]
4 test = dataset[Num_train:, :]
5
6 # Choosing window size of three neural networks window1 < window2 < window3
7 window1 = 5
8 window2 = 12
9 window3 = 20
10 train1X, _ = createDataset(train, window=window1, smpl=1, stride=1, predicting_feature=0)
11 train2X, _ = createDataset(train, window=window2, smpl=1, stride=1, predicting_feature=0)
12 train3X, trainY = createDataset(train, window=window3, smpl=1, stride=1, predicting_feature=0)
13
14 test1X, _ = createDataset(test, window=window1, smpl=1, stride=1, predicting_feature=0)
15 test2X, _ = createDataset(test, window=window2, smpl=1, stride=1, predicting_feature=0)
16 test3X, testY = createDataset(test, window=window3, smpl=1, stride=1, predicting_feature=0)
17
18 # Synchronizing train datasets
19 train1X = train1X[(window3 - window1):]
20 print(f"train1X shape = {train1X.shape}")
21
22 train2X = train2X[(window3 - window2):]
23 print(f"train2X shape = {train2X.shape}")
24
25 print(f"train3X shape = {train3X.shape}")
26 print(f"trainY shape = {trainY.shape}\n")
27
28 # Synchronizing test datasets
29 test1X = test1X[(window3 - window1):]
30 print(f"test1X shape = {test1X.shape}")
31
32 test2X = test2X[(window3 - window2):]
33 print(f"test2X shape = {test2X.shape}")
34
35 print(f"test3X shape = {test3X.shape}")
36 print(f"testY shape = {testY.shape}\n")
```

شکل 6-1: آماده کردن دیتاست های ورودی سه شبکه و خروجی

ابعاد ورودی ها و خروجی در شکل 6-2 نشان داده شده است.

```

train1X shape = (39398, 5, 8)
train2X shape = (39398, 12, 8)
train3X shape = (39398, 20, 8)
trainY shape = (39398,)

test1X shape = (4359, 5, 8)
test2X shape = (4359, 12, 8)
test3X shape = (4359, 20, 8)
testY shape = (4359,)

```

شکل ۶-۲: ابعاد داده های **train** و **test**

شکل های ۶-۳ ، ۶-۴ و ۶-۵ به ترتیب ساختار سه شبکه ای که موازی کرده ایم را نشان می دهد.

model 1:

```

# Recurrent_Layer: (0 ==> RNN) (1 ==> LSTM) (2 ==> GRU)
Recurrent_Layer = 0
# Number of units:
units = 50
# fully connected layer
N_neurons = 100
# Initializing model as Sequential
input1 = Input(shape=(window1, Num_features))
# Adding Recurrent layer
if Recurrent_Layer == 0:
    rec1 = SimpleRNN(units)(input1)
elif Recurrent_Layer == 1:
    rec1 = LSTM(units)(input1)
elif Recurrent_Layer == 2:
    rec1 = GRU(units)(input1)
# Adding fully connected layer
dns1 = Dense(N_neurons, activation = "relu")(rec1)
# Adding output layer
out1 = Dense(1)(dns1)

```

شکل ۶-۳: ساختار شبکه اول

model 2:

```
1 # Recurrent_Layer: (0 ==> RNN) (1 ==> LSTM) (2 ==> GRU)
2 Recurrent_Layer = 0
3
4 # Number of units:
5 units = 50
6
7 # fully connected Layer
8 N_neurons = 100
9
10 # Initializing model as Sequential
11 input2 = Input(shape=(window2, Num_features))
12
13 # Adding Recurrent layer
14 if Recurrent_Layer == 0:
15     rec2 = SimpleRNN(units)(input2)
16 elif Recurrent_Layer == 1:
17     rec2 = LSTM(units)(input2)
18 elif Recurrent_Layer == 2:
19     rec2 = GRU(units)(input2)
20
21 # Adding fully connected layer
22 dns2 = Dense(N_neurons, activation = "relu")(rec2)
23
24 # Adding output layer
25 out2 = Dense(1)(dns2)
```

شكل 4: ساختار شبکه دوم

model 3:

```
1 # Recurrent_Layer: (0 ==> RNN) (1 ==> LSTM) (2 ==> GRU)
2 Recurrent_Layer = 0
3
4 # Number of units:
5 units = 50
6
7 # fully connected layer
8 N_neurons = 100
9
10 # Initializing model as Sequential
11 input3 = Input(shape=(window3, Num_features))
12
13 # Adding Recurrent layer
14 if Recurrent_Layer == 0:
15     rec3 = SimpleRNN(units)(input3)
16 elif Recurrent_Layer == 1:
17     rec3 = LSTM(units)(input3)
18 elif Recurrent_Layer == 2:
19     rec3 = GRU(units)(input3)
20
21 # Adding fully connected layer
22 dns3 = Dense(N_neurons, activation = "relu")(rec3)
23
24 # Adding output layer
25 out3 = Dense(1)(dns3)
```

شكل 5: ساختار شبکه سوم

حال به کمک لایه Average خروجی این سه شبکه را مطابق شکل 6-6 به هم وصل می کنیم.

Parallel and average three networks:

```
▶ 1 avg = Average()([out1, out2, out3])
  2 model = tf.keras.models.Model(inputs=[input1, input2, input3], outputs=avg)
```

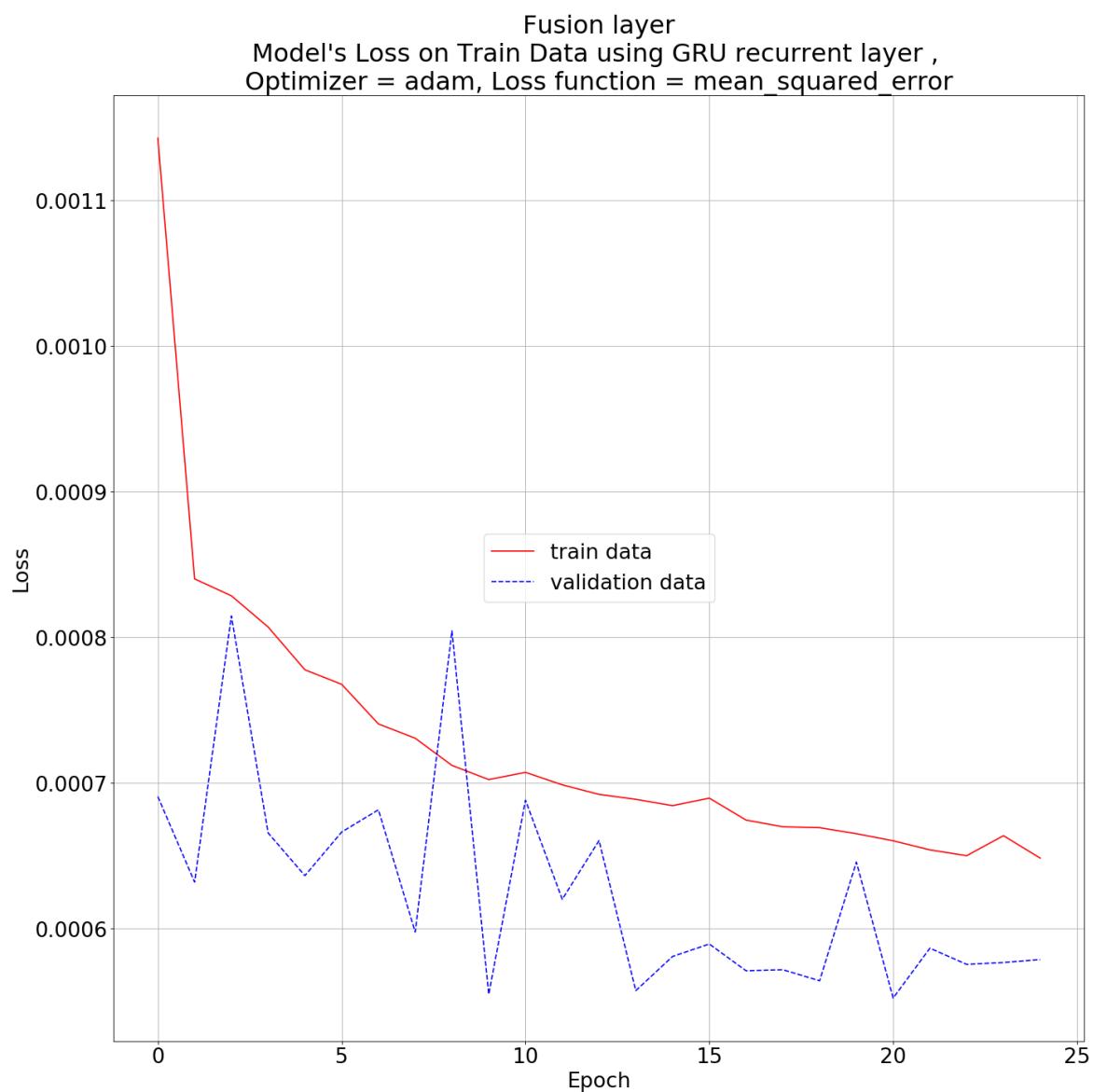
شکل 6-6: اتصال سه شبکه موازی

در نهایت شبکه بدست آمده را که از موازی کردن سه شبکه بدست آمده است را مطابق شکل 6-7 آموزش می دهیم.

```
1 # We use history to plot loss and accuracy
2 history = History()
3
4 # Number of epochs and batchSize
5 epochs = 25
6 batchSize = 32
7
8 # chooseLoss: (0 ==> MSE) (1 ==> MAE)
9 chooseLoss = 0
10
11 # chooseOptimizer: (0 ==> adam) (1 ==> adagard)
12 chooseOptimizer = 0
13
14 # Choosing Loss function
15 if chooseLoss == 0:
16     Loss = 'mean_squared_error'
17 elif chooseLoss == 1:
18     Loss = 'mean_squared_error'
19
20 # Choosing Optimizer
21 if chooseOptimizer == 0:
22     Optimizer = "adam"
23 elif chooseOptimizer == 1:
24     Optimizer = "adagard"
25
26 model.compile(loss = Loss, optimizer = Optimizer, metrics = ['mean_squared_error', 'mean_absolute_error'])
27
28 model.fit([train1X, train2X, train3X], y=trainY, epochs = epochs, callbacks = [history],
29           batch_size = batchSize, validation_split = 0.1)
30
31 Results_fusion[Recurrent_Layer] = history.history
32 test_evaluation_fusion[Recurrent_Layer] = model.evaluate([test1X, test2X, test3X], testY)
```

شکل 6-7: نحوه آموزش شبکه

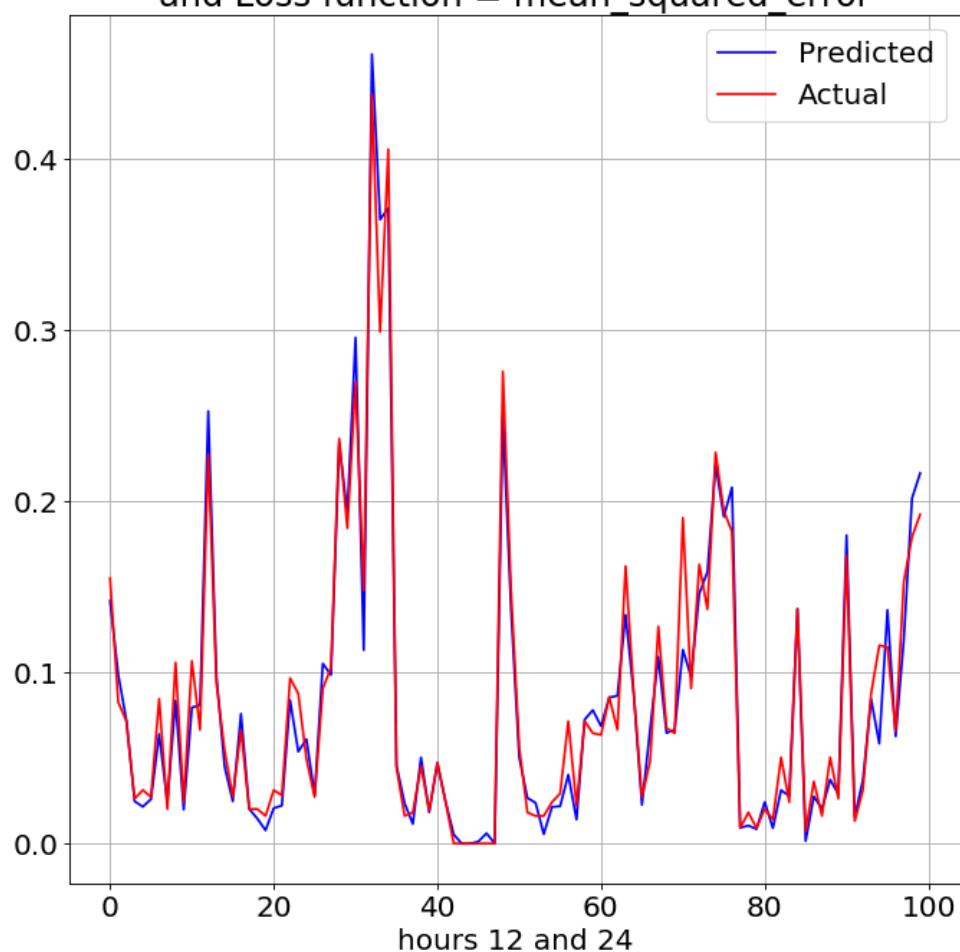
پس از آموزش شبکه نمودار loss بر حسب epoch برای داده های train به صورت شکل 6-8 خواهد بود.



شکل 6-8: نمودار loss بر حسب epoch

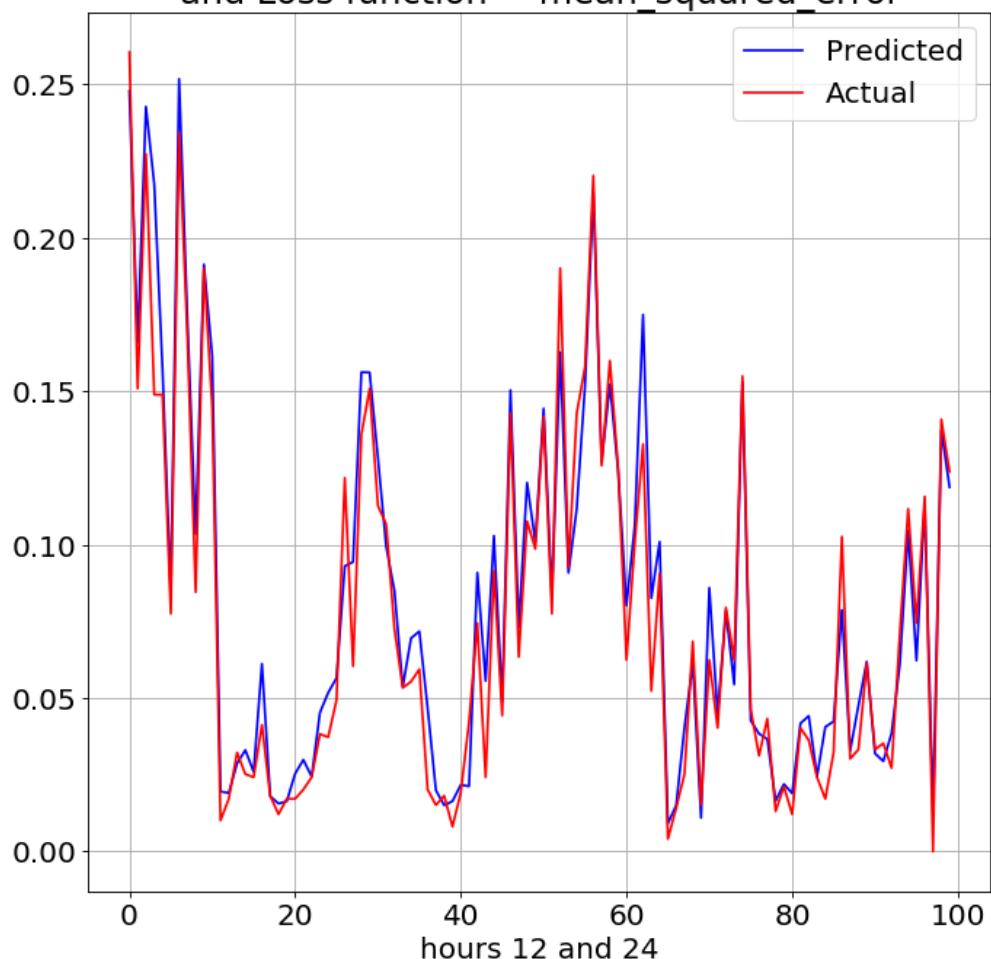
همچنین نمودارهای پیش بینی داده های train و test به ترتیب در شکل های 6-9 و 6-10 قابل مشاهده می باشد:

Predicted and Actual Train Data using fusion layer (Average layer)
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 6.9: پیش بینی داده های train

Predicted and Actual Test Data using fusion layer (Average layer)
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 6-10: پیش بینی داده های test توسط شبکه

و در نهایت مقدار خطای MSE شبکه روی داده های test به صورت شکل 6-11 خواهد بود.

Test evaluations:

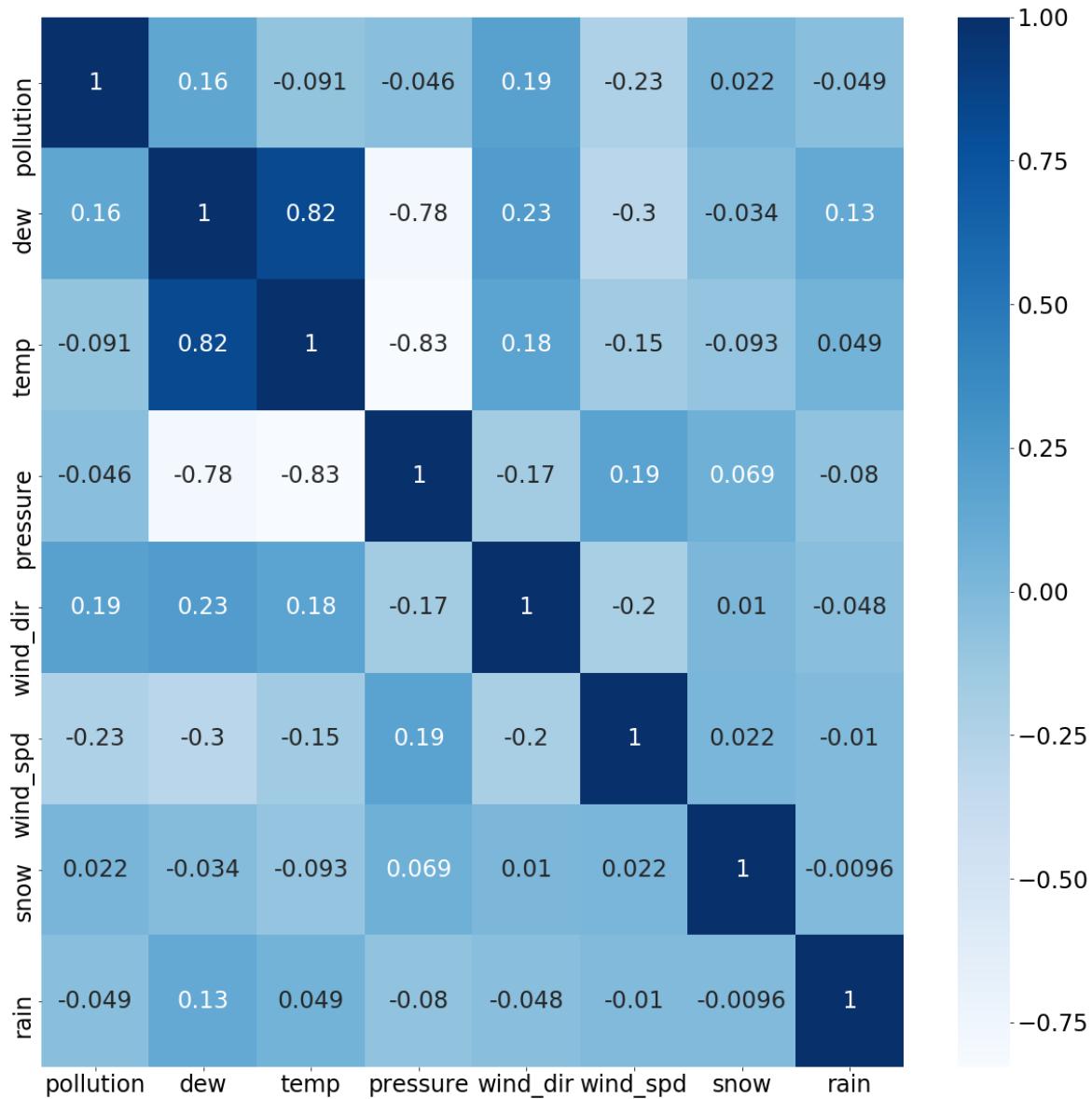
```
1 print(f"GRU layer:")
2 print(f"MSE = {test_evaluation_fusion[2][0]}, MAE = {test_evaluation_fus
3
GRU layer:
MSE = 0.00047067118286435705, MAE = 0.00047067113337107003
```

شکل 6-11: خطای MSE روی داده های تست

با مقایسه شکل 2-8 و 11-6 نتیجه می گیریم عملکرد شبکه حاصل کمی بهتر شده است.

بخش 7 و 8) پیش بینی با استفاده از سه ستون

برای انتخاب دو ستون که بیشترین تأثیر را در پیش بینی آلودگی هوا داشته باشند ماتریس همبستگی فیچر ها را محاسبه و رسم می کنیم که در شکل 1-8 قابل مشاهده است:

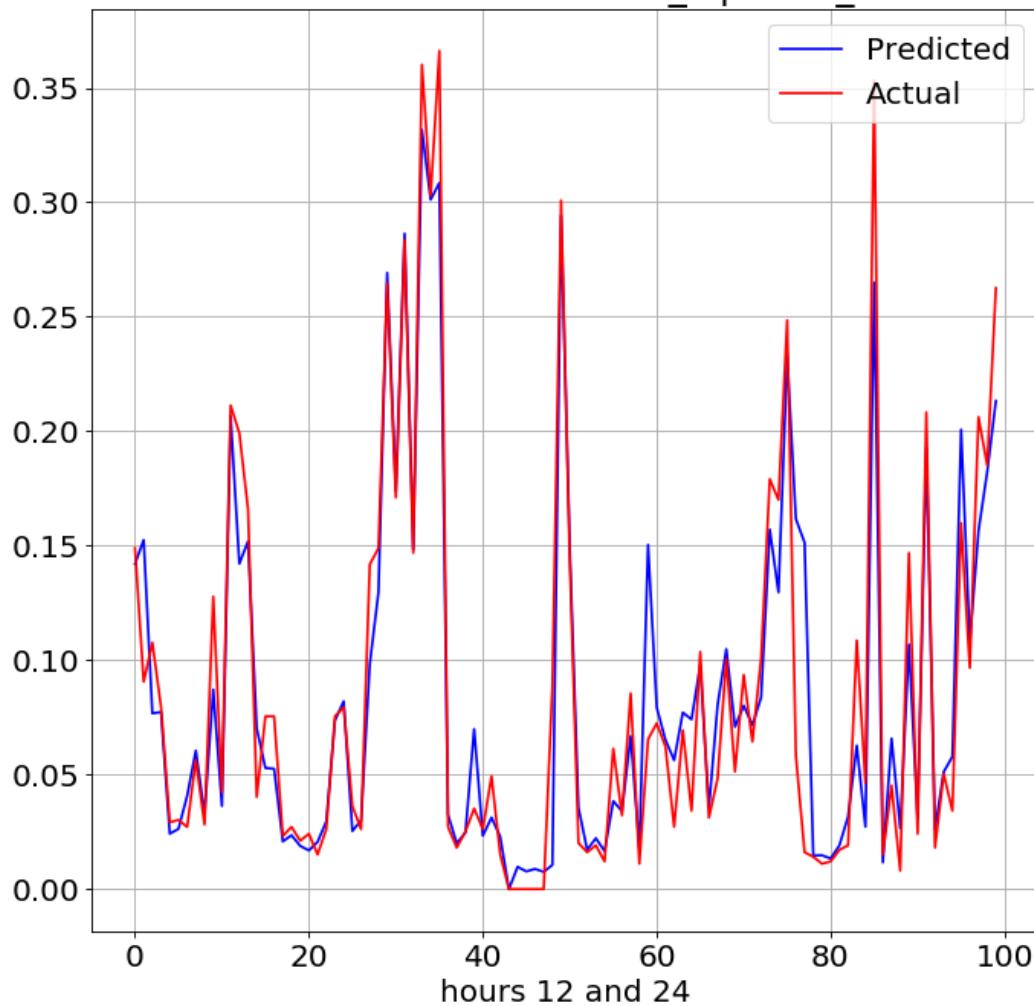


شکل 1-8: ماتریس همبستگی فیچرها

با توجه به ستون مربوط به آلودگی دو ویژگی که بیشترین correlation را از لحاظ قدر مطلق دارند و wind_spd و wind_dir هستند. این دو ویژگی را به همراه ویژگی pollution در نظر گرفته و شبکه را سه بار با لایه های recurrent مختلف آموزش می دهیم. adam وتابع loss را

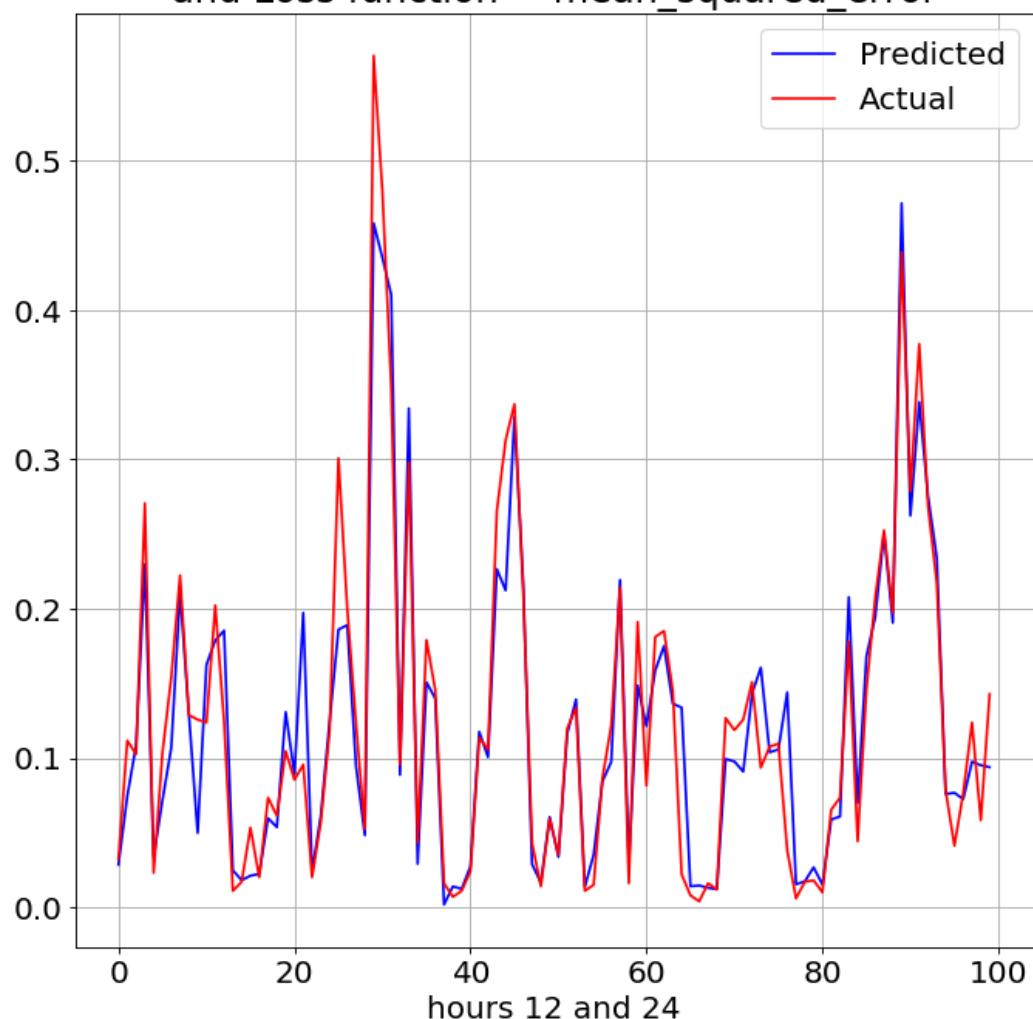
انتخاب می کنیم. نمودارهای پیش بینی داده های train و tset برای لایه های GRU و LSTM ، RNN پس از آموزش شبکه به ترتیب در شکل های 8-7 و 8-2 قابل مشاهده می باشند.

Using 3 features: ['pollution', 'wind_dir', 'wind_spd']
Predicted and Actual Train Data without using DropOut Layers
using RNN Layer, Optimizer = adam
and Loss function = mean_squared_error



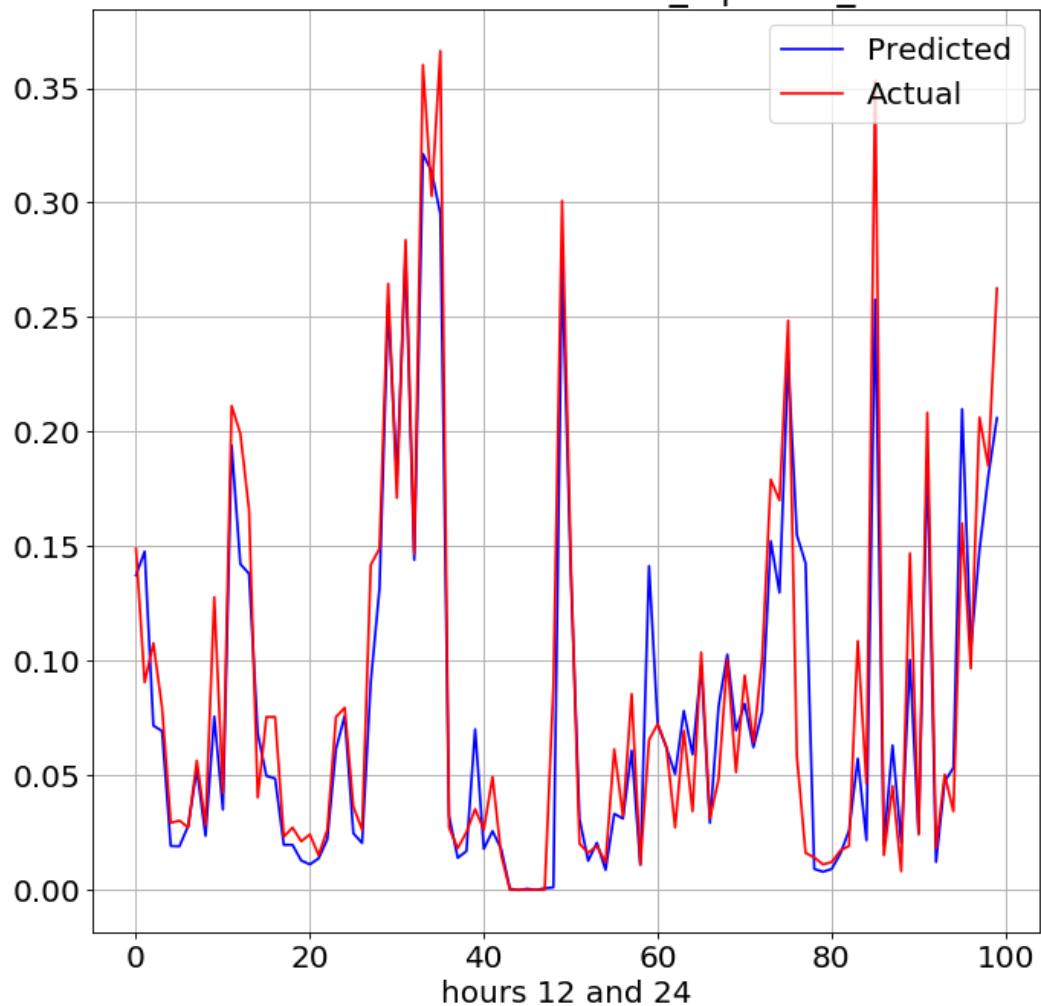
8-2 شکل

Using 3 features: ['pollution', 'wind_dir', 'wind_spd']
Predicted and Actual Test Data without using DropOut Layers
using RNN Layer, Optimizer = adam
and Loss function = mean_squared_error



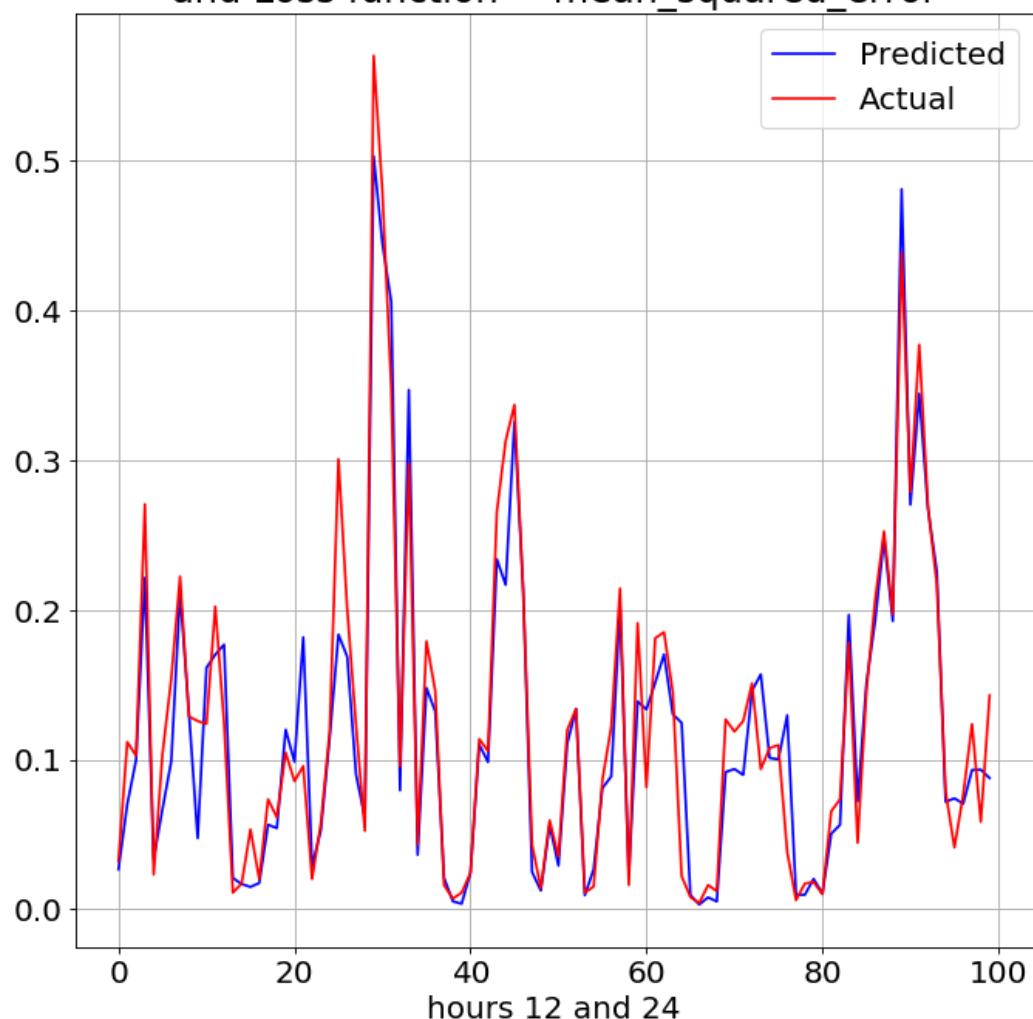
شكل 8-3

Using 3 features: ['pollution', 'wind_dir', 'wind_spd']
Predicted and Actual Train Data without using DropOut Layers
using LSTM Layer, Optimizer = adam
and Loss function = mean_squared_error



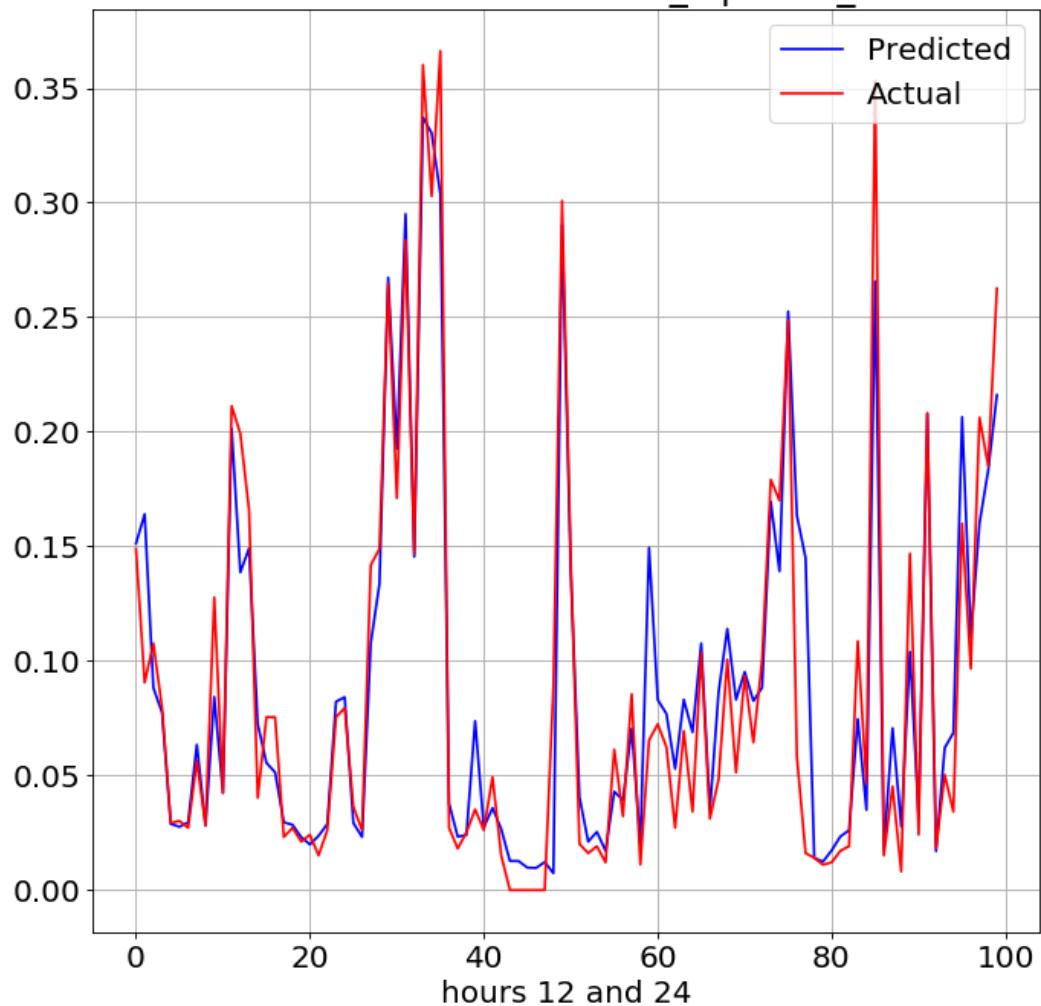
شكل 8.4

Using 3 features: ['pollution', 'wind_dir', 'wind_spd']
Predicted and Actual Test Data without using DropOut Layers
using LSTM Layer, Optimizer = adam
and Loss function = mean_squared_error



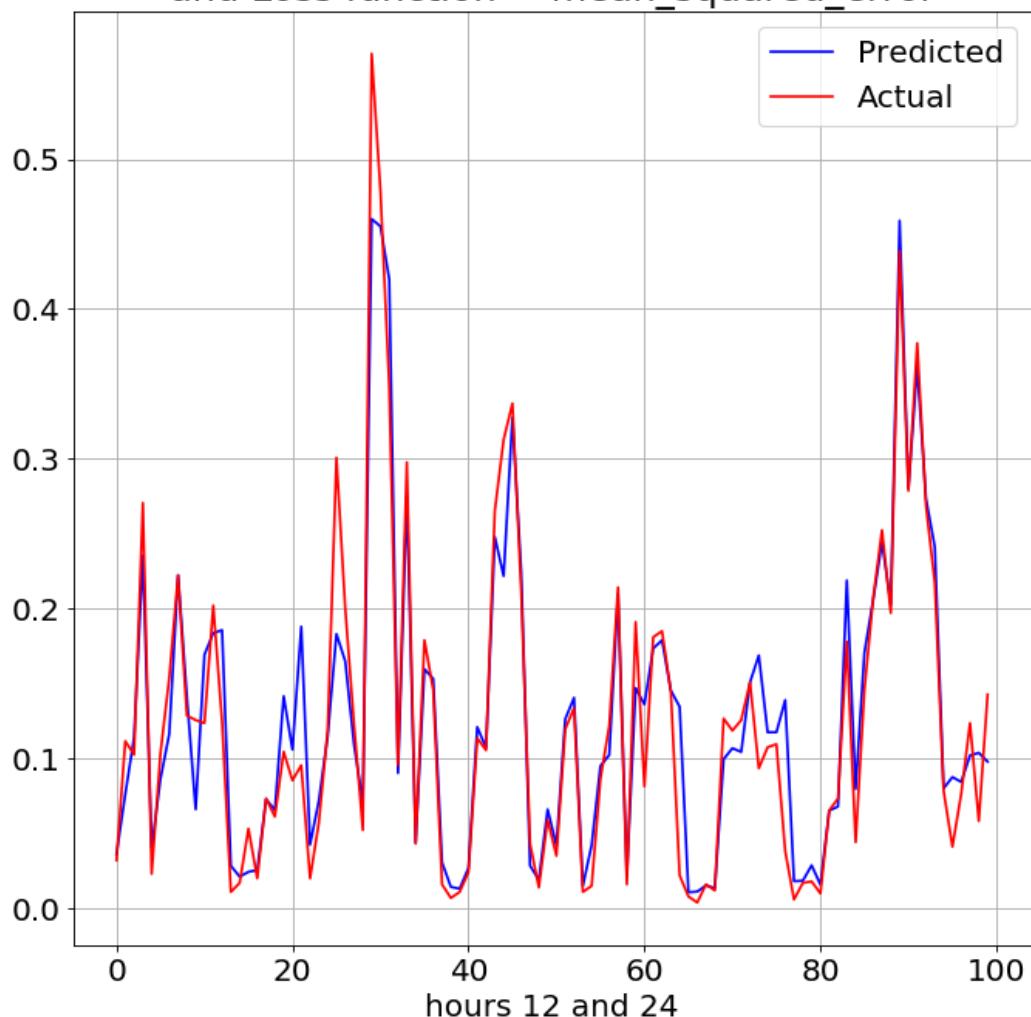
شكل 8-5

Using 3 features: ['pollution', 'wind_dir', 'wind_spd']
Predicted and Actual Train Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



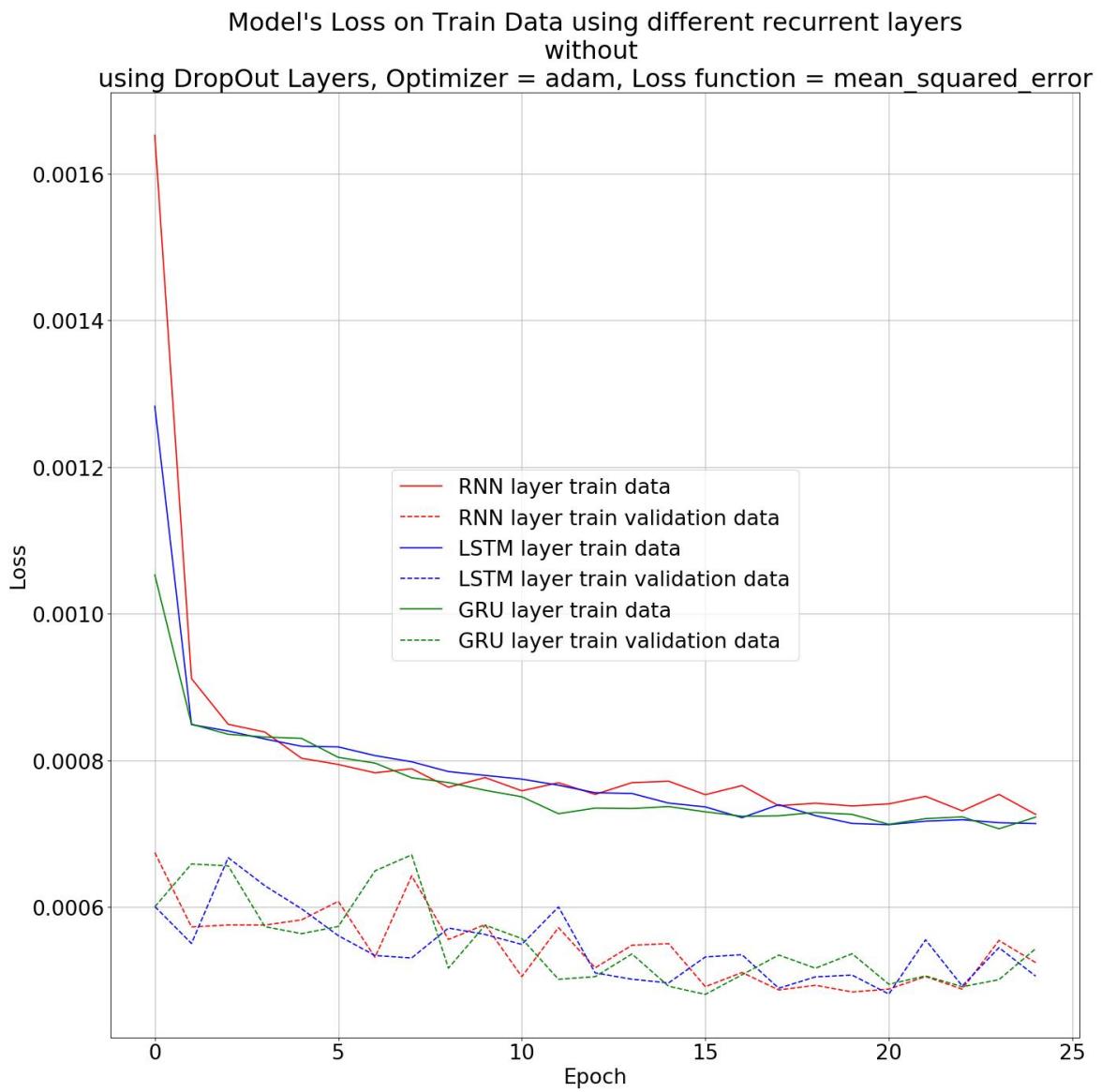
شكل 8.6

Using 3 features: ['pollution', 'wind_dir', 'wind_spd']
Predicted and Actual Test Data without using DropOut Layers
using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



8-7 شکل

نمودار loss سه شبکه در شکل 8-8 رسم شده است:



شکل 8-8: نمودار loss سه شبکه

همچنین نتیج سه شبکه روی داده های test و زمان آموزش شبکه در شکل 9-8 نشان داده شده است.

```
RNN layer:  
Time taken = 81.46726584434509 seconds  
Test evaluation:  
MSE=0.0005893364506599387
```

```
LSTM layer:  
Time taken = 141.17907571792603 seconds  
Test evaluation:  
MSE=0.0005763707063721221
```

```
GRU layer:  
Time taken = 458.7524981498718 seconds  
Test evaluation:  
MSE=0.0006498267014349335
```

شکل 8-9: نتایج روی داده های test

قسمت دوم) نقصان دادگان (بخش ۱)

در ابتدا به صورت رندوم طبق خواسته سوال 20% از داده های هر یک از هشت ویژگی دیتابست را به صورت شکل ۱-۱ حذف می کنیم و به جای آنها nan می گذاریم. لازم به ذکر است داده های حذف شده در یک آرایه نگه میداریم تا بعدا در محاسبه MSE از آنها استفاده کنیم.

Missing Values:

```
dataset_missing = np.copy(dataset).astype('float64')
remove_percentage = 0.2
N_delete = int(remove_percentage * Num_data)
nan_vec = np.NaN * np.zeros(N_delete)
deleted_index = np.zeros((N_delete, Num_features))
actual_data = np.zeros((N_delete, Num_features))

for i in range(Num_features):
    index = np.random.randint(Num_data, size = N_delete)
    deleted_index[:, i] = index
    actual_data[:, i] = dataset_missing[index, i]
    dataset_missing[index, i] = nan_vec
deleted_index = deleted_index.astype('int')
```

شكل ۱-۱: حذف 20% از دادگان هر ویژگی به صورت رندوم

بخش 2، 3، 4 و 5) نقصان دادگان (بخش ۱)

در این بخش ۵ استراتژی مختلف را توضیح می دهیم:

استراتژی ۱) پر کردن داده های از دست رفته با یک مقدار ثابت:

در این استراتژی تمام مقادیر nan دیتاست با یک مقدار ثابت مثلا صفر جایگزین می شود. بدیهی است این استراتژی بهینه نیست و خطای بسیار زیادی دارد. برای پیاده سازی این استراتژی از کلاس مطابق شکل ۲-۱ استفاده می کنیم.

Strategy 1 :Filling missing values with zero:

```
1 #Impute the values using scikit-learn SimpleImputer class
2
3 imp_zero = SimpleImputer(strategy='constant', fill_value = 0)
4 imp_zero.fit(dataset_missing)
5 imputed_dataset_strategy1 = imp_zero.transform(dataset_missing).astype('
6 strategy1_mse = np.zeros(Num_features)
7
8 for i in range(Num_features):
9     strategy1_mse[i] = MSE(actual_data[:, i], imputed_dataset_strategy1[
10
11 strategy1_mse
```

)]: array([0.01813271, 0.42444218, 0.30781448, 0.2505557 , 0.4035849 ,
0.00902167, 0.00083207, 0.00152136])

شکل ۱-۲: استراتژی پر کردن مقادیر nan با مقدار ثابت

همانطور که مشاهده می شود خطای MSE در تمام فیچرها عددی بالا است.

استراتژی ۲) پر کردن داده های از دست رفته با میانگین داده های باقی مانده:

در این استراتژی تمام مقادیر nan را با میانگین داده های باقیمانده فیچر مربوطه پر می کنیم. این استراتژی را نیز به کمک کلاس SimpleImputer به صورت شکل ۲-۲ پیاده سازی می کنیم.

Strategy 2 :Using mean of existing data:

```
1 #Impute the values using scikit-learn SimpleImputer class
2
3 imp_mean = SimpleImputer(strategy='mean')
4 imp_mean.fit(dataset_missing)
5 imputed_dataset_strategy2 = imp_mean.transform(dataset_missing).astype('
6 strategy2_mse = np.zeros(Num_features)
7
8 for i in range(Num_features):
9     strategy2_mse[i] = MSE(actual_data[:, i], imputed_dataset_strategy2[:
10
11 strategy2_mse
```

1]: array([0.00900863, 0.04583787, 0.03966971, 0.03478819, 0.09863287,
0.00738221, 0.00082803, 0.00149637])

شکل 2-2: استراتژی میانگین

همانطور که مشاهده می شود مقدار خطای فیچرها نسبت به استراتژی قبل بسیار کاهش یافته (حدود 0.1 برابر شده) ولی همچنان مقدار خطأ بالاست.

استراتژی 3) پر کردن داده های از دست رفته با میانه داده های باقی مانده:

این روش مشابه استراتژی میانگین است با این تفاوت که از میانه داده های باقی مانده برای پر کردن داده های از دست رفته استفاده می کنیم که در شکل 2-3 نشان داده شده است.

Strategy 3 :Using median of existing data:

```
1 #Impute the values using scikit-learn SimpleImputer class
2
3 imp_median = SimpleImputer(strategy='median')
4 imp_median.fit(dataset_missing)
5 imputed_dataset_strategy3 = imp_median.transform(dataset_missing).astype('
6 strategy3_mse = np.zeros(Num_features)
7
8 for i in range(Num_features):
9     strategy3_mse[i] = MSE(actual_data[:, i], imputed_dataset_strategy3[:
10
11 strategy3_mse
```

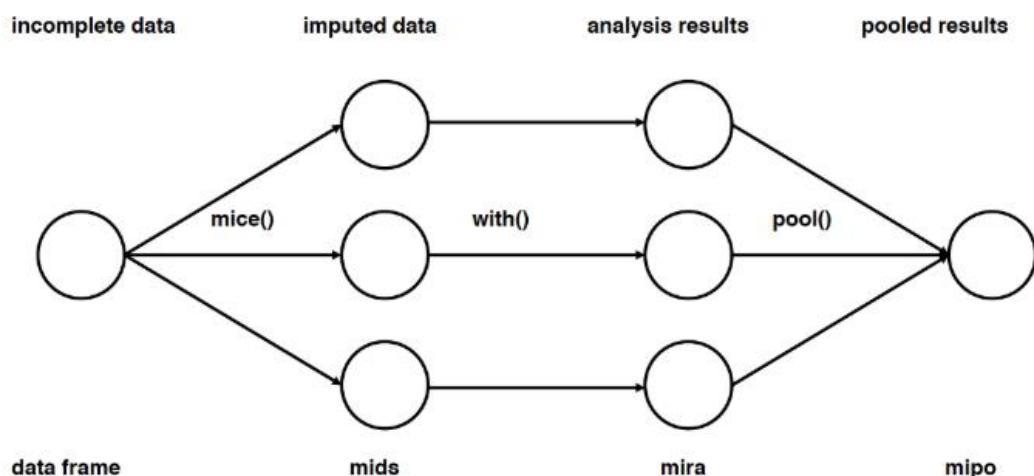
2]: array([0.00974259, 0.04584333, 0.04019874, 0.0348821 , 0.11172001,
0.00841142, 0.00083207, 0.00152136])

شکل 3-2: استراتژی میانه

همانطور که مشاهده می شود مقدار خطا به صورت ناچیز نسبت به میانگین بدتر شده است و تقریباً برابر است.

استراتژی 4 (4) Imputation Using Multivariate Imputation by Chained Equation (MICE)

به طور خلاصه این روش چندین بار عمل imputation را با الگوریتم خاصی انجام می دهد و بنابراین uncertainty بین داده های از دست رفته را بهتر محاسبه می کند. شماتیک این روش در شکل 4-2 نشان داده شده است:



شکل 2-4: روش mice

این استراتژی را مطابق شکل 5-2 پیاده می کنیم.

Strategy 4: Imputation Using Multivariate Imputation by Chained Equation (MICE):

```
1 # MICE strategy
2
3 imputed_dataset_strategy4 = mice(dataset_missing)
4 strategy4_mse = np.zeros(Num_features)
5
6 for i in range(Num_features):
7     strategy4_mse[i] = MSE(actual_data[:, i], imputed_dataset_strategy4[
8
9 strategy4_mse
10
11
12 ]: array([0.00759193, 0.01313771, 0.01043478, 0.01165715, 0.09051868,
13     0.00647876, 0.00081784, 0.00148234])
```

شکل ۲-۵: نحوه پیاده سازی استراتژی mice

همانطور که مشاهده می شود نتیجه بهتر شده است.

استراتژی ۵) درونیابی:

در این استراتژی که از استراتژی های قبل بسیار بهتر و بهینه تر است از روش درونیابی به کمک کتابخانه pandas و تابع interpolate با متد nearest استفاده می کنیم که در شکل ۶-۲ نشان داده شده است.

Strategy 5: Imputation Using Interpolation:

```
1 # Interpolation strategy
2
3 df_missed = pd.DataFrame(dataset_missing, columns = feature_names)
4 imputed_dataset_strategy5 = df_missed.interpolate(method = 'nearest').val
5
6 imputed_dataset_strategy5 = imputed_dataset_strategy5.astype('float64')
7 imp_mean = SimpleImputer(strategy='mean')
8 imp_mean.fit(imputed_dataset_strategy5)
9 imputed_dataset_strategy5 = imp_mean.transform(imputed_dataset_strategy5)
10 strategy5_mse = np.zeros(Num_features)
11
12 for i in range(Num_features):
13     strategy5_mse[i] = MSE(actual_data[:, i], imputed_dataset_strategy5[
14
15 strategy5_mse
16 #df_missed
17
18 ]: array([7.21251908e-04, 4.23242374e-04, 6.60784386e-04, 1.65911945e-04,
19     1.16071087e-01, 8.05163575e-04, 4.93319766e-05, 4.31919064e-04])
```

شکل ۶-۲: نحوه پیاده سازی استراتژی interpolation

همانطور که مشاهده می شود میزان خطا نسبت به استراتژی های قبل بسیار کاهش یافته و حدود 0.01 برابر از استراتژی میانگین (به جز ویژگی wind_dir) بهتر عمل کرده است. علت اینکه خطای categorical wind_dir کاهش محسوسی در هیچ یک از استراتژی ها پیدا نمی کند این است که این فیچر most frequent ، KNN ، Bayesian و ... که می باشد و باستی برای این ویژگی از روش هایی مانند categorical می باشد استفاده کرد.

در آخر تمام خطاها محاسبه شده در 5 استراتژی مذکور را برای مشاهده و مقایسه بهتر در یک دیتافریم ذخیره می کنیم و نشان می دهیم که در شکل 2-7 قابل مشاهده است.

| | pollution | dew | temp | pressure | wind_dir | wind_spd | snow | rain |
|----------------------|-----------|----------|----------|----------|----------|----------|----------|----------|
| zero fill | 0.018133 | 0.424442 | 0.307814 | 0.250556 | 0.403585 | 0.009022 | 0.000832 | 0.001521 |
| mean | 0.009009 | 0.045838 | 0.039670 | 0.034788 | 0.098633 | 0.007382 | 0.000828 | 0.001496 |
| median | 0.009743 | 0.045843 | 0.040199 | 0.034882 | 0.111720 | 0.008411 | 0.000832 | 0.001521 |
| mice | 0.007592 | 0.013138 | 0.010435 | 0.011657 | 0.090519 | 0.006479 | 0.000818 | 0.001482 |
| Interpolation | 0.000721 | 0.000423 | 0.000661 | 0.000166 | 0.116071 | 0.000805 | 0.000049 | 0.000432 |

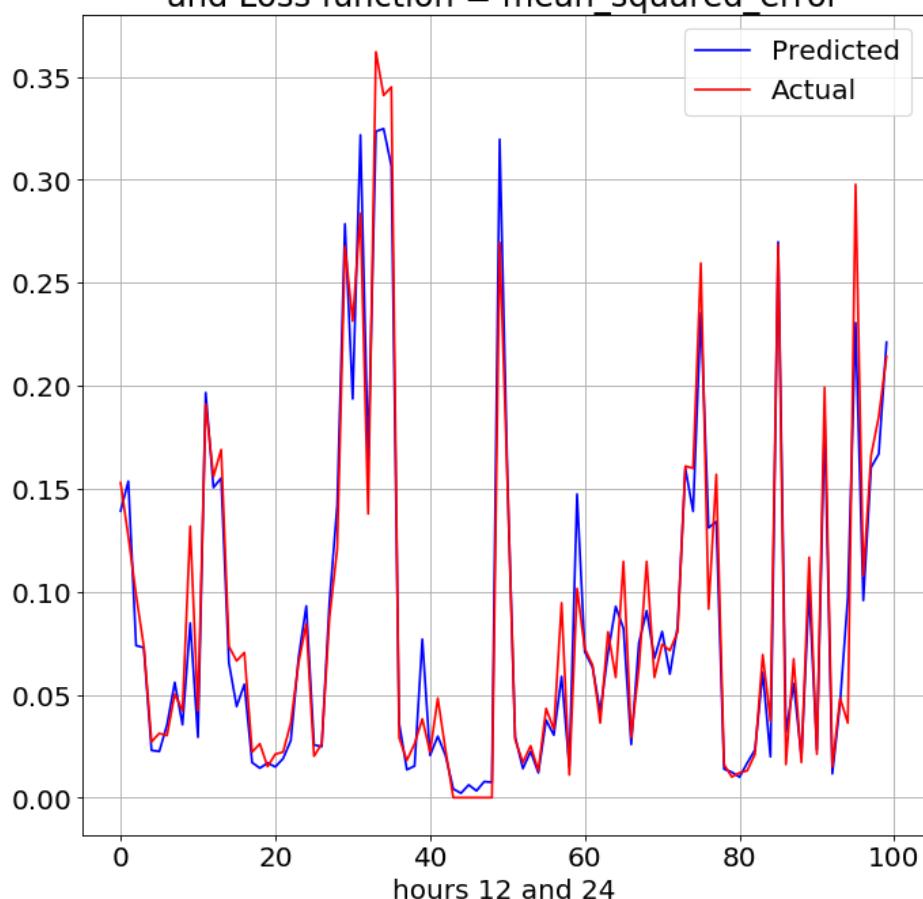
شکل 2-7: مقدار خطا با استراتژی های مختلف

همانطور که در شکل 2-7 مشاهده می شود استراتژی درونیابی یا interpolation از بقیه ها استراتژی ها بسیار بهتر است.

بخش 6) پیش بینی آلودگی پس از برطرف کردن missing values

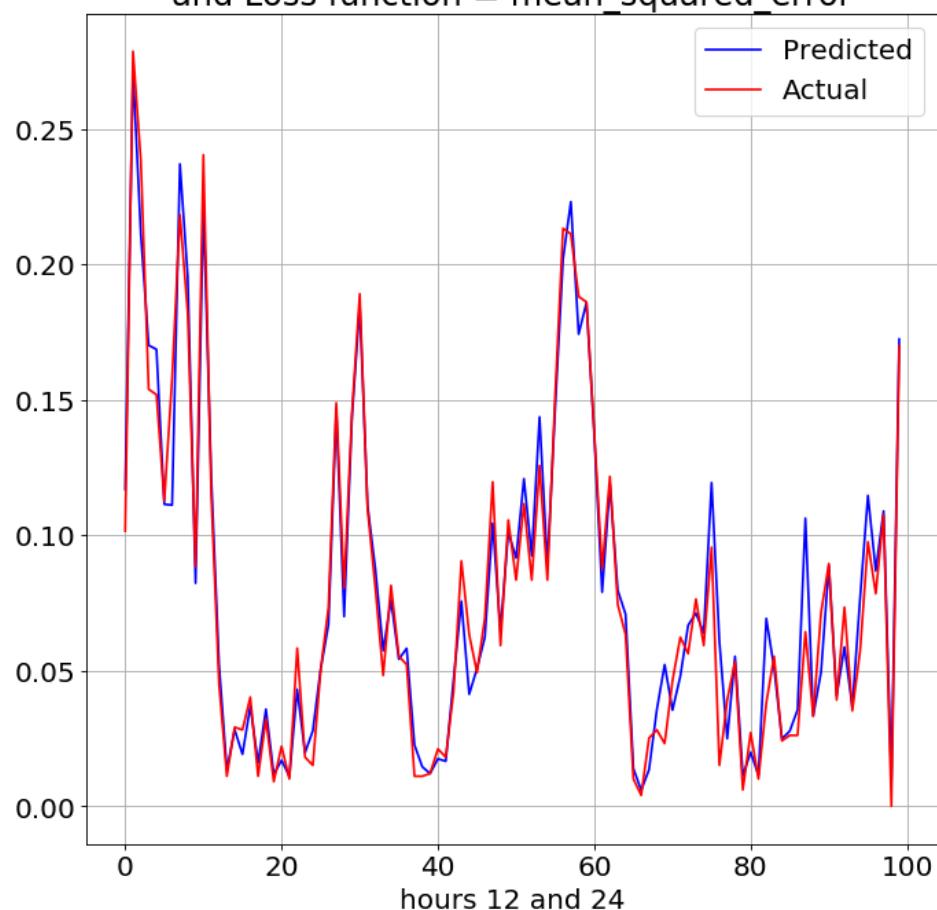
در این قسمت خروجی استراتژی درونیابی را که از بقیه استراتژی ها بهتر بود را به عنوان ورودی به شبکه می دهیم. ساختار شبکه را مشابه قسمت اول پروژه در نظر می گیریم (optimizer adam و تابع loss mse در نظر می گیریم) و برای دولایه GRU و LSTM شبکه را اجرا کرده و نتایج را رسم می کنیم. در ابتدا لایه LSTM را به عنوان لایه Recurrent در نظر می گیریم و شبکه را train می کنیم. نمودارهای پیش بینی داده های train و test به صورت شکل های 6-1 و 6-2 خواهد بود.

Predicted and Actual Train Data using LSTM Layer, Optimizer = adam and Loss function = mean_squared_error



.6-1 شکل

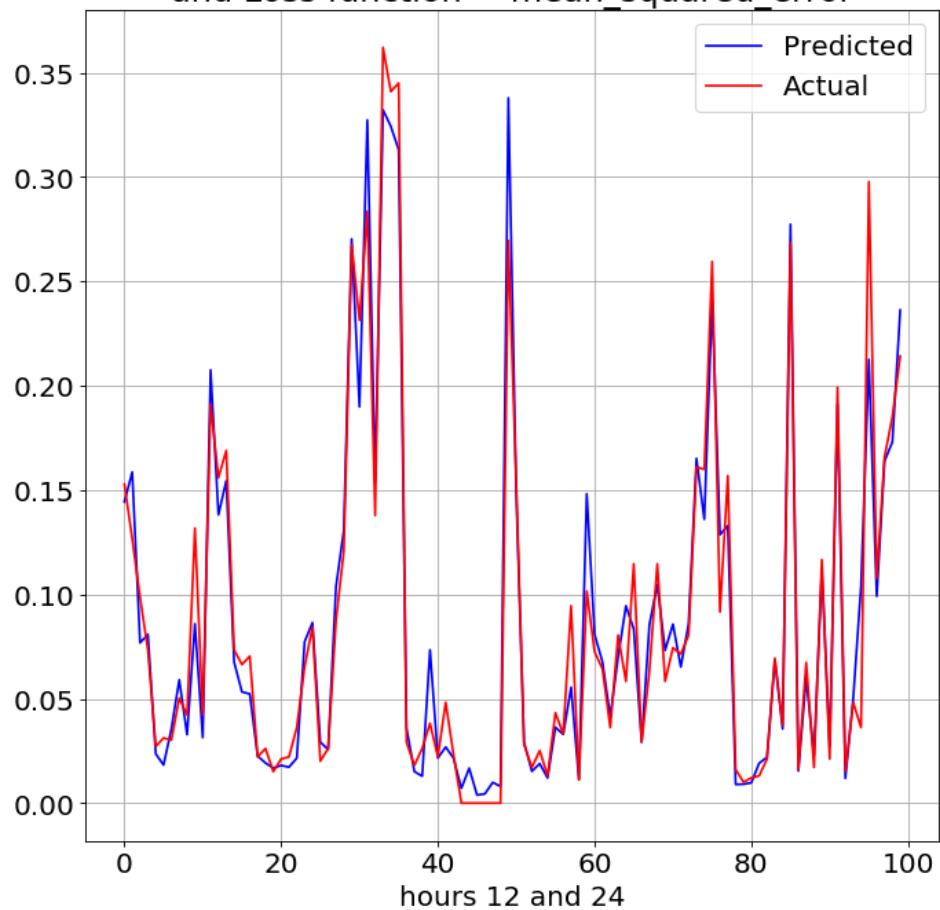
Predicted and Actual Test Data using LSTM Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 6-2

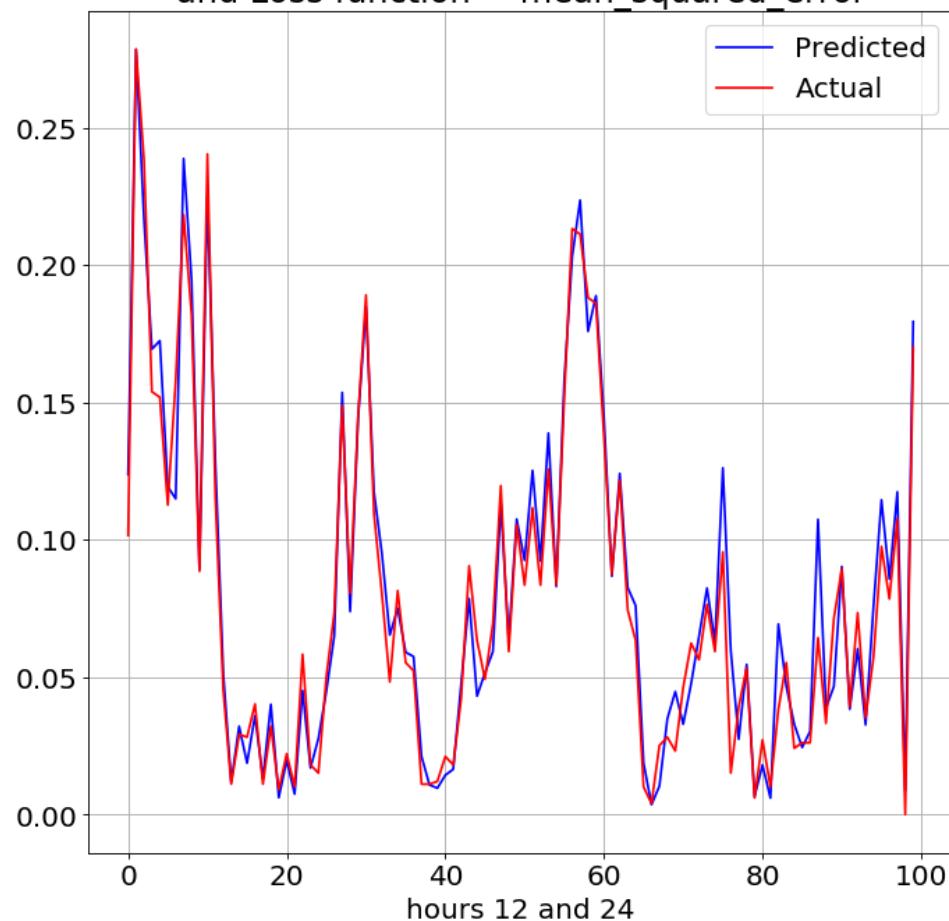
حال از لایه GRU استفاده می کنیم و شبکه را دوباره آموزش می دهیم. پس از آموزش شبکه ، نتیجه به صورت شکل های 6-3 و 6-4 خواهد بود.

Predicted and Actual Train Data using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



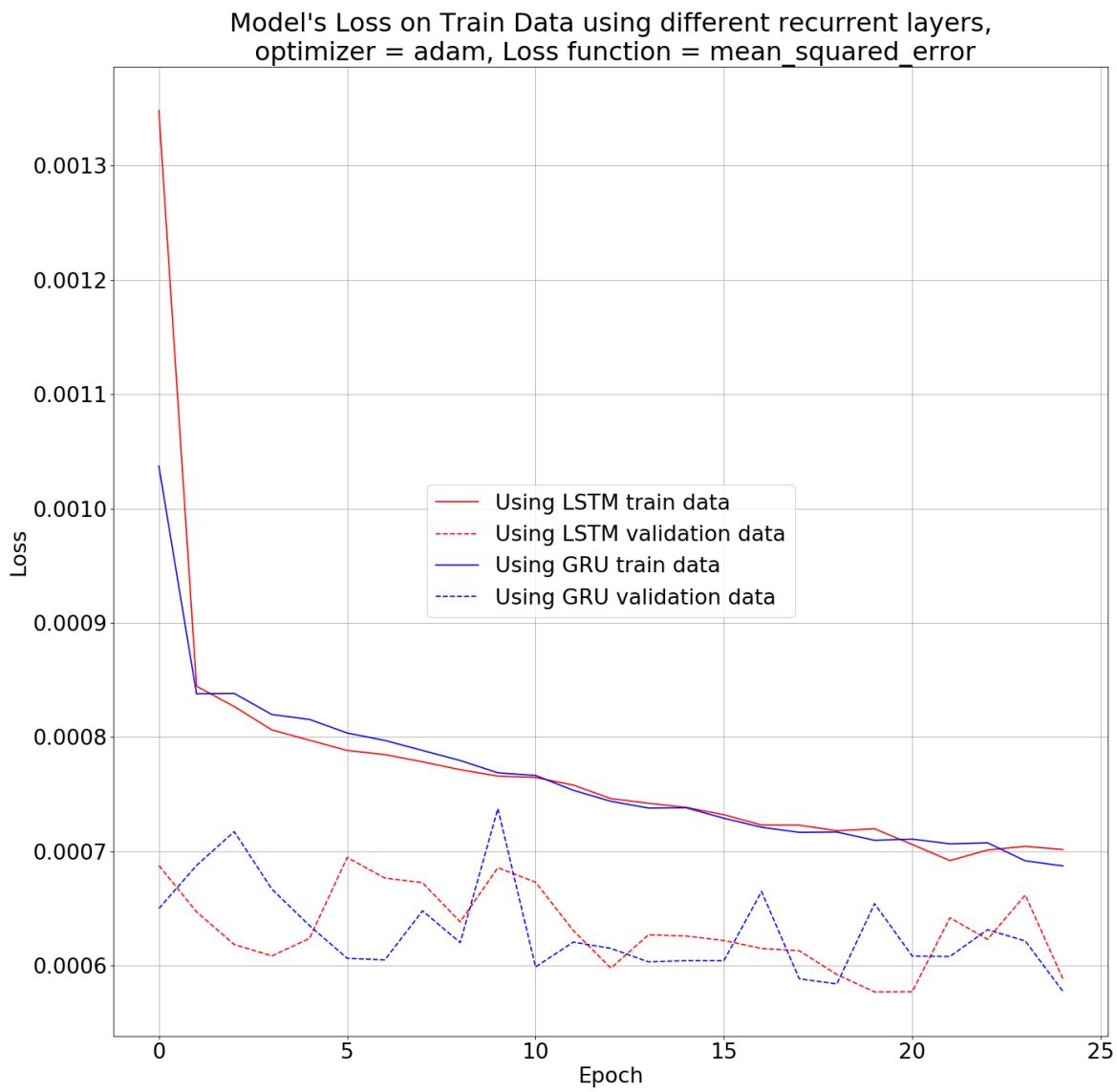
شكل ٦-٣

Predicted and Actual Test Data using GRU Layer, Optimizer = adam
and Loss function = mean_squared_error



شکل 6-4

نمودار loss دو شبکه فوق در شکل 6-5 رسم شده است.



شکل 6-5: نمودار loss

همانطور که مشاهده می شود عملکرد دو لایه از لحاظ loss روی داده های train مانند همیگر است.

نتایج دو شبکه فوق روی داده های test و زمان اجرای آموزش دو شبکه در شکل 6-6 نشان داده شده

است:

```
LSTM layer:  
Time taken = 174.55785536766052 seconds  
Test evaluation:  
MSE=0.0004945827661756014, MAE=0.0004945827531628311
```

```
GRU layer:  
Time taken = 188.89175987243652 seconds  
Test evaluation:  
MSE=0.0005121526483447106, MAE=0.0005121525609865785
```

شکل 6-6: نتایج شبکه روی داده های تست وقتی دیتاست اولیه نقصان داده داشته باشد

برای مقایسه بهتر دوباره نتایج شبکه قسمت اول را که با دیتاست بدون نقصان داده اجرا کرده بودیم را رسم می کنیم که در شکل 7-6 نشان داده شده است.

```
RNN layer:  
Time taken = [102.5763590335846] seconds  
Test evaluation:  
MSE=0.0004968974855208737, MAE=0.0004968975554220378
```

```
LSTM layer:  
Time taken = [195.58380937576294] seconds  
Test evaluation:  
MSE=0.0005318282384252645, MAE=0.0005318281473591924
```

```
GRU layer:  
Time taken = [180.7485213279724] seconds  
Test evaluation:  
MSE=0.0004998963947630497, MAE=0.0004998964723199606
```

شکل 6-7: نتایج وقتی نقصان داده نداشتیم

با مقایسه شکل 6-6 و 6-7 مشاهده می شود نتایج وقتی نقصان داده نداشتیم تنها کمتر از 0.0001 بهتر از حالتیست که در ابتداء نقصان داده داشته باشیم. نتیجه می گیریم که مقادیر missing value را با دقت بسیار خوبی handle کرده ایم.