مقدمه:

در این پروژه می‌خواهیم با استفاده از یک دیتا ست مربوط به وضعیت آب و هوا در ساعت‌های مختلف در سال‌های مختلف شبکه ای طراحی کنیم تا بتواند با توجه به وضعیت هوا در گذشته مقدار آلودگی هوا را در ۱ ساعت بعد پیش‌بینی کند برای این کار نیاز به استفاده از شبکه‌های حافظه دار و بازگشتی داریم.

طراحی شبکه:

برای ایجاد یک شبکه ابتدا نیاز داریم تا داده‌های خود را تمیز کنیم.
برای این کار ابتدا باید داده‌ها را از فایل بخوانیم. داده‌ها شامل ۸ ستون اطلاعات است که شرایط هوا را می‌گوید و برای نشان ه گزاری ردیف ها از تاریخ آن روز به همراه ساعت آن استفاده می‌کنیم به این معنا که در هر ساعد در هر تاریخ ما اطلاعات هوا را در اختیار داریم
سپس باید این داده‌های خوانده شده را مرتب و تمییز کنیم.
برای این کار ابتدا ستون مربوط به جهت باد که یک نوشته است را باید تغییر دهیم برای این کار می‌توانیم از one Hot کردن و یا لیبل کردن آن استفاده کنیم در اینجا چون مقادیر ما ۴ تا است و مقدار دیگری نیز به آن اضافه نمی‌شود از Label کردن آن‌ها استفاده کرده‌ایم تا تمام نوشته‌ها را به عدد های آن نسبت دهیم

سپس تمام مقادیر را به ممیز شناور تبدیل می‌کنیم تا محاسبات همگی به یک شکل باشند

سپس با استفاده از MinMaxScaller تمام ستون‌ها را نرمالایز می‌کنیم تا شبکه بهتر بتواند وزن ها را تغییر دهد و دچار overfitt نشود

و در آخر چون باید یک دوره از زمان را به شبکه بدهیم نیاز داریم تا دیتا ها را به دوره های مختلف تبدیل کنیم برای اینکار باید یک پنجره از بازه زمانی تعیین کنیم به این معنا که هر ردیف ما دارای چه تعداد زمان باشد تا به شبکه بدهیم و آن را آموزش بدهیم

برای تمیز کردن دیتا ردیف های اول که مقدار آلودگی هوا ندارند را حذف کردیم و بقیه موارد که نداشتند آن مقدار را با ۰ جایگزین کردیم و به این صورت ۸ ستون داده و یک ستون برای نمایش زمان استفاده کرده‌ایم

کد مربوط به آن در زیر آمده است:

```
# load dataset
dataset = read_csv('pollution.csv', header=0, index_col=0)
values = dataset.values
# integer encode direction
encoder = LabelEncoder()
values[:, 4] = encoder.fit_transform(values[:, 4])
# ensure all data is float
values = values.astype('float32')
# normalize features
scaler = MinMaxScaler(feature_range=(0, 1))
scaled = scaler.fit_transform(values)
# specify the number of lag hours
n_hours = 1
n_features = 8
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)
print(reframed.shape)
print(reframed)
```

سپس برای دادن دیتا ها به شبکه لازم است آن‌ها را به دو دسته تست و ترین تقسیم کنیم و سپس از آن خروجی و ورودی را جدا کنیم

```
# split into train and test sets
values = reframed.values
split = 12000
train = values[:split, :]
test = values[split:split+3000, :]

# split into input and outputs
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print(train_X.shape, len(train_X), train_y.shape)

# reshape input to be 3D [samples, timesteps, features]
train_X = train_X.reshape((train_X.shape[0], n_hours, n_features))
test_X = test_X.reshape((test_X.shape[0], n_hours, n_features))
print(train_X.shape, train_y.shape, test_X.shape, test_y.shape)
```

و سپس ورودی ها را به ارایه های ۳ بعدی که شامل تاریخ ساعت‌های قبلی و ستون‌های داده است تقسیم می‌کنیم

ساخت شبکه:

حال باید شبکه را طراحی کنیم
شبکه را ابتدا با حالت LSTM و ۵۰ نورون آن را آموزش می‌دهیم و در خروجی یک لایه ۱ نورونی برای پیش‌بینی قرار می‌دهیم و شبکه را با بهینه ساز های مختلف و ارور های متفاوت کامپایل می‌کنیم
در اینجا شبکه را برای ۲۰ epoch و با batch_size =64 انجام داده ایم

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam', metrics=['accuracy'])
# fit network
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_data=(test_X, test_y), verbose=2,
                    shuffle=False)
```

سپس نمودار های مربوط به دقت و لاس را برای هر دو داده ی تست و ترین می‌کشیم

```
# plot history
pyplot.plot(history.history['loss'], label='train')
pyplot.plot(history.history['val_loss'], label='validation')
pyplot.legend()
pyplot.show()

# plot history
pyplot.plot(history.history['accuracy'], label='train')
pyplot.plot(history.history['val_accuracy'], label='validation')
pyplot.legend()
pyplot.show()
```

سپس با همین شبکه برای داده‌های تست خروجی را پیش‌بینی می‌کنیم ولی در اینجا چون ما ابتدا داده‌ها را اسکیل کرده بودیم نیاز داریم تا این اسکیل را برگردانیم و دوباره به داده‌های اصلی تبدیل کنیم
و سپس با مقادیر واعی آن مقایسه کنیم و نمودار این دو مقدار را بکشیم

```
# make a prediction
yhat = model.predict(test_X)
test_X = test_X.reshape((test_X.shape[0], n_hours * n_features))
# invert scaling for forecast
inv_yhat = concatenate((yhat, test_X[:, -7:]), axis=1)
inv_yhat = scaler.inverse_transform(inv_yhat)
inv_yhat = inv_yhat[:, 0]
# invert scaling for actual
test_y = test_y.reshape((len(test_y), 1))
inv_y = concatenate((test_y, test_X[:, -7:]), axis=1)
inv_y = scaler.inverse_transform(inv_y)
inv_y = inv_y[:, 0]
print(inv_yhat)
print(inv_y)

# plot history
pyplot.plot(inv_yhat, 'bo', label='predict')
pyplot.plot(inv_y, 'yo', label='real')
pyplot.legend()
pyplot.show()
```

```
# calculate RMSE
rmse = sqrt(mean_squared_error(inv_y, inv_yhat))
print('Test RMSE: %.3f' % rmse)
```

و در آخر برای مقایسه شبکه‌های مختلف مقدار mean square error را برای داده‌های واقعی و پیشبینی شده بدست می‌آوریم تا معیاری برای دقت شبکه باشد
حال شبکه را با بهینه ساز ها توابع لاس و حالت‌های مختلف به ترتیب درست می‌کنیم و نمودار های آن را می‌کشیم

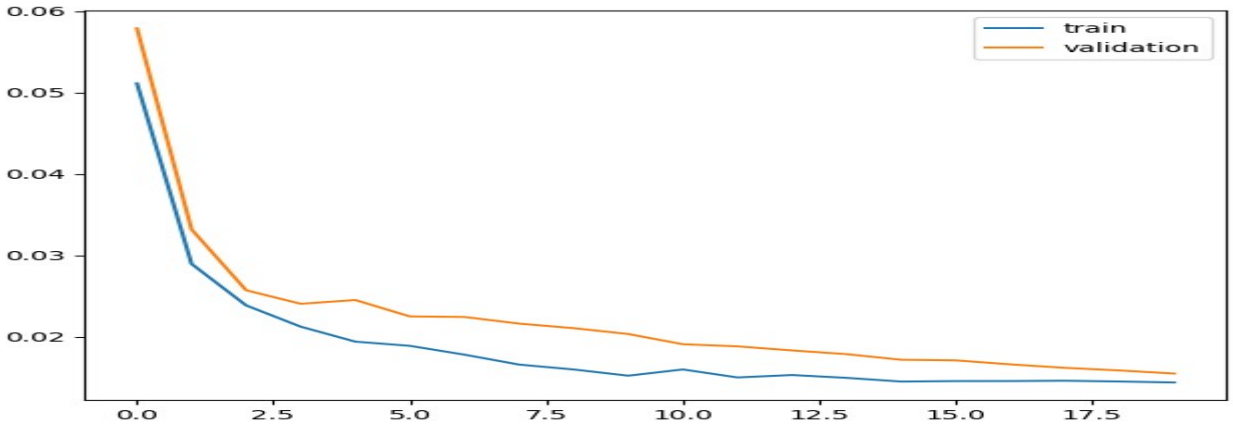
LSTM , adam, mae(1

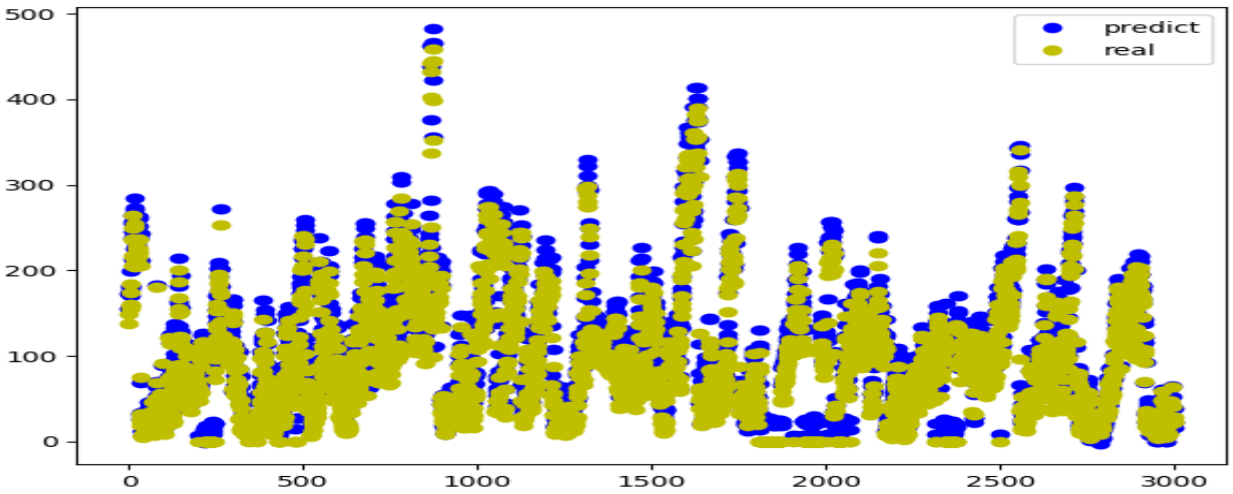
```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam', metrics=['accuracy'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

و بعد از اجرا داریم:

```
Epoch 20/20
 - 4s - loss: 0.0143 - accuracy: 0.0745 - val_loss: 0.0157 - val_accuracy: 0.1267
Elapsed Time for fitiing:  75.01484370231628
Test RMSE: 26.840
```

و بعد از پیش‌بینی خروجی ها هر ۱۲ ساعت می‌توانیم آلودگی را گزارش دهیم خروجی را در فایل تکست می‌ریزیم
LSTM_Adam_MAE.txt





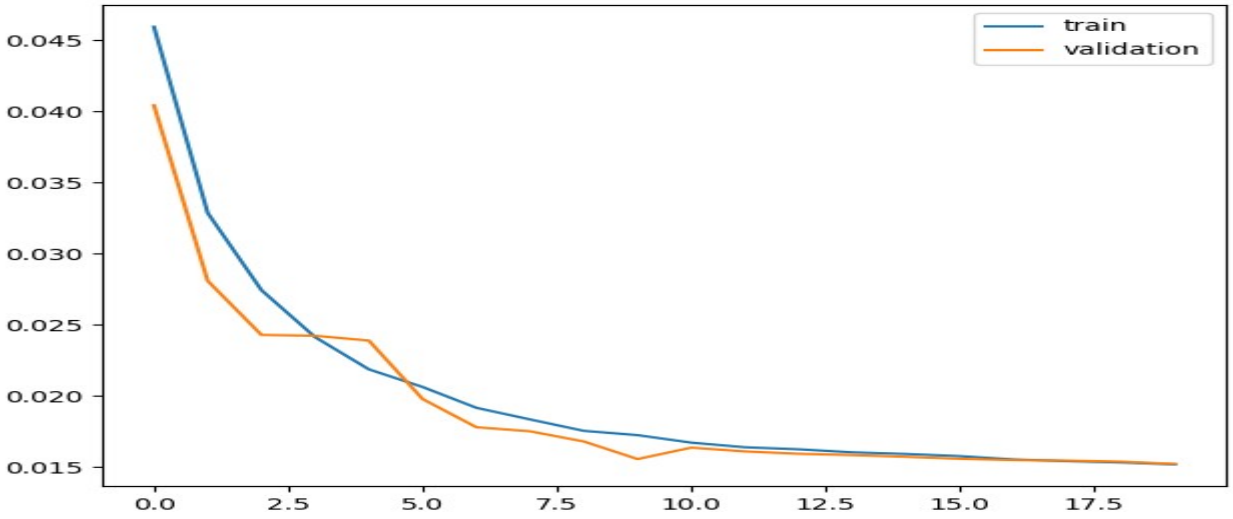
2) LSTM , RMSProp, MAE

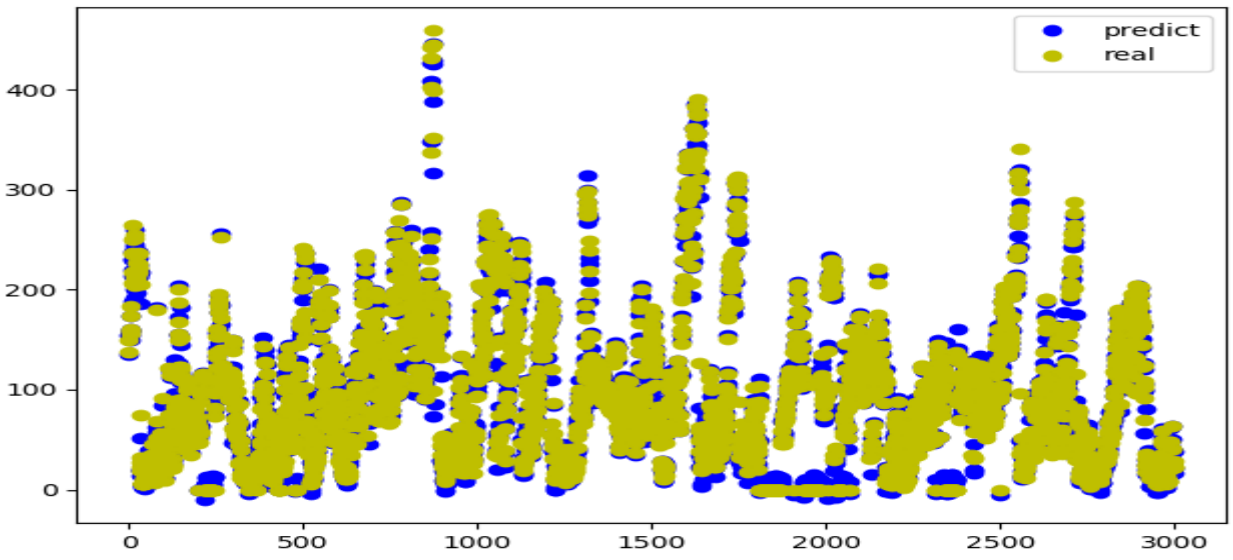
```
model.compile(loss='mae', optimizer='rmsprop', metrics=['accuracy'])
```

بعد از اجرا داریم:

```
Epoch 20/20
 - 3s - loss: 0.0152 - accuracy: 0.0745 - val_loss: 0.0152 - val_accuracy: 0.1267
Elapsed Time for fitiing:  63.095338106155396
Test RMSE: 21.838
```

و بعد از پیش‌بینی خروجی ها هر ۱۲ ساعت می‌توانیم آلودگی را گزارش دهیم خروجی را در فایل تکست می‌ریزیم LSTM_RMSProp_MAE.txt





3) LSTM, ADAgrad, MAE

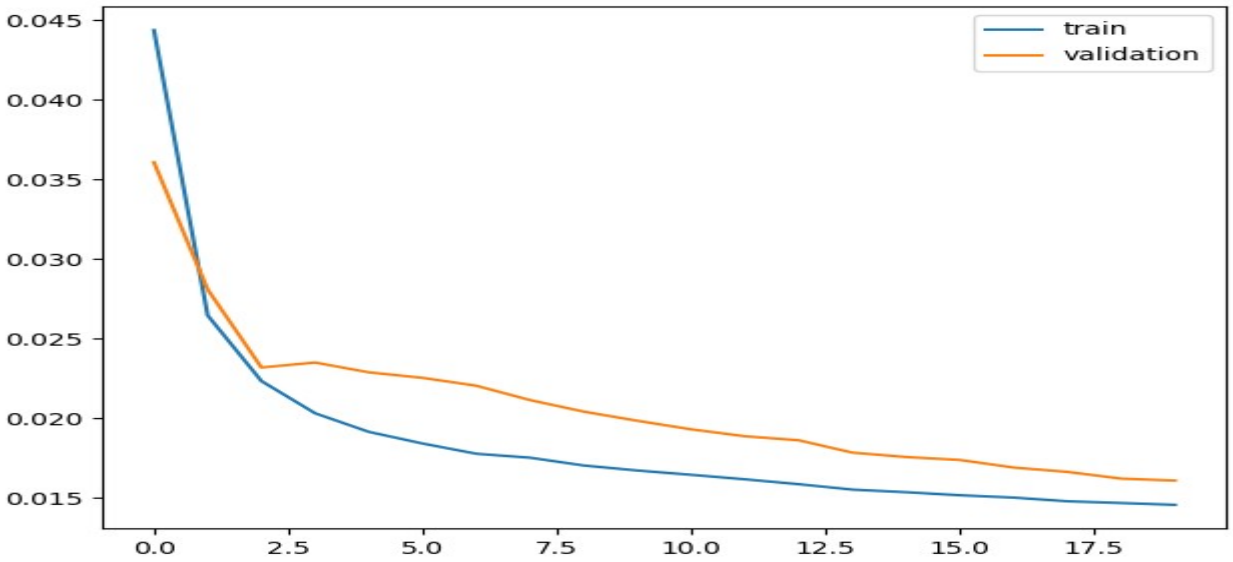
```
model.compile(loss='mae', optimizer='ADAgrad', metrics=['accuracy'])
```

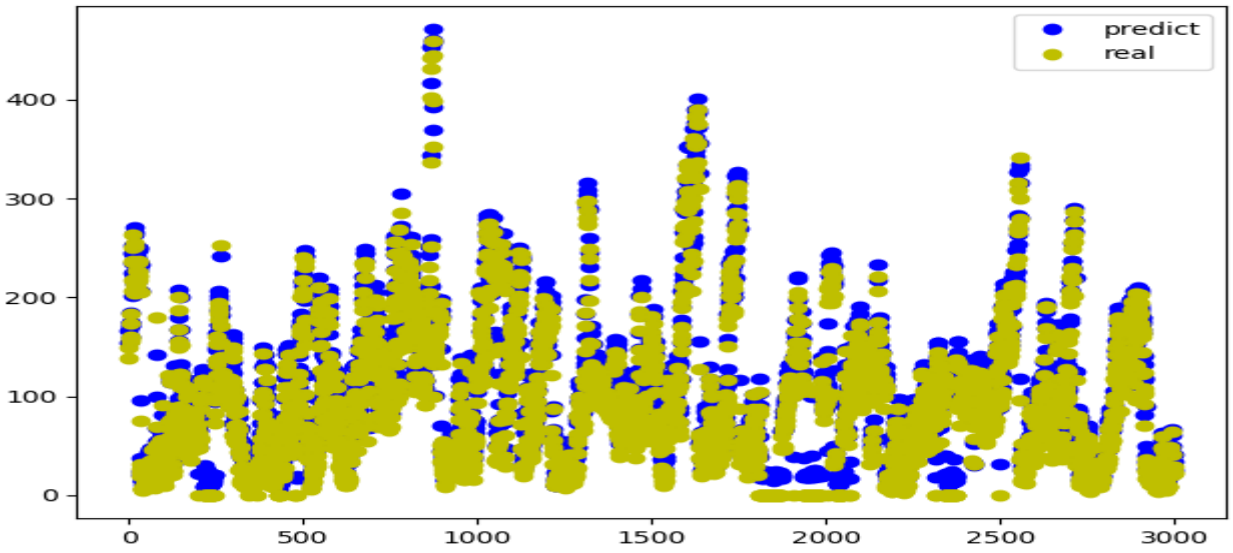
و بعد از اجرا:

```
Epoch 20/20
 - 3s - loss: 0.0146 - accuracy: 0.0745 - val_loss: 0.0161 - val_accuracy: 0.1267
Elapsed Time for fitiing:  64.325035572052
Test RMSE: 25.784
```

و خروجی در فایل:

LSTM_ADAgrad_MAE.txt



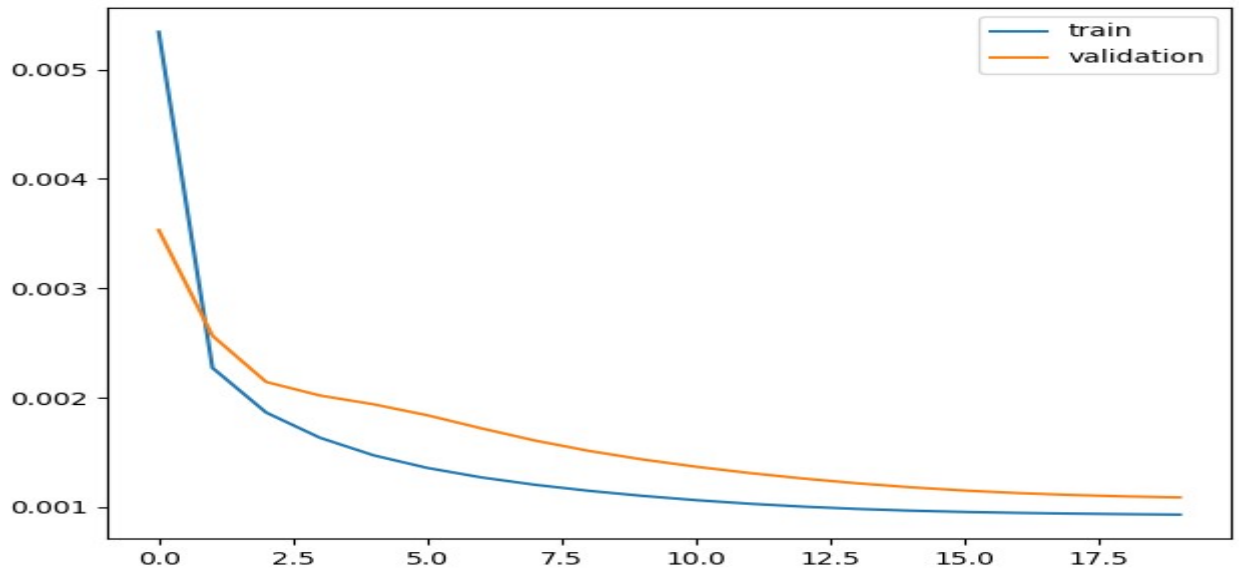


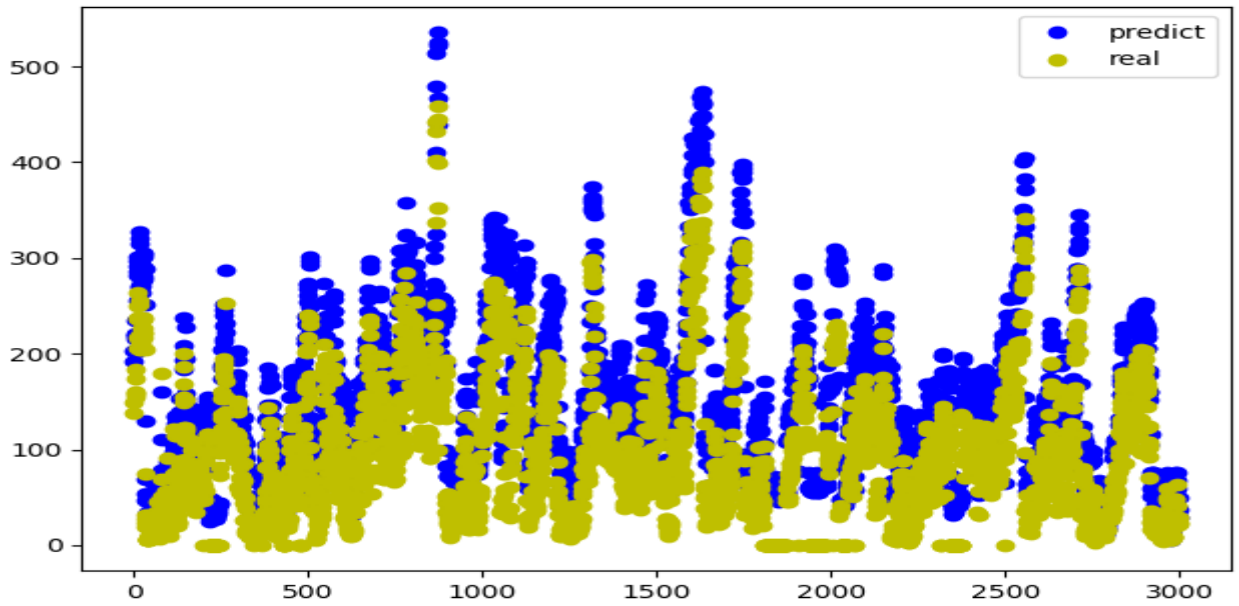
LSTM, Adam, MSE (4

```
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
```

```
Epoch 20/20
 - 3s - loss: 9.3130e-04 - accuracy: 0.0745 - val_loss: 0.0011 - val_accuracy: 0.1267
Elapsed Time for fitiing:  67.92519950866699
Test RMSE: 59.394
```

LSTM_Adam_MSE.txt



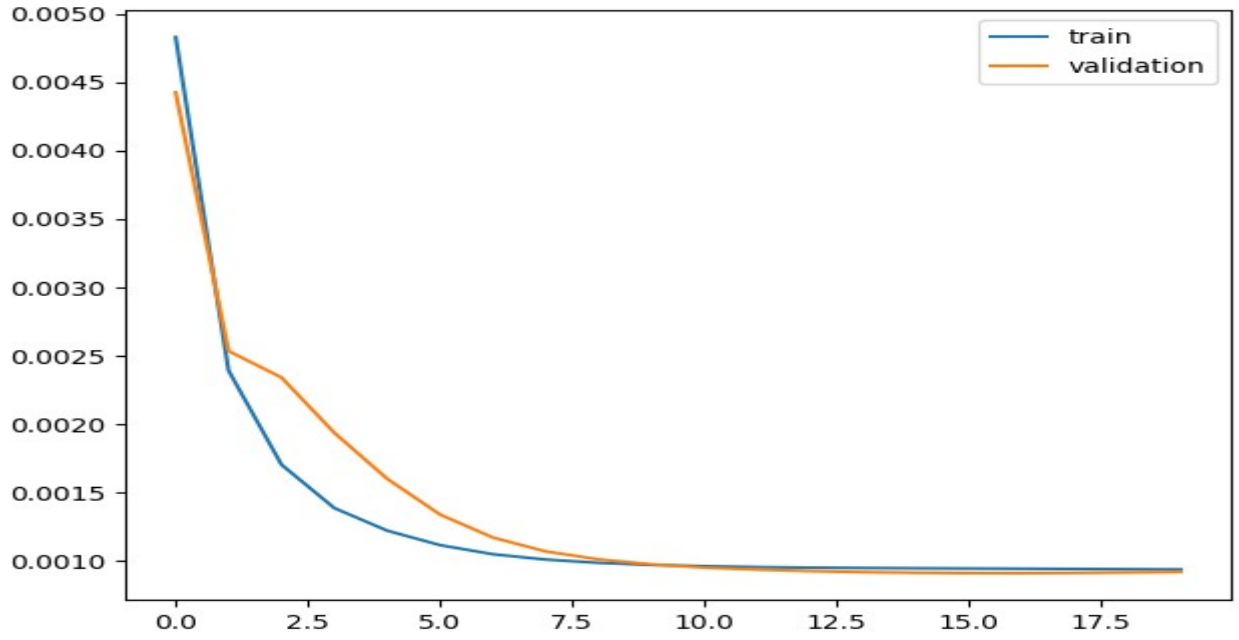


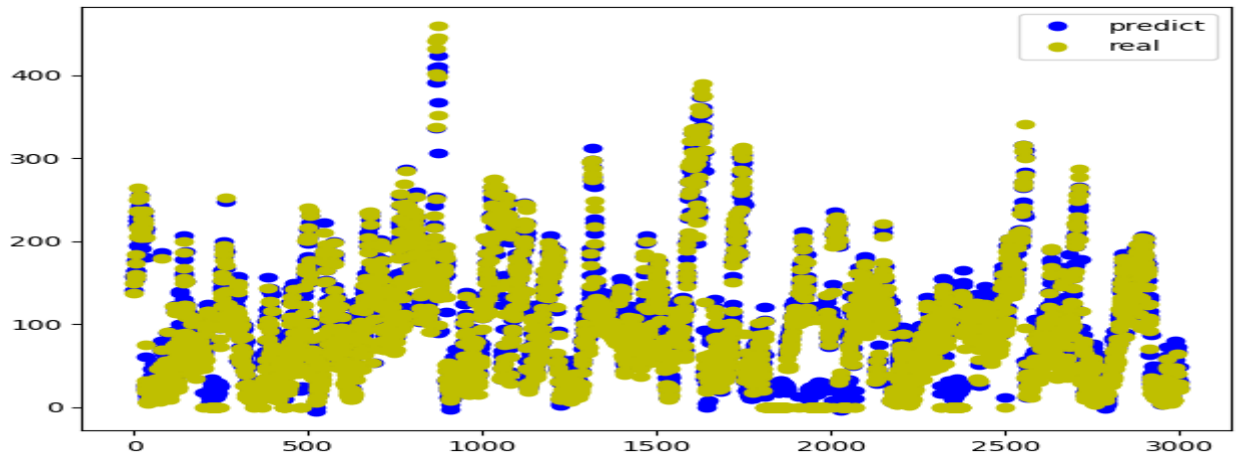
LSTM, RMSProp, MSE (5

```
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
```

```
Epoch 20/20
 - 3s - loss: 9.3778e-04 - accuracy: 0.0745 - val_loss: 9.1961e-04 - val_accuracy: 0.1267
Elapsed Time for fitiing:  80.52875661849976
Test RMSE: 24.471
```

LSTM_RMSProp_MSE.txt



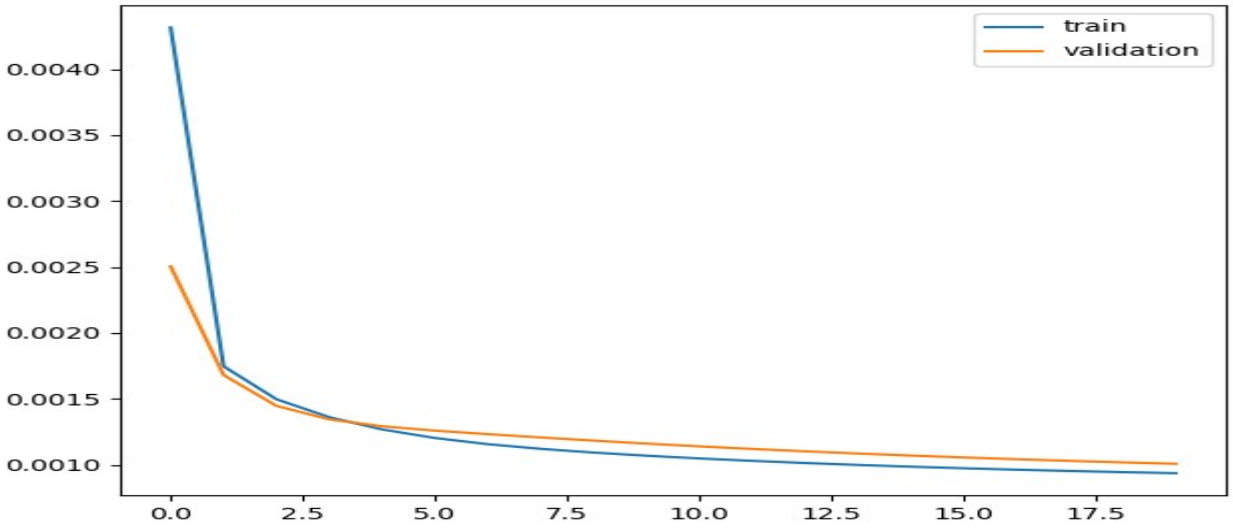


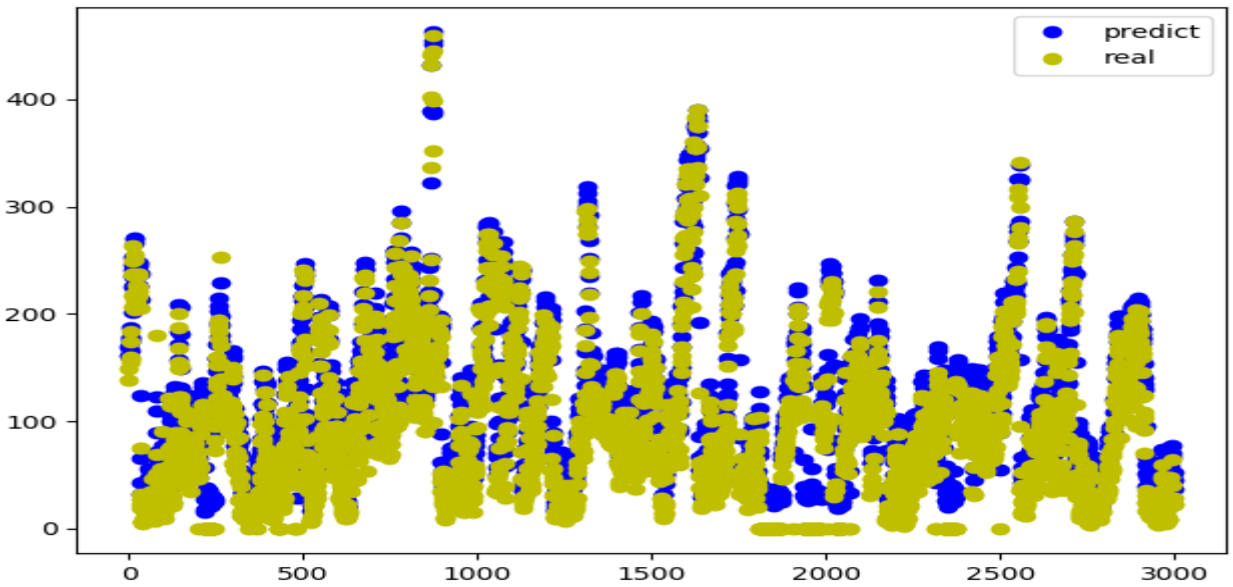
LSTM, ADAgrad, MSE (6

```
model.compile(loss='mse', optimizer='adagrad', metrics=['accuracy'])
```

```
Epoch 20/20
 - 3s - loss: 9.3554e-04 - accuracy: 0.0746 - val_loss: 0.0010 - val_accuracy: 0.1267
Elapsed Time for fitiing:  71.58344602584839
Test RMSE: 31.608
```

LSTM_ADAgrad_MSE.txt



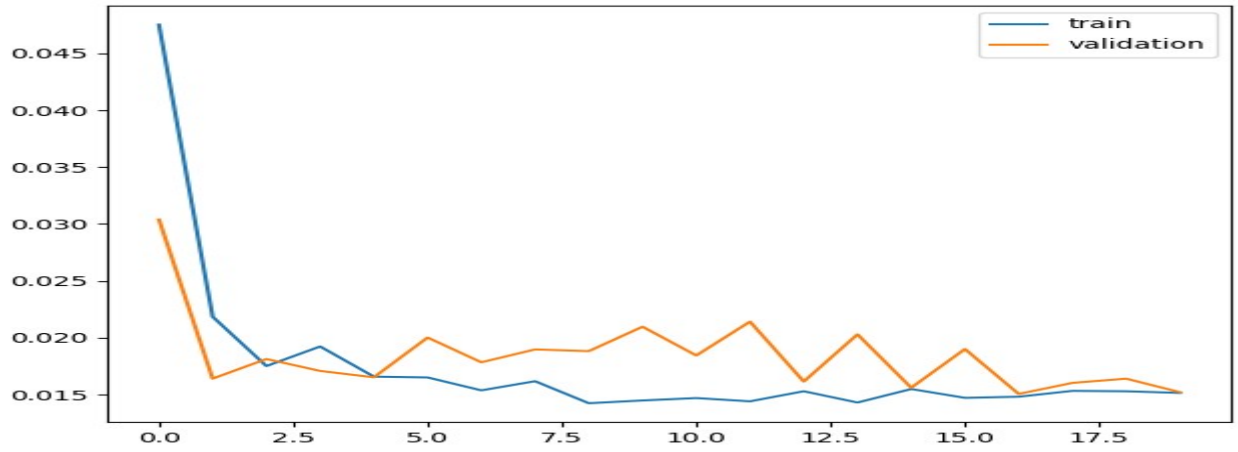


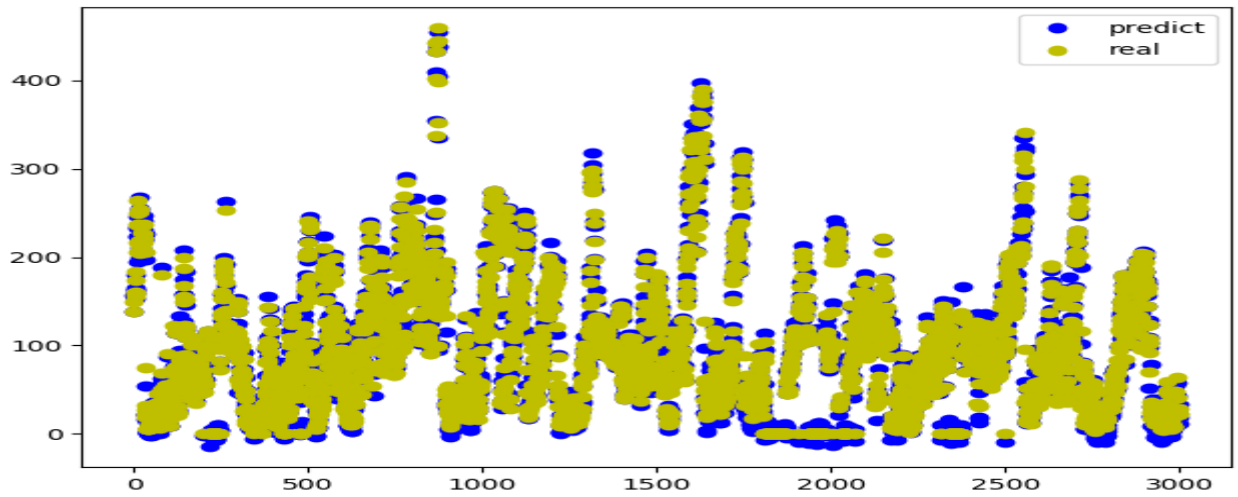
GRU, Adam, MAE (7

```
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adam', metrics=['accuracy'])
```

```
Epoch 20/20
 - 4s - loss: 0.0152 - accuracy: 0.0745 - val_loss: 0.0152 - val_accuracy: 0.1267
Elapsed Time for fitiing:  84.88402819633484
Test RMSE: 22.535
```

GRU_Adam_MAE.txt



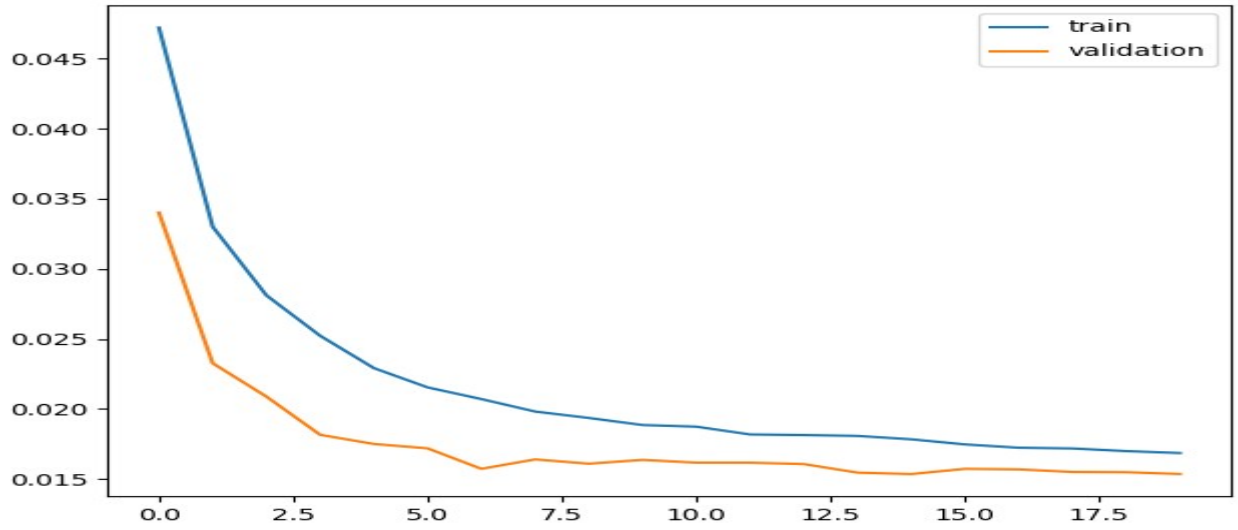


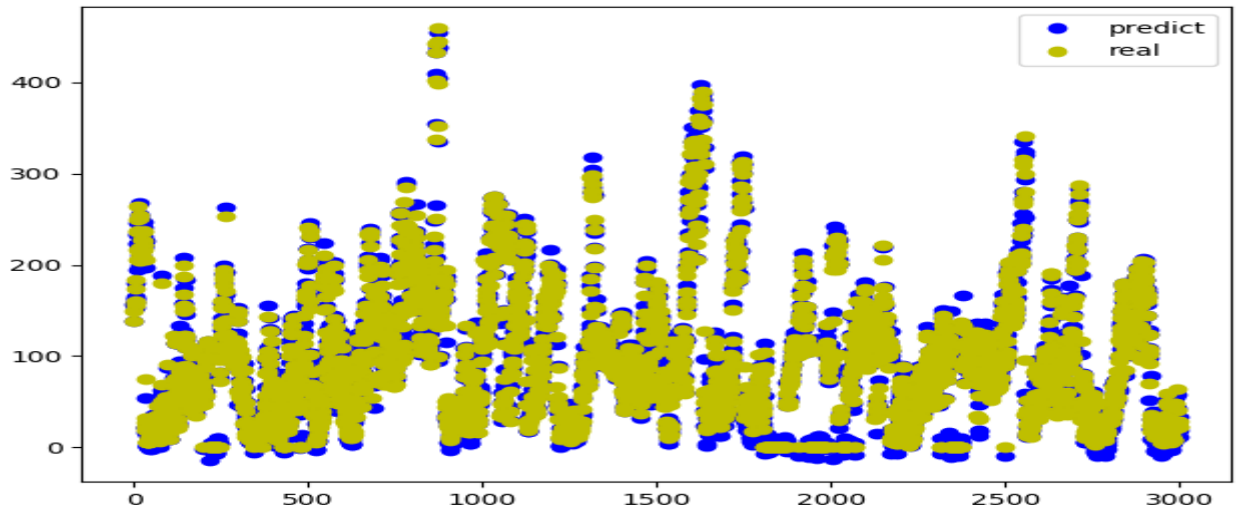
GRU, RMSProp, MAE(8

```
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['accuracy'])
```

```
Epoch 20/20
 - 4s - loss: 0.0169 - accuracy: 0.0745 - val_loss: 0.0154 - val_accuracy: 0.1267
Elapsed Time for fitiing:  87.98702049255371
Test RMSE: 21.996
```

GRU_RMSProp_MAE.txt

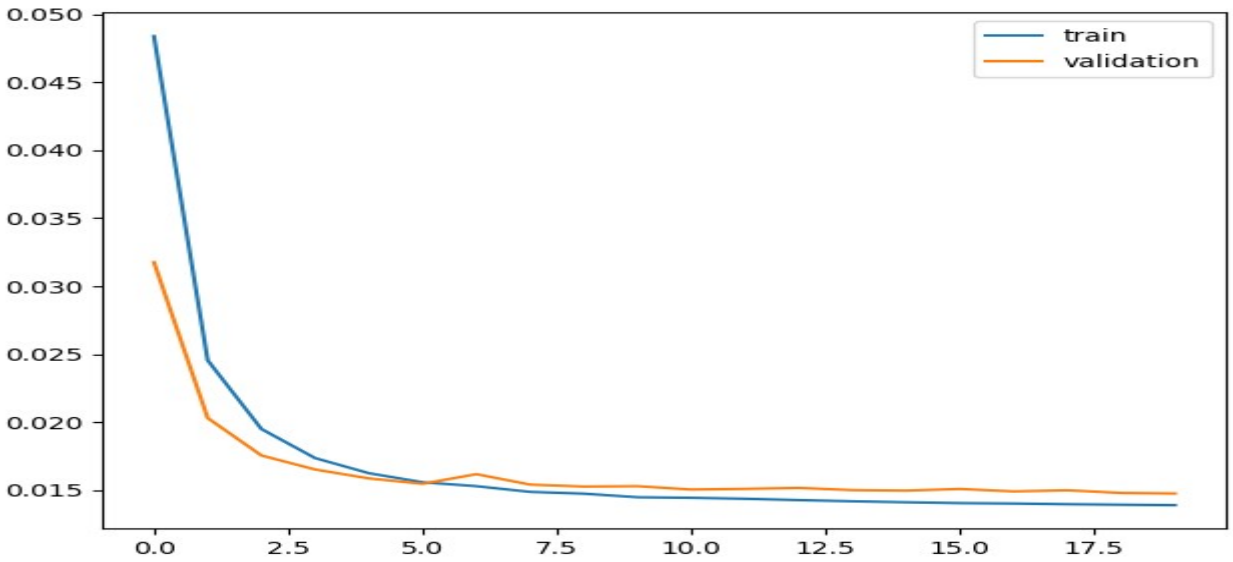


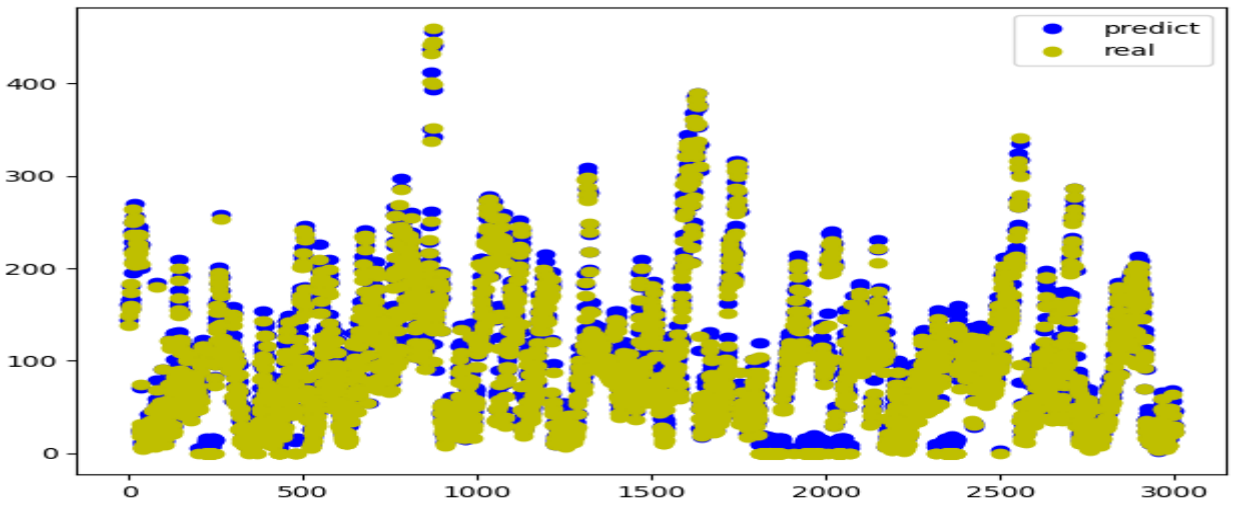


```
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adagrad', metrics=['accuracy'])
```

```
Epoch 20/20
 - 4s - loss: 0.0139 - accuracy: 0.0745 - val_loss: 0.0148 - val_accuracy: 0.1267
Elapsed Time for fitiing:  74.78105044364929
Test RMSE: 23.359
```

GRU_ADAgrad_MAE.txt



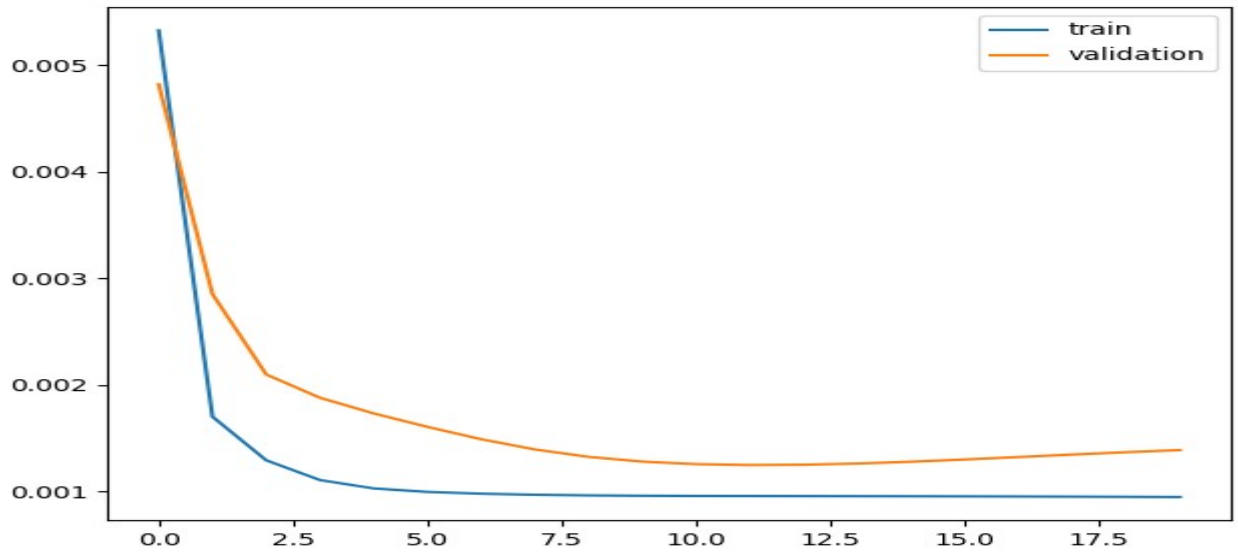


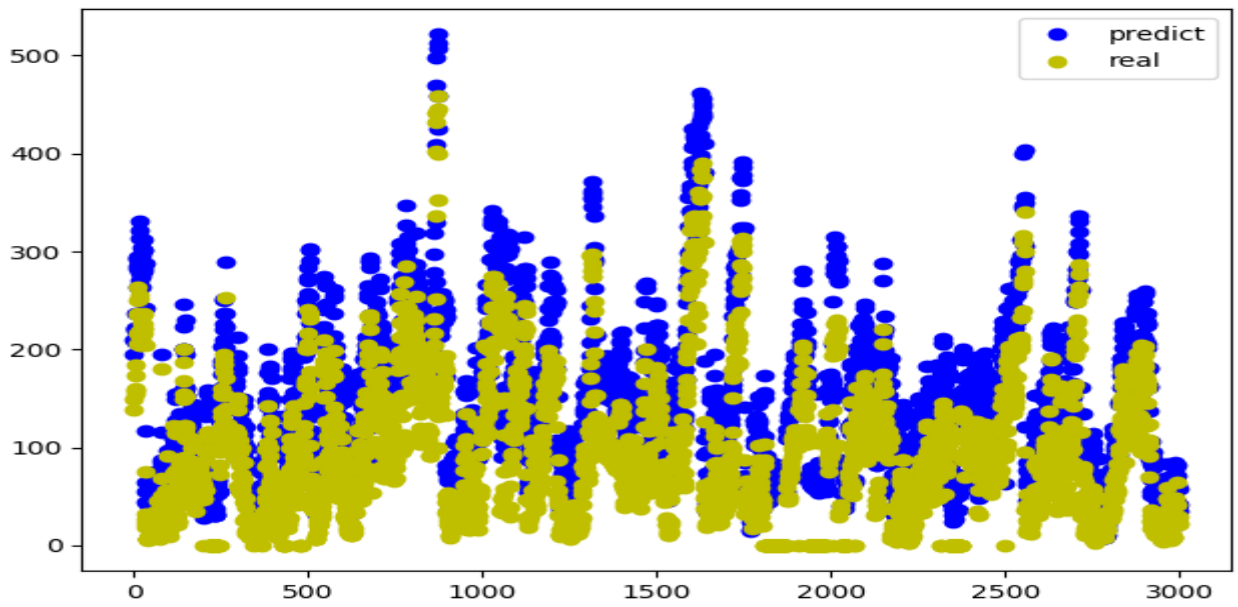
GRU, Adam, MSE(10

```
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
```

```
Epoch 20/20
 - 4s - loss: 9.4573e-04 - accuracy: 0.0745 - val_loss: 0.0014 - val_accuracy: 0.1267
Elapsed Time for fitiing:  93.52144622802734
Test RMSE: 60.674
```

GRU_Adam_MSE.txt



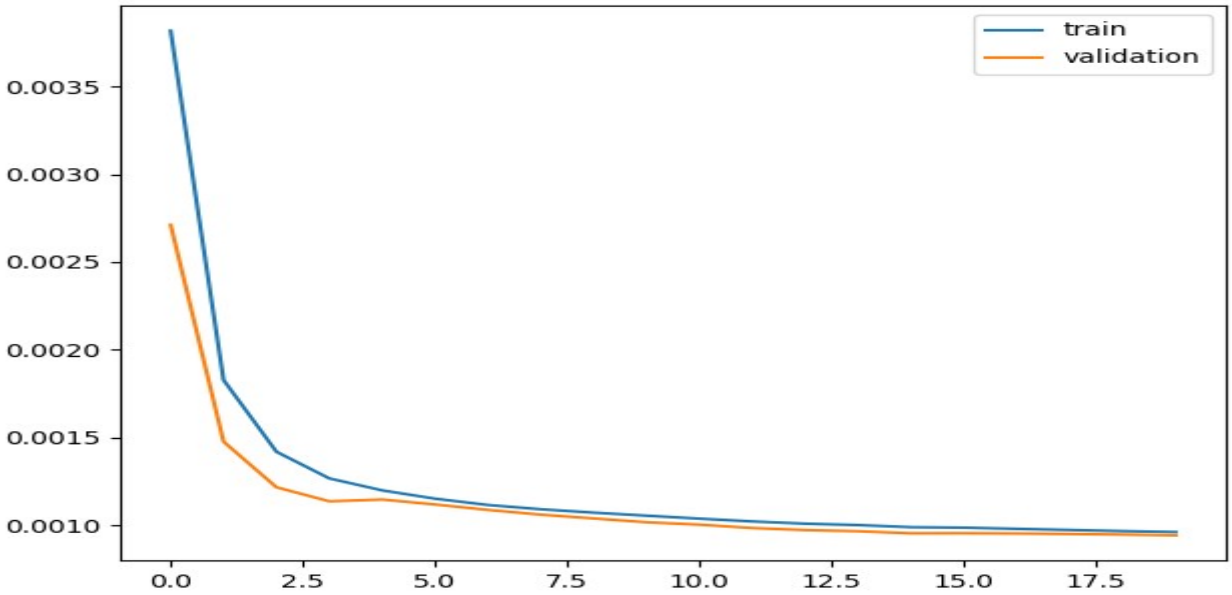


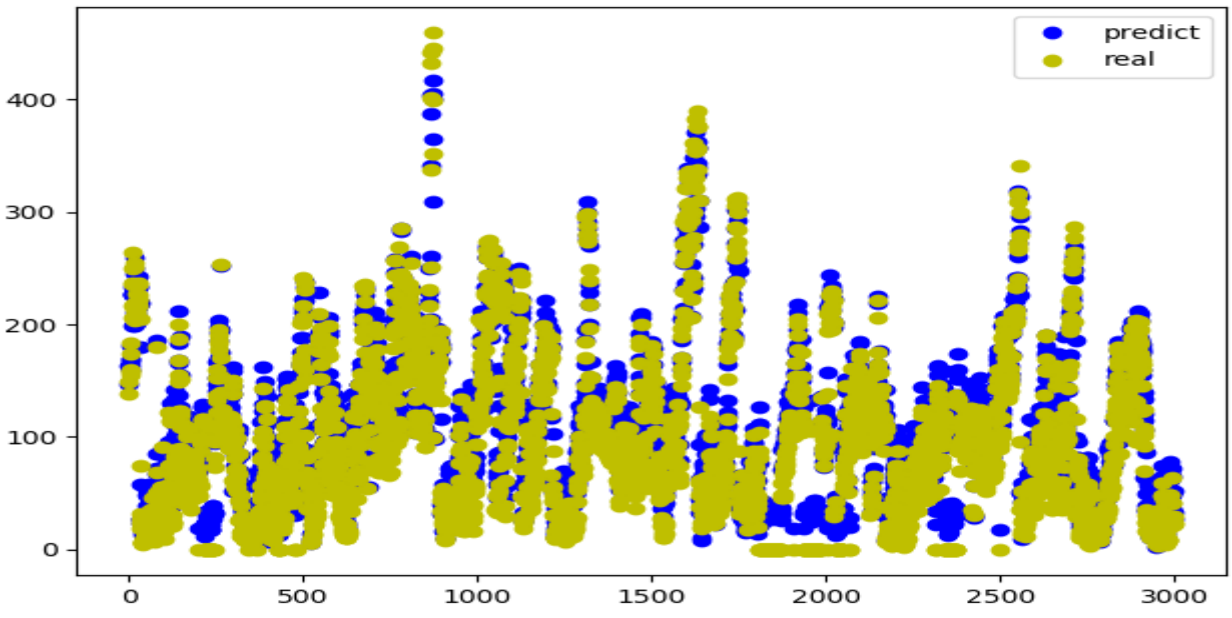
GRU, RMSProp, MSE(11

```
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='rmsprop', metrics=['accuracy'])
```

```
Epoch 20/20
 - 4s - loss: 9.6058e-04 - accuracy: 0.0745 - val_loss: 9.4181e-04 - val_accuracy: 0.1267
Elapsed Time for fitiing:  87.19293546676636
Test RMSE: 27.594
```

GRU_RMSProp_MSE.txt



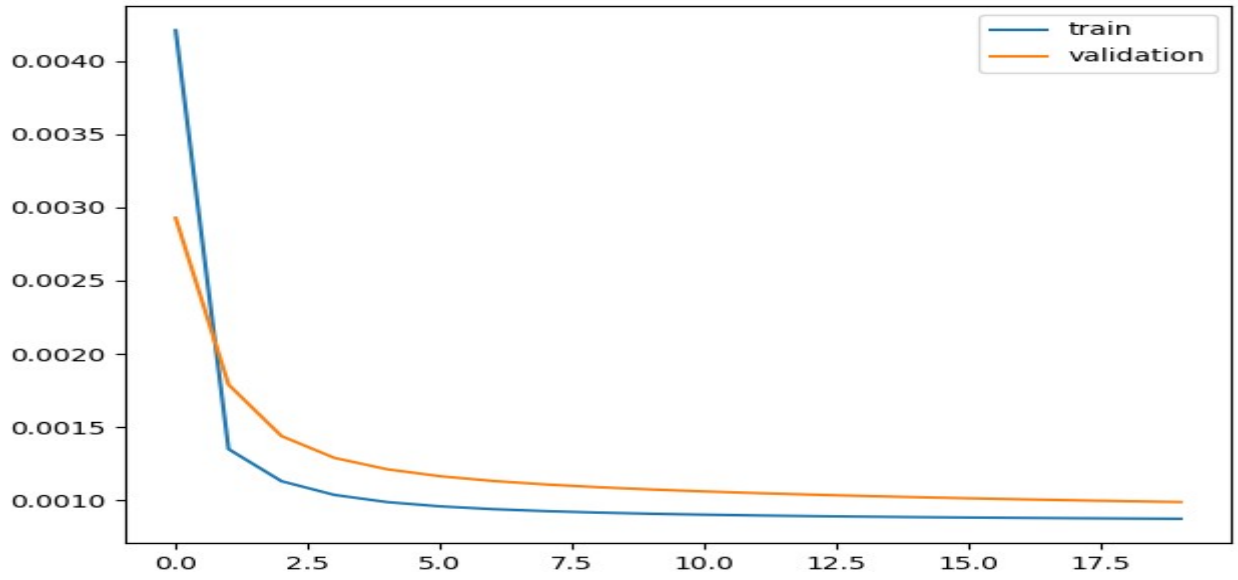


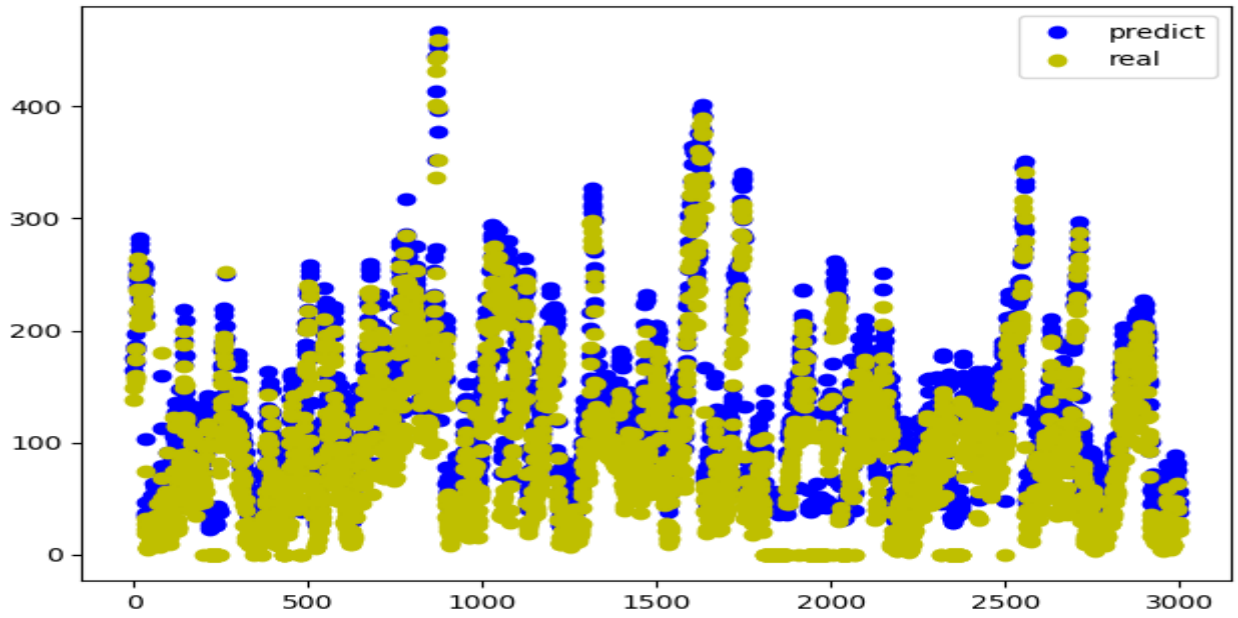
GRU, ADAgrad, MSE (12

```
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adagrad', metrics=['accuracy'])
```

```
Epoch 20/20
 - 5s - loss: 8.7323e-04 - accuracy: 0.0745 - val_loss: 9.8741e-04 - val_accuracy: 0.1267
Elapsed Time for fitiing:  90.49602675437927
Test RMSE: 39.246
```

GRU_ADAgrad_MSE.txt



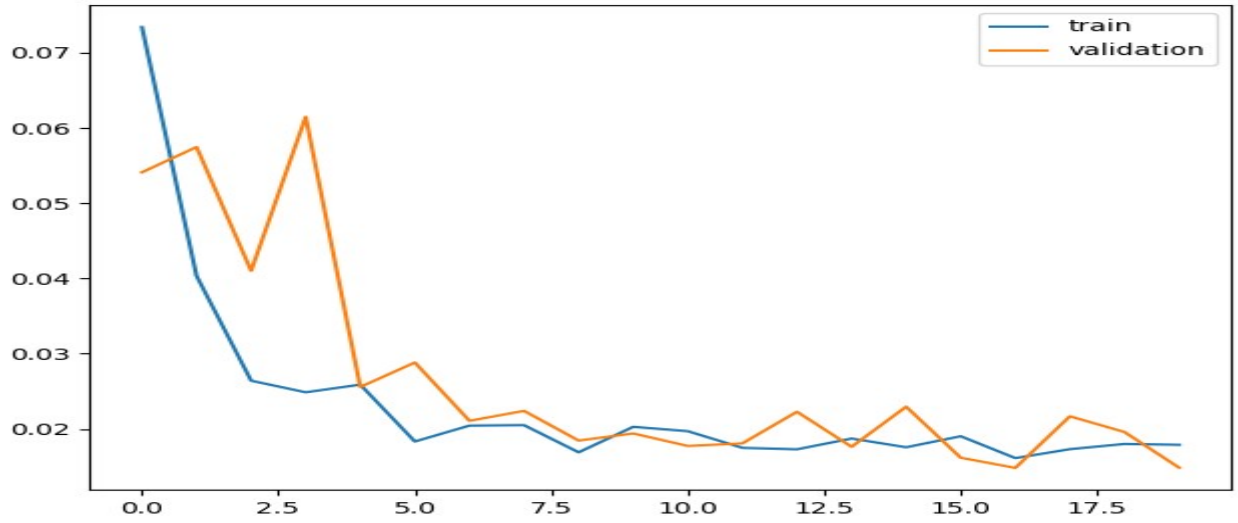


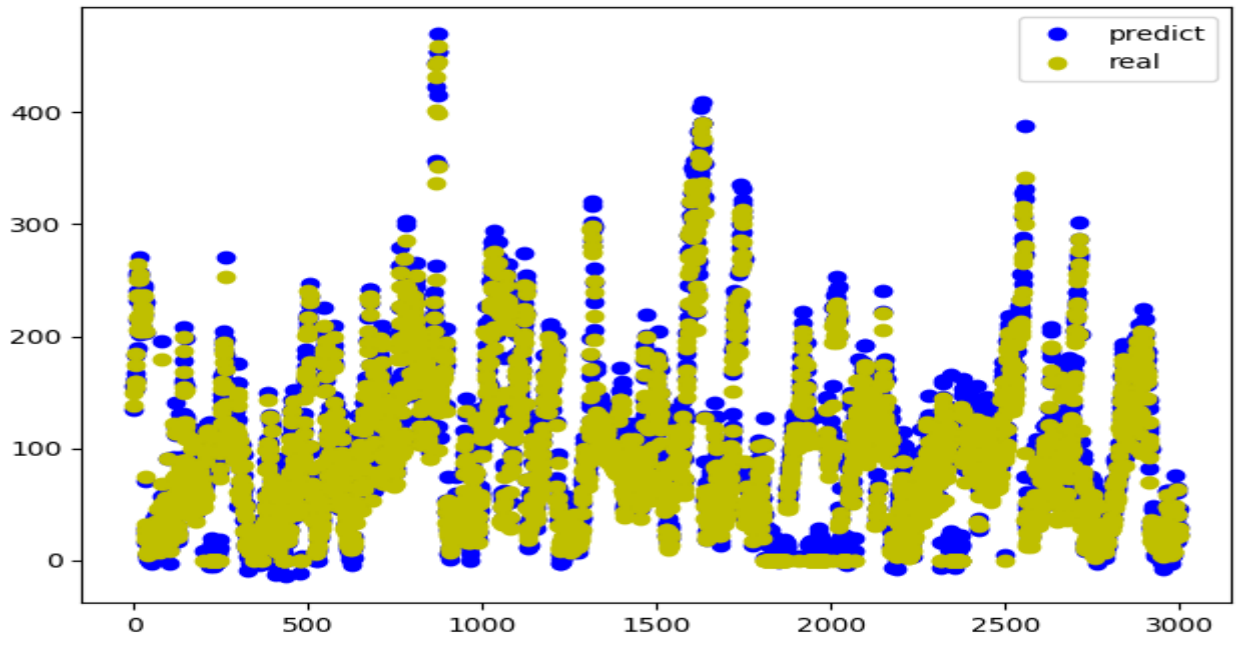
RNN, Adam, MAE(13

```
# design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='Adam', metrics=['accuracy'])
```

```
Epoch 20/20
 - 2s - loss: 0.0179 - accuracy: 0.0745 - val_loss: 0.0148 - val_accuracy: 0.1267
Elapsed Time for fitiing:  39.93316841125488
Test RMSE: 25.185
```

RNN_Adam_MAE.txt



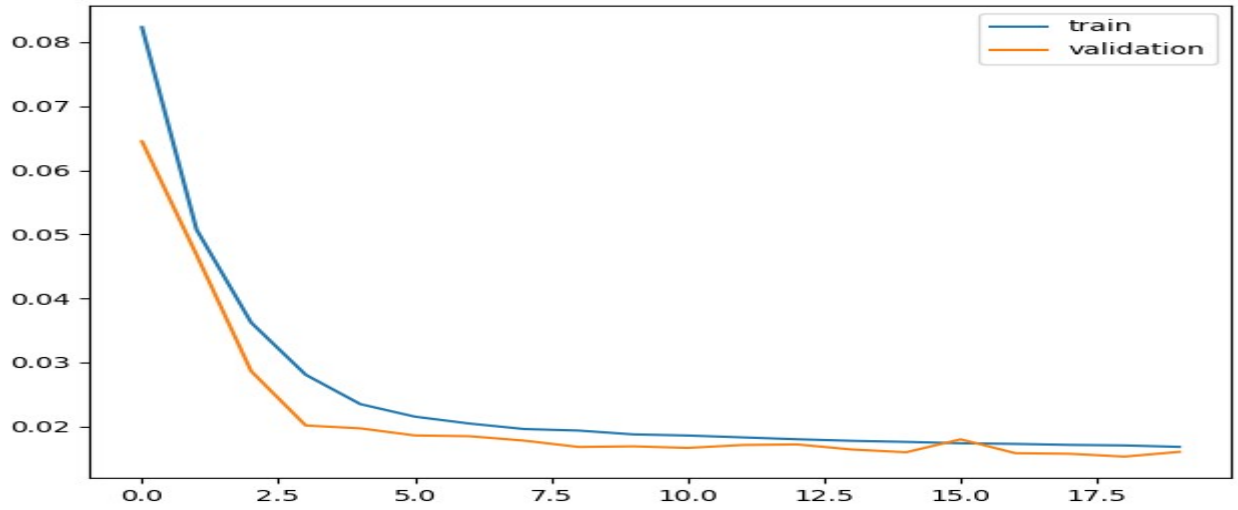


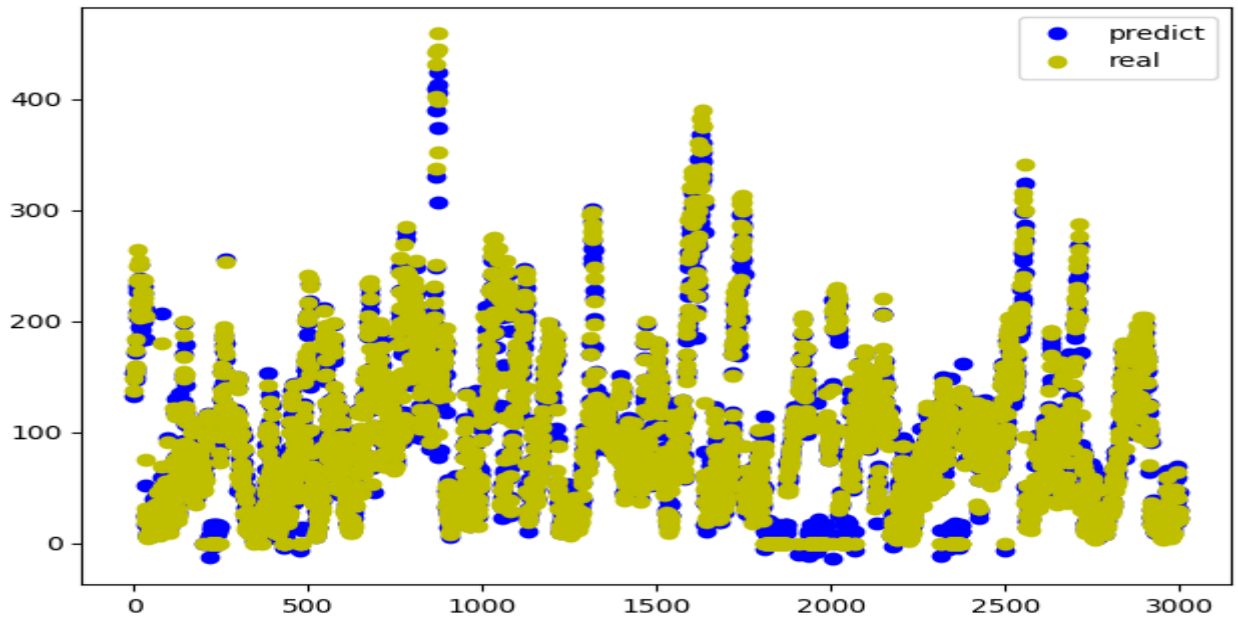
RNN, RMSProp, MAE(14

```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['accuracy'])
```

```
Epoch 20/20
 - 2s - loss: 0.0168 - accuracy: 0.0745 - val_loss: 0.0160 - val_accuracy: 0.1267
Elapsed Time for fitiing:  37.270283222198486
Test RMSE: 22.362
```

RNN_RMSProp_MAE.txt



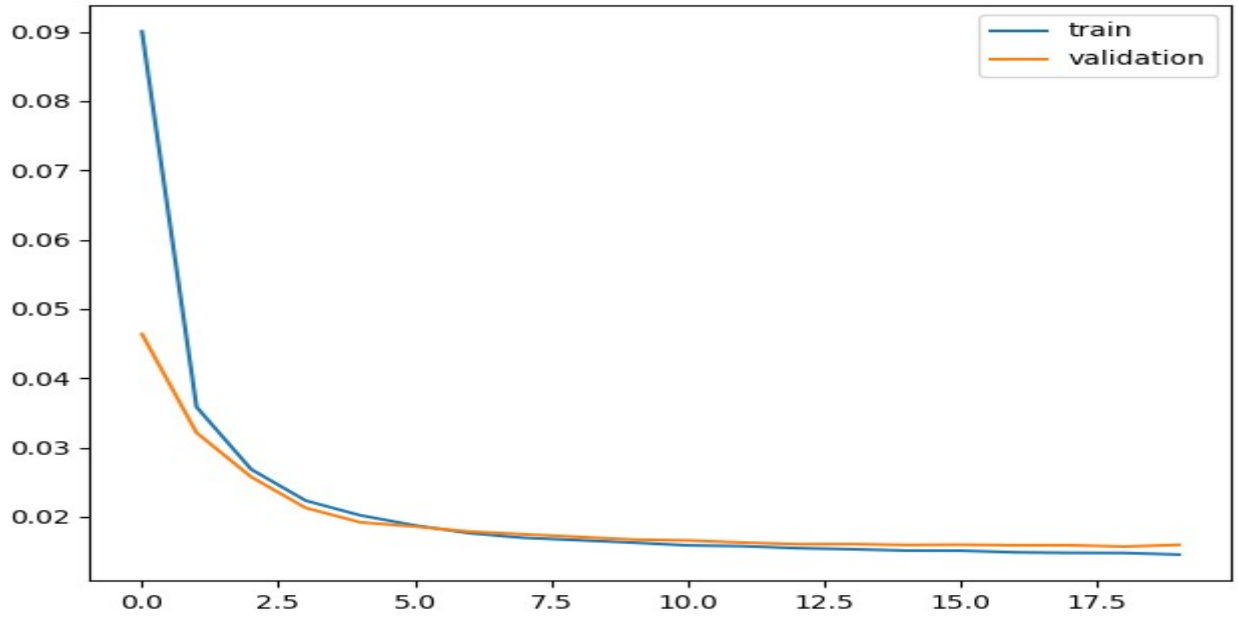


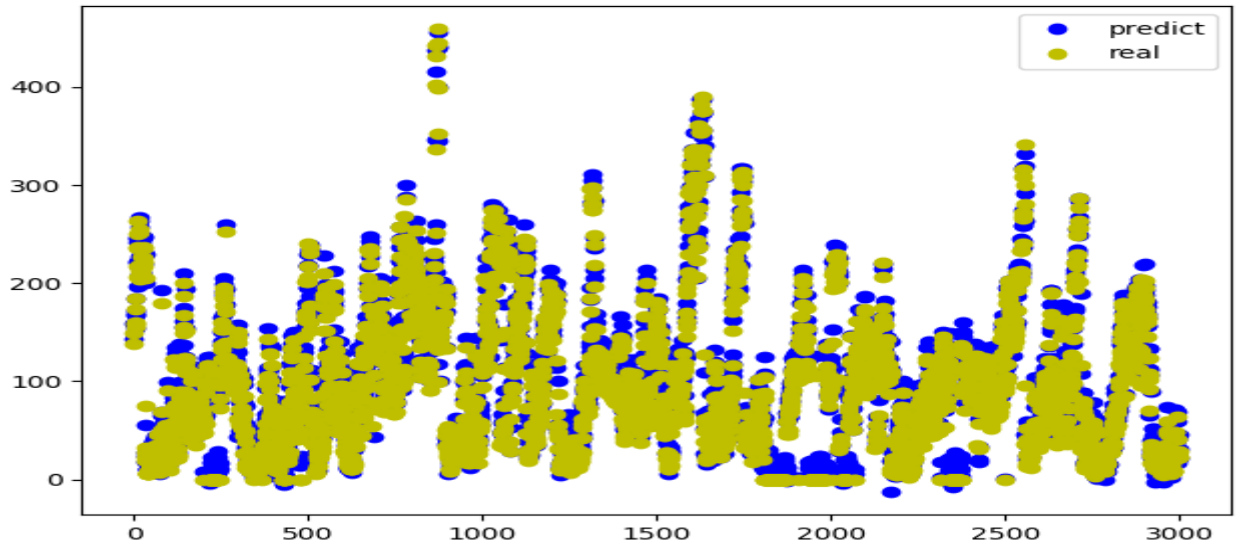
RNN, ADAgrad, MAE (15

```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='adagrad', metrics=['accuracy'])
```

```
Epoch 20/20
 - 1s - loss: 0.0146 - accuracy: 0.0745 - val_loss: 0.0160 - val_accuracy: 0.1267
Elapsed Time for fitiing:  35.39950442314148
Test RMSE: 24.155
```

RNN_ADAgrad_MAE.txt



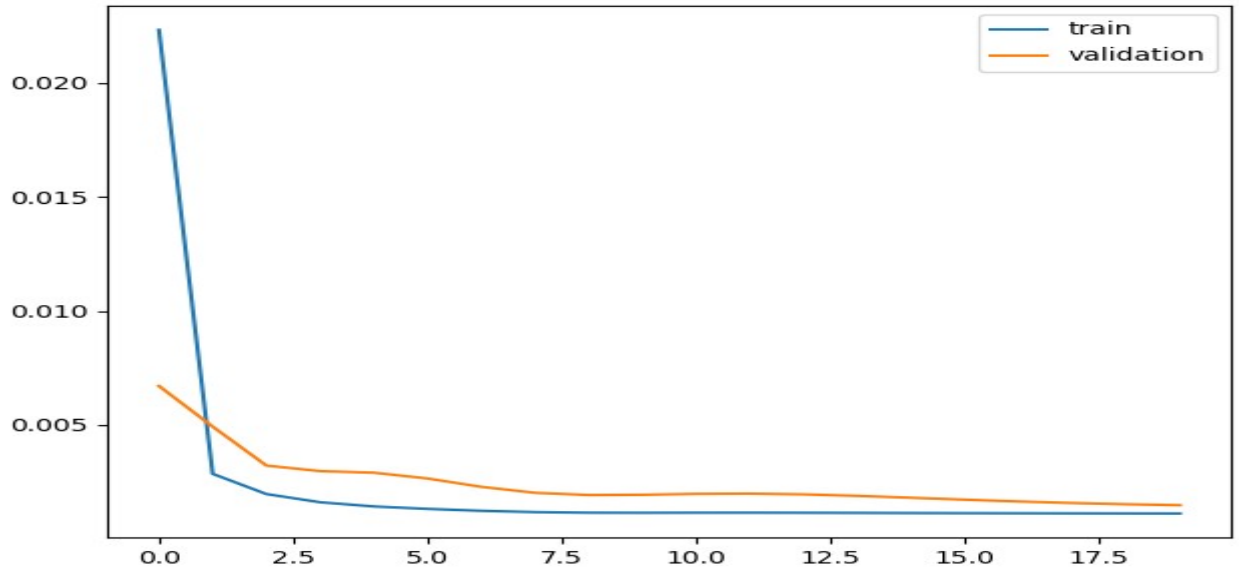


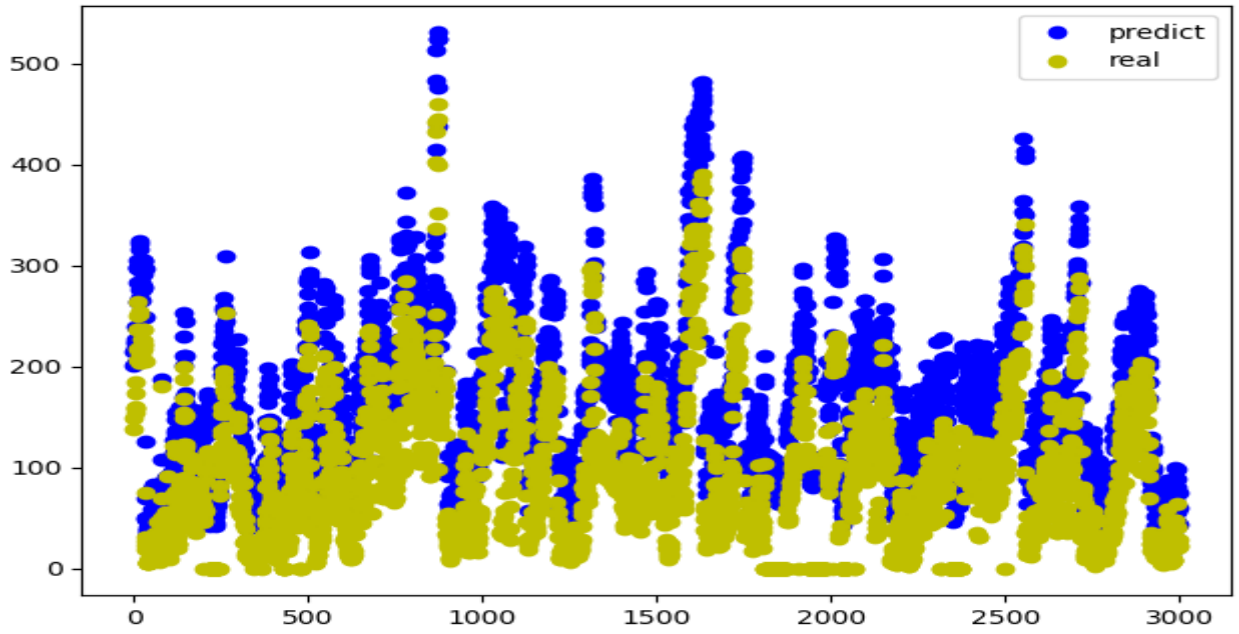
RNN, Adam, MSE (16

```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])
```

```
Epoch 20/20
 - 3s - loss: 0.0011 - accuracy: 0.0745 - val_loss: 0.0015 - val_accuracy: 0.1267
Elapsed Time for fitiing:  37.10362219810486
Test RMSE: 77.151
```

RNN_Adam_MSE.txt



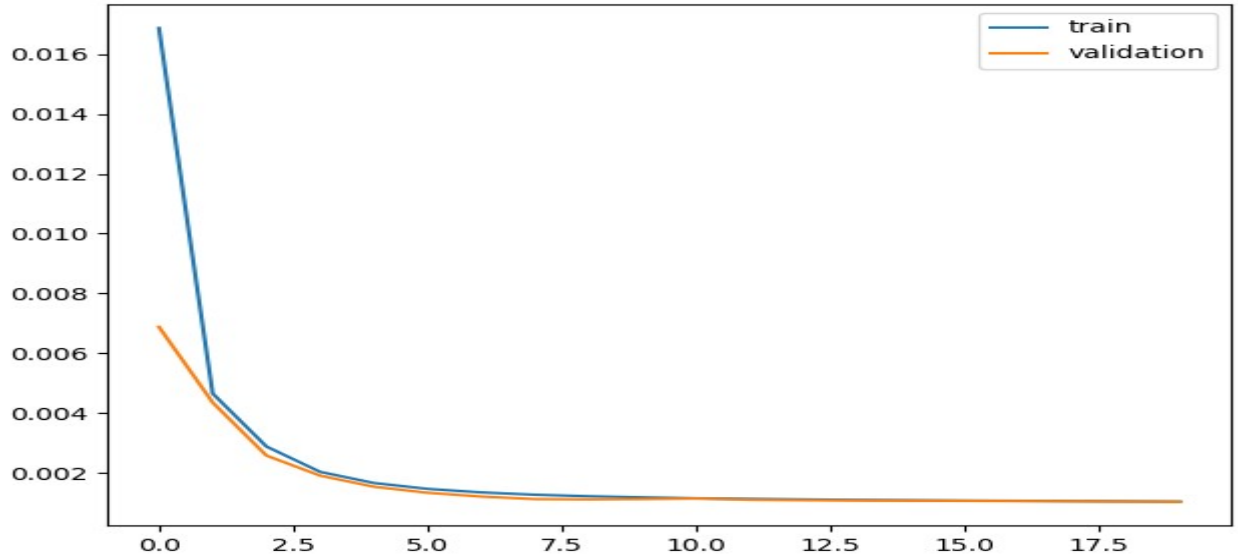


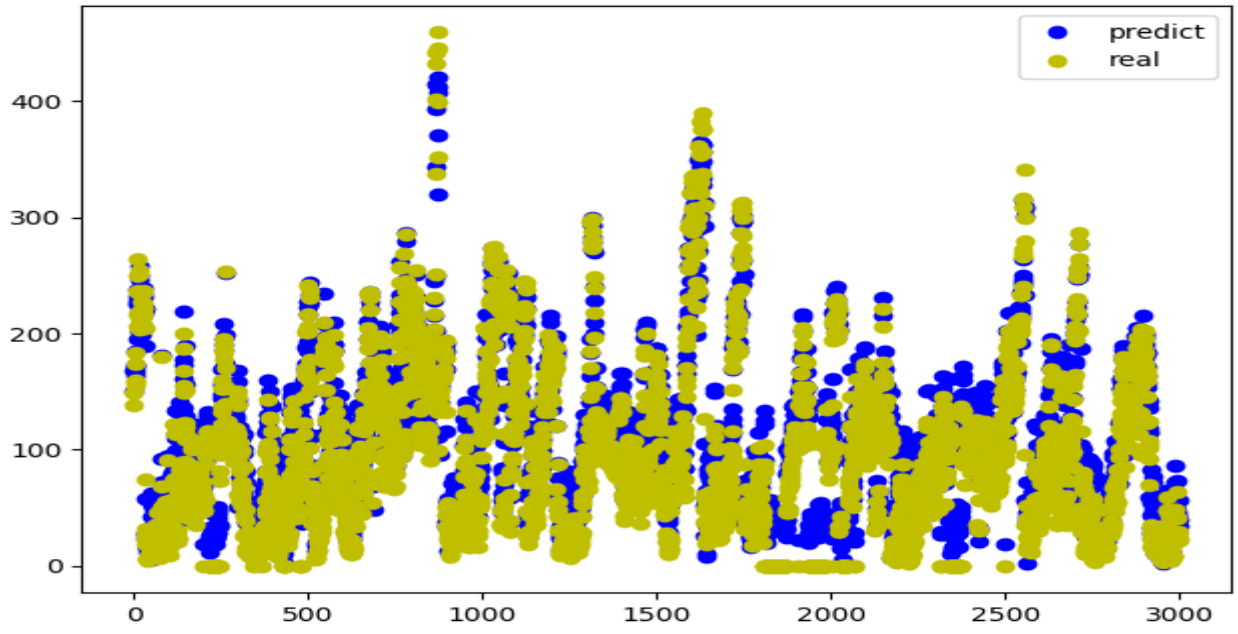
RNN, RMSProp, MSE (17

```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='rmsprop', metrics=['accuracy'])
```

```
Epoch 20/20
 - 2s - loss: 0.0010 - accuracy: 0.0745 - val_loss: 0.0010 - val_accuracy: 0.1267
Elapsed Time for fitiing:  37.236234188079834
Test RMSE: 29.541
```

RNN_RMSProp_MSE.txt



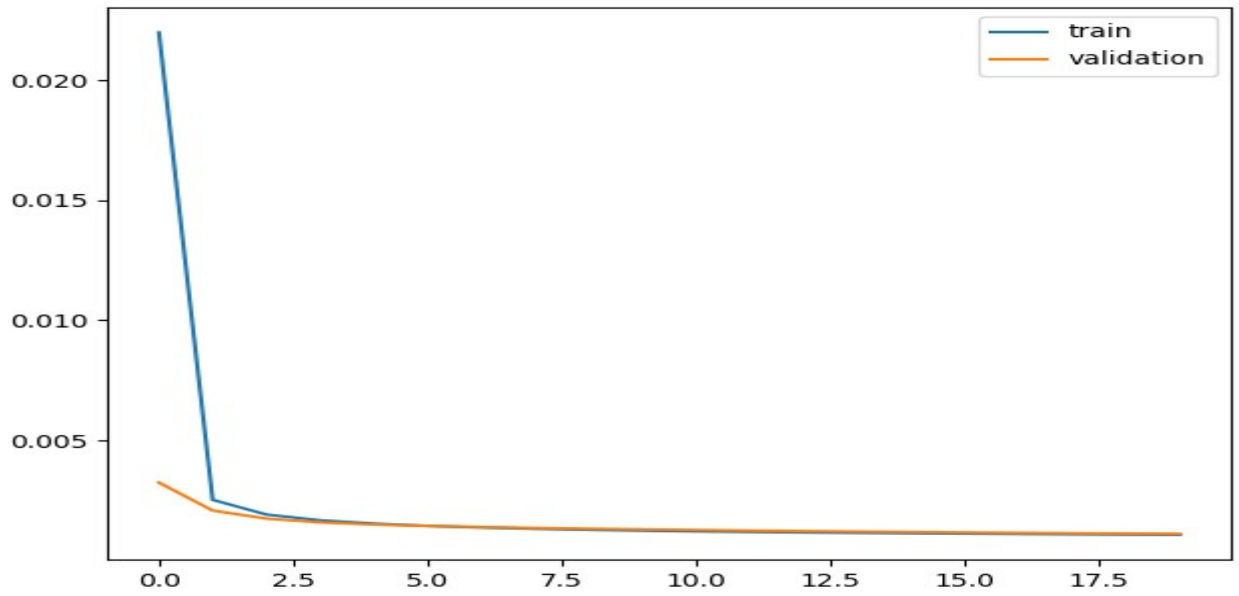


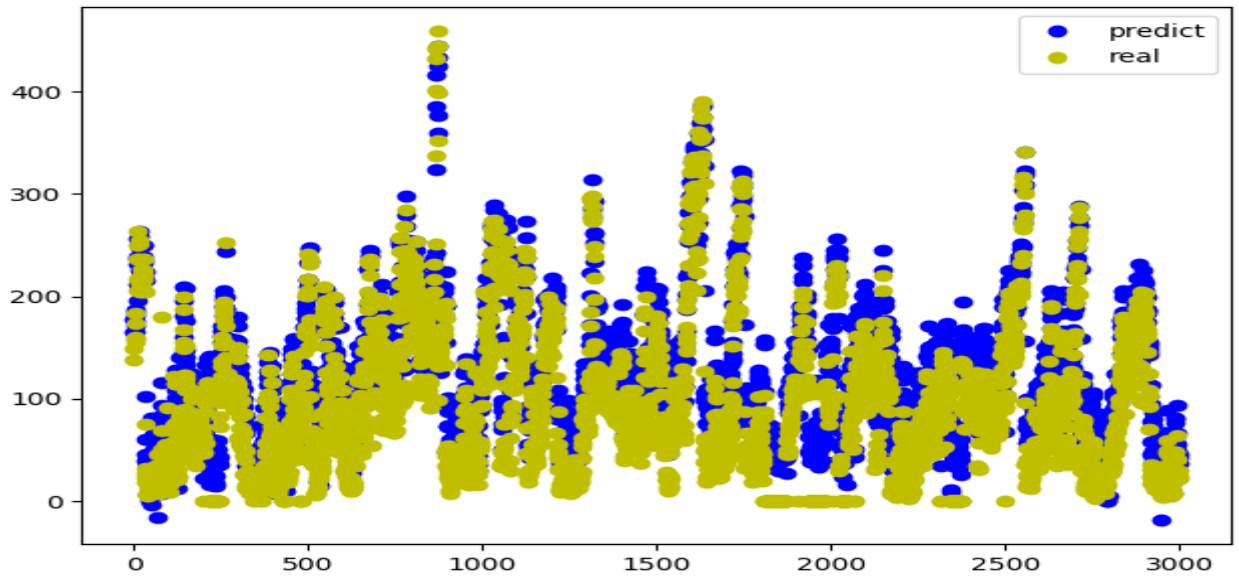
RNN, ADAgrad, MSE (18

```
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adagrad', metrics=['accuracy'])
```

```
Epoch 20/20
 - 2s - loss: 0.0011 - accuracy: 0.0746 - val_loss: 0.0011 - val_accuracy: 0.1267
Elapsed Time for fitiing:  37.11992001533508
Test RMSE: 41.002
```

RNN_ADAgrad_MSE.txt





همان‌طور که می‌بینیم بهترین دقت در پیشبینی آلودگی برای شبکه با LSTM , RMSProp, MAE بوده است
ولی در عین حال شبکه‌های SimpleRNN بیشترین سرعت را دارد که می‌دانیم درست است زیرا کمترین پارامتر برای آموزش را دارد
و در همهه حالت‌های شبکه‌ها تابع RMSProp بیشترین دقت را بدست آورده است

4) در این قسمت می‌خواهیم ۳ شبکه را با سری زمانی های متفاوت انجام دهیم:
ابتدا با سری زمانی هفتگی انجام می دهمیم: یم ساعت رندوم در هفته را کرفته و برای ۶ روز پیاپی در نظر می‌گیریم و روز ۷ ام را حدس میزنیم پس داریم:

```
#find one hour of day in week
random = np.random.randint(0, 23)
print(random)
dataset = dataset.iloc[random::24, :]
print(dataset)
```

```
n_hours = 6
n_features = 8
# frame as supervised learning
reframed = series_to_supervised(scaled, n_hours, 1)
print(reframed)
```

حال ۳ شبکه LSTM GRU Recurrent را با RMSProp و MAE آموزش می هیم:
در اینجا شبکه زیر را آموزش می‌دهیم و داریم:

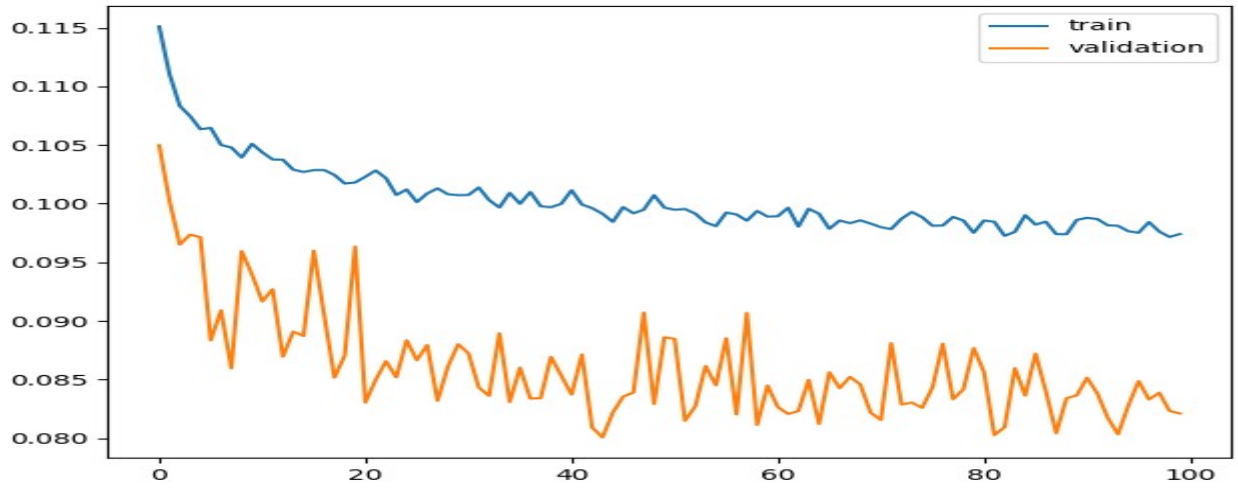
در اینجا چون تعداد داده‌ها بسیار کم می‌شود شبکه نمی‌تواند به دقت کافی برست ولی نزولی بودن لاس در طول آموزش مشهود است:
تعداد داده‌ها به صورت زیر است:

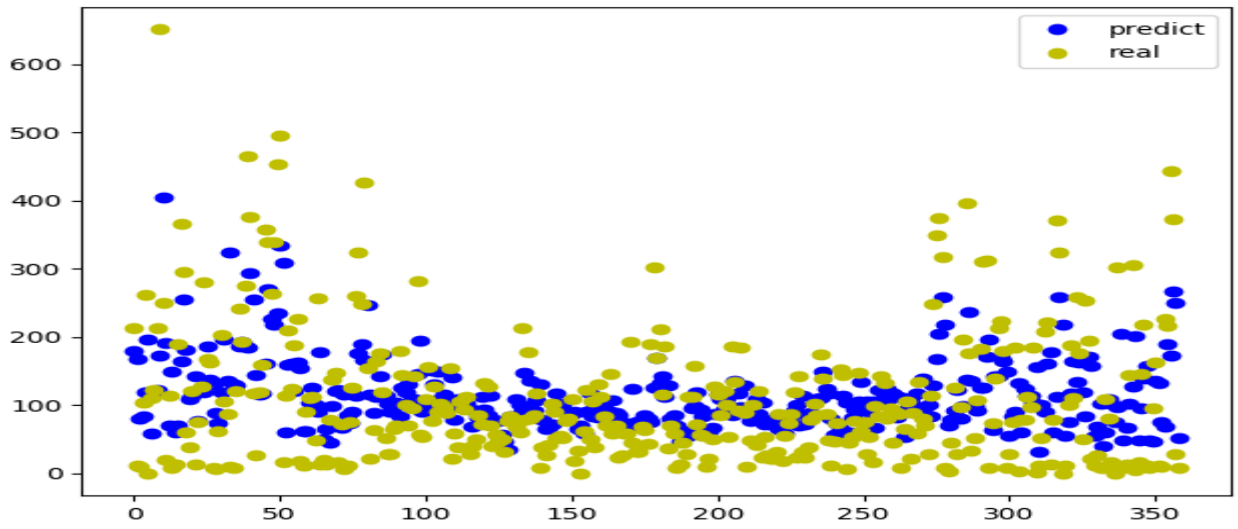
```
[1819 rows x 56 columns]
(1460, 48) 1460 (1460,)
(1460, 6, 8) (1460,) (359, 6, 8) (359,)
```

LSTM MAE RMSProp Dropout (1

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=100, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 100/100
 - 0s - loss: 0.0974 - mse: 0.0180 - val_loss: 0.0821 - val_mse: 0.0122
Elapsed Time for fitiing:  39.79531645774841
Test RMSE: 89.571
```

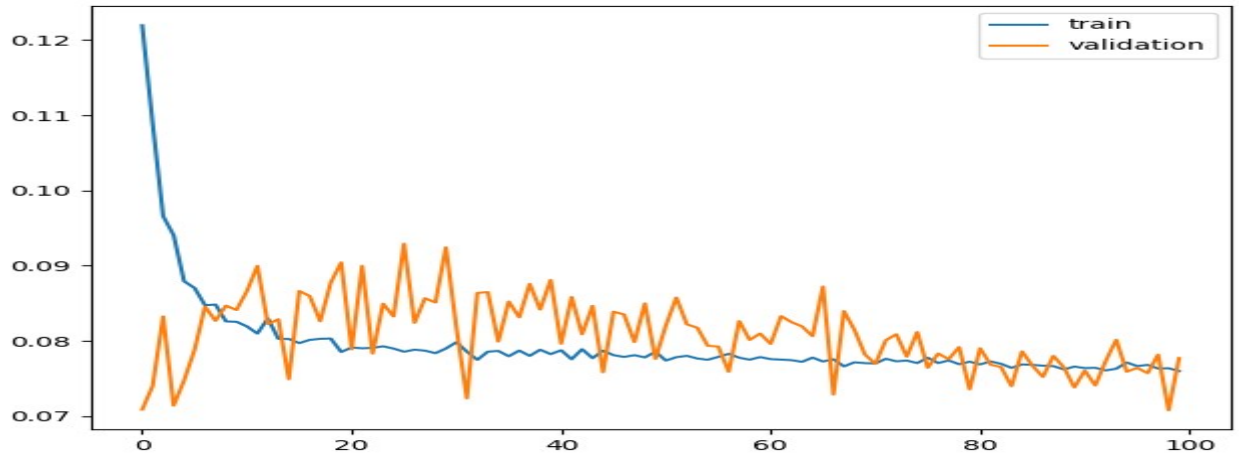


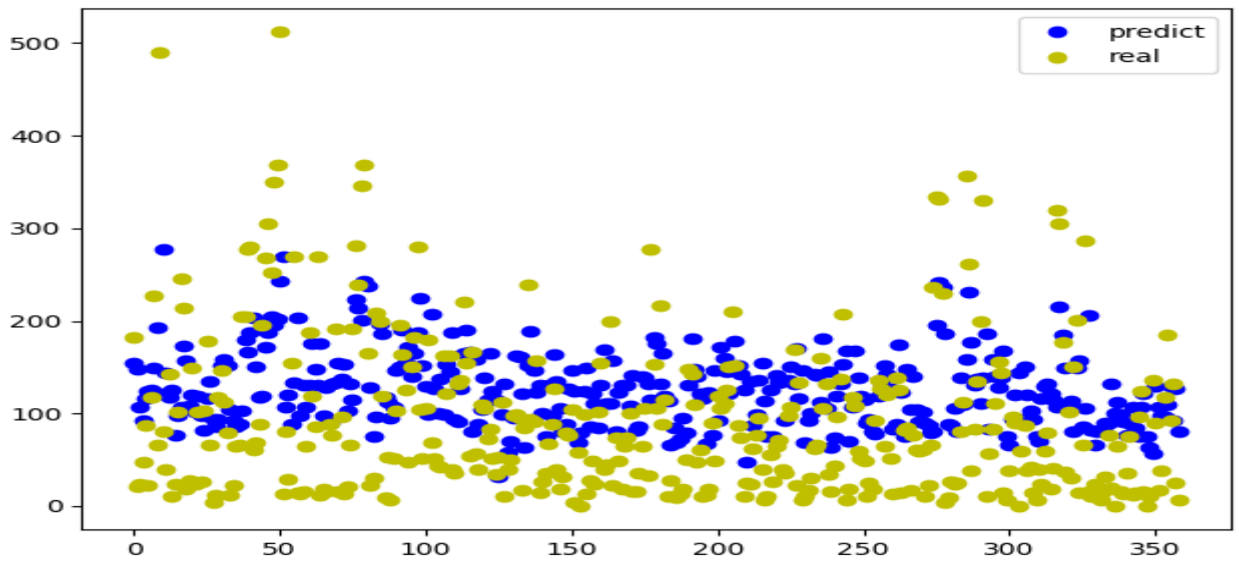


GRU RMSProp MAE Dropout (2

```
# design network
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=100, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 100/100
 - 1s - loss: 0.0760 - mse: 0.0116 - val_loss: 0.0777 - val_mse: 0.0091
Elapsed Time for fitiing:  52.19652795791626
Test RMSE: 84.776
```

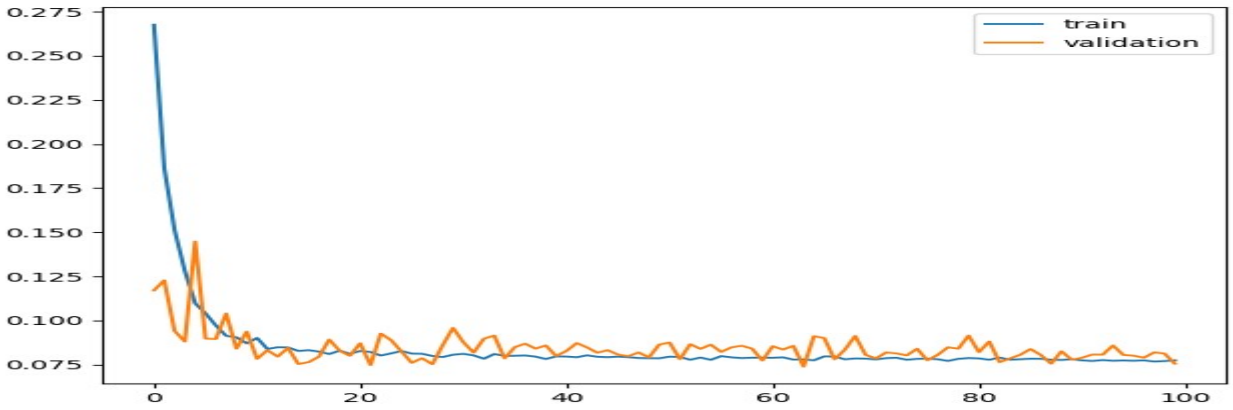


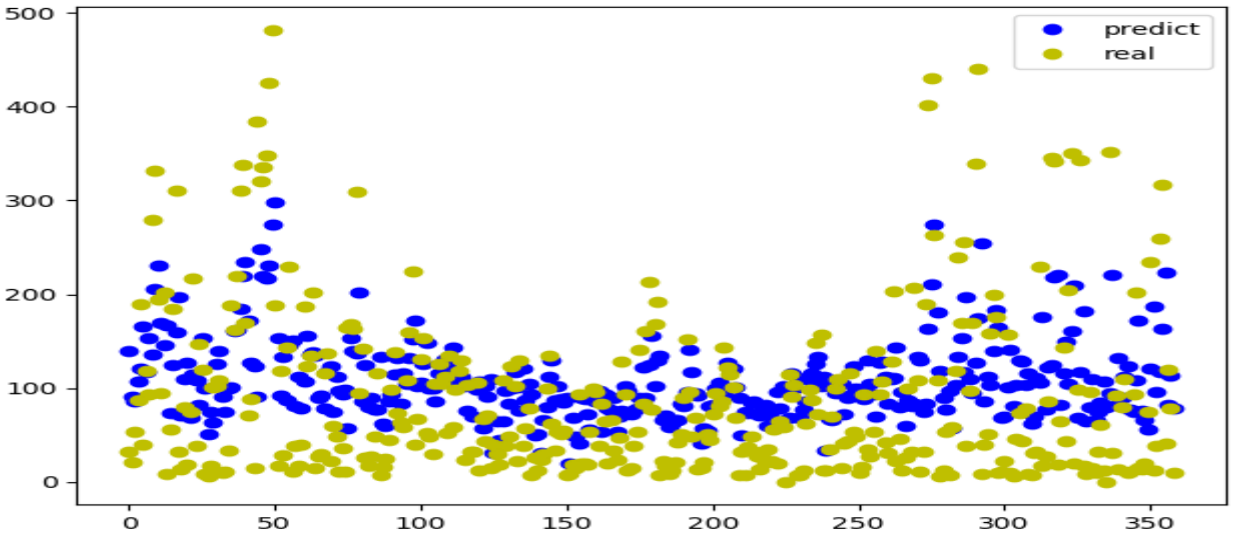


Recurrent RMSProp MAE Dropout (3

```
# design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=100, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 100/100
 - 0s - loss: 0.0775 - mse: 0.0122 - val_loss: 0.0755 - val_mse: 0.0095
Elapsed Time for fitiing:  25.207951545715332
Test RMSE: 82.097
```





حال برای همین ۳ شبکه به جای هفتگی داده‌ها را ماهانه جدا می‌کنیم در این صورت تعداد داده‌ها کمتر هم می‌شود که دقت را پایین می‌آورد ولی همچنن نزولی بودن آن معلوم است:

```
random = np.random.randint(0, 23)
print(random)
random_day = np.random.randint(1, 31)
print(random_day)
dataset = dataset.iloc[random::24, :]
print(dataset)
dataset = dataset.iloc[random_day::31, :]
```

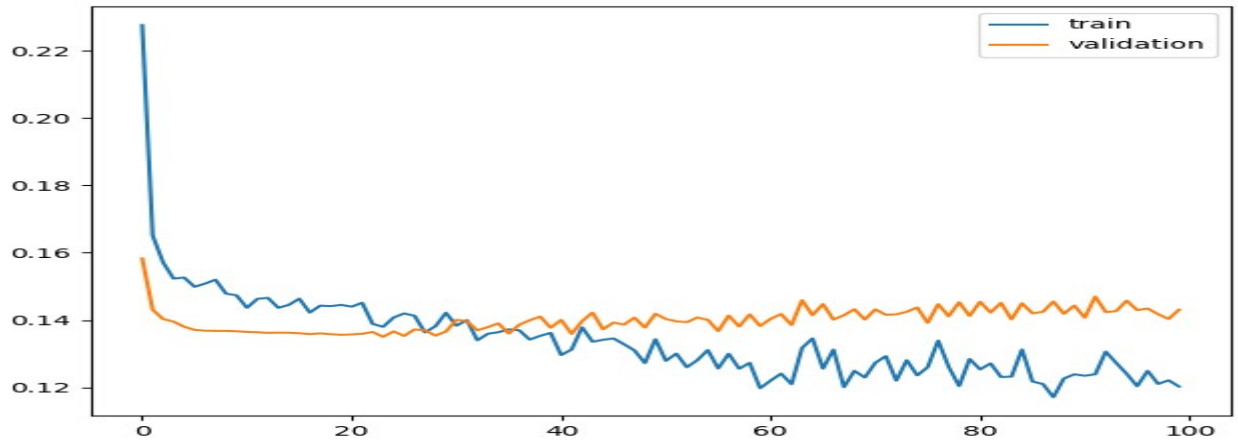
	pollution	dew	temp	press	wnd_dir	wnd_spd	snow	rain
date								
2010-01-02 07:00:00	124.0	-7	-5.0	1024.0	SE	10.72	0	0
2010-01-03 07:00:00	86.0	-10	-9.0	1024.0	SE	84.92	11	0
2010-01-04 07:00:00	29.0	-21	-13.0	1027.0	NW	73.75	0	0
2010-01-05 07:00:00	27.0	-27	-16.0	1034.0	NE	13.86	0	0
2010-01-06 07:00:00	25.0	-26	-15.0	1034.0	NE	33.52	0	0

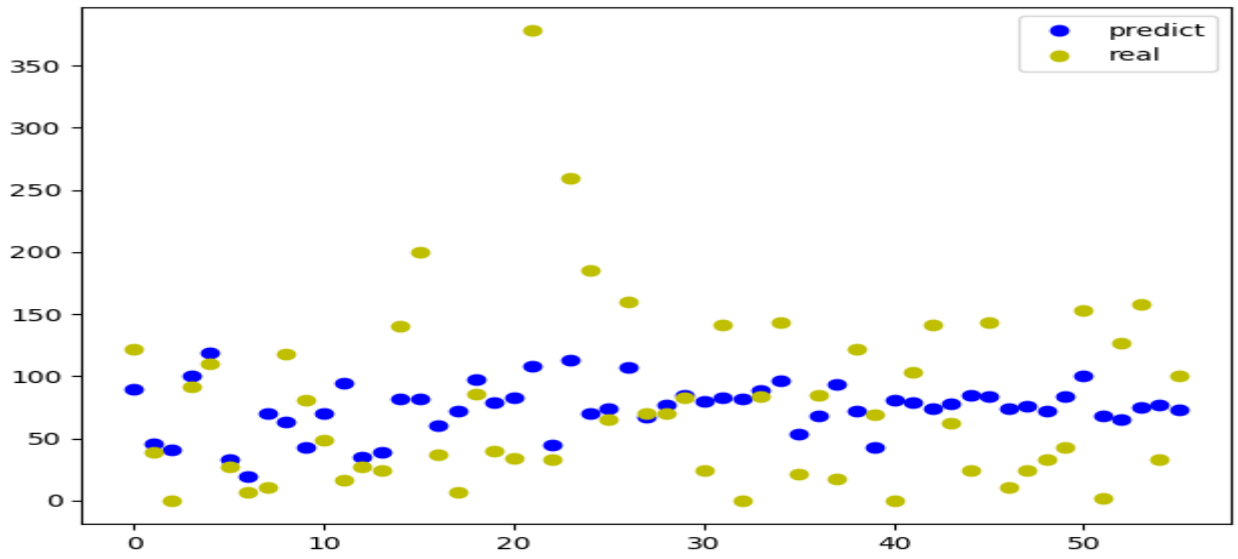
و ۳ هفته ی متوالی را برای پیش‌بینی هفته ۴ انتخاب می کنیم:
و در اینجا چون داده‌های ما کم است از همان داده‌های آموزش برای تست نیز استفاده می‌کنیم پس داریم:

LSTM RMSProp MAE Dropout (1

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=100, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 100/100
 - 0s - loss: 0.1203 - mse: 0.0331 - val_loss: 0.1431 - val_mse: 0.0219
Elapsed Time for fitiing:  3.7606234550476074
Test RMSE: 64.057
```

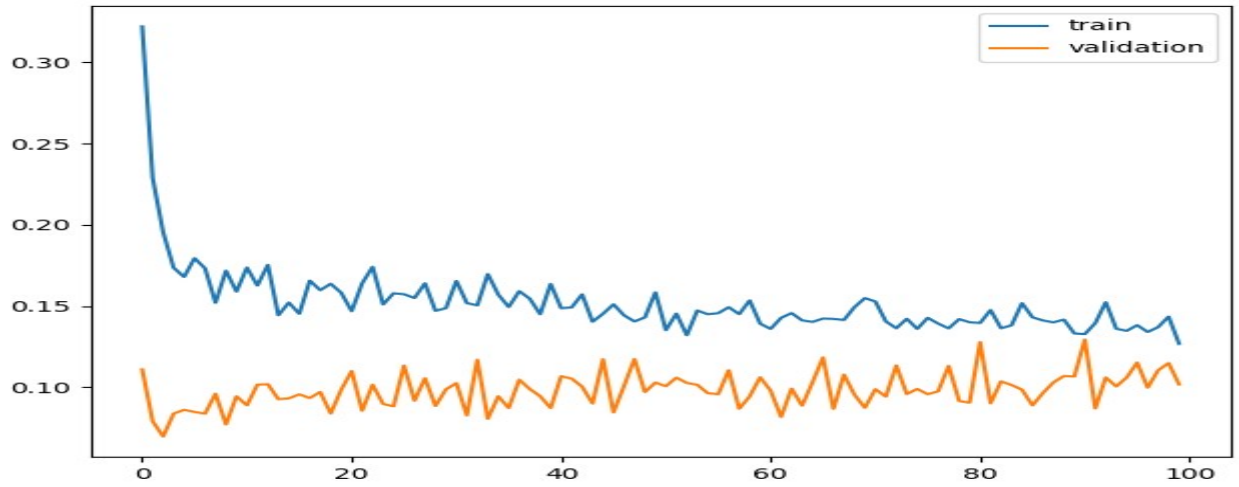


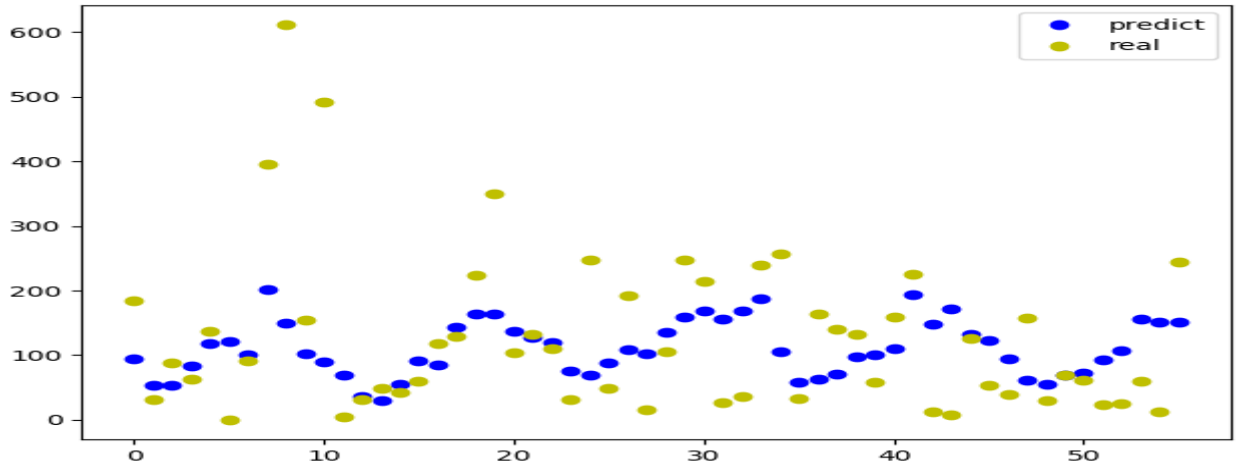


GRU RMSProp MAE Dropout (2

```
# design network
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=100, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 100/100
 - 0s - loss: 0.1509 - mse: 0.0608 - val_loss: 0.1892 - val_mse: 0.0485
Elapsed Time for fitiing:  4.1118323802948
Test RMSE: 101.717
```

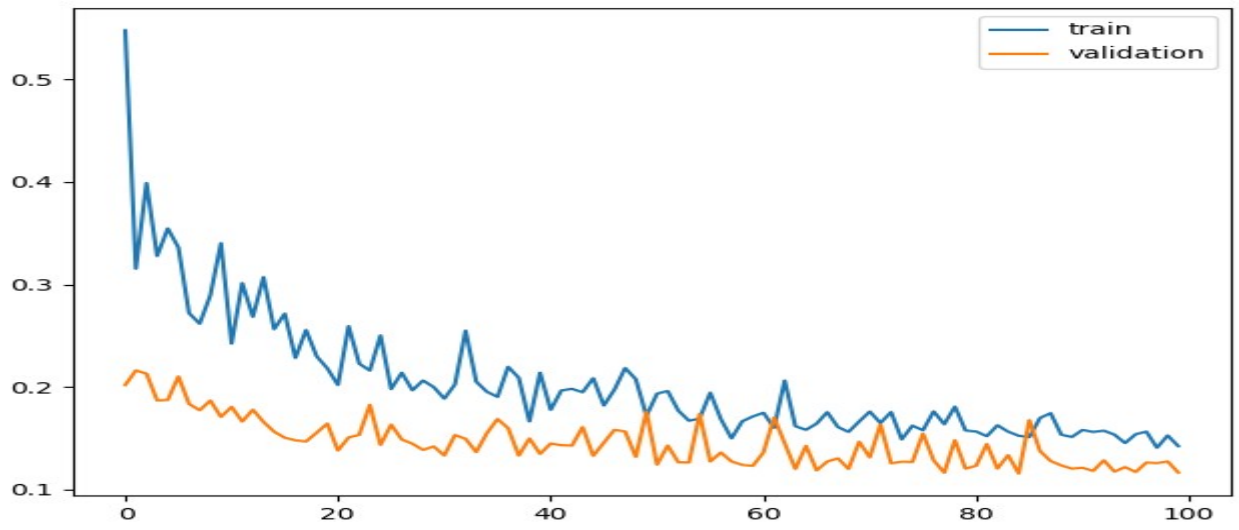


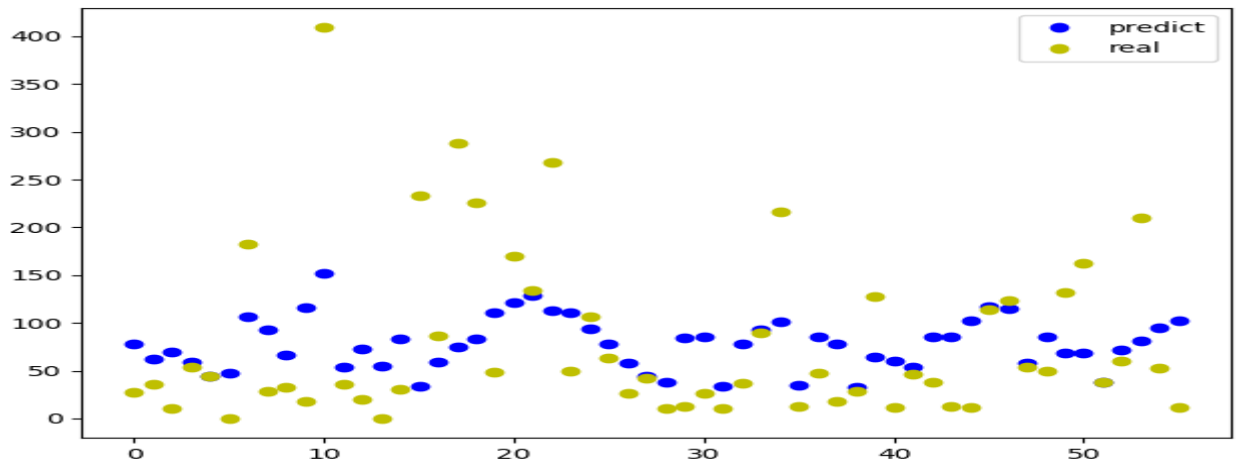


RNN RMSProp MAE Dropout (3

```
# design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=100, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 100/100
 - 0s - loss: 0.1647 - mse: 0.0445 - val_loss: 0.1787 - val_mse: 0.0464
Elapsed Time for fitiing:  2.669114828109741
Test RMSE: 81.537
```





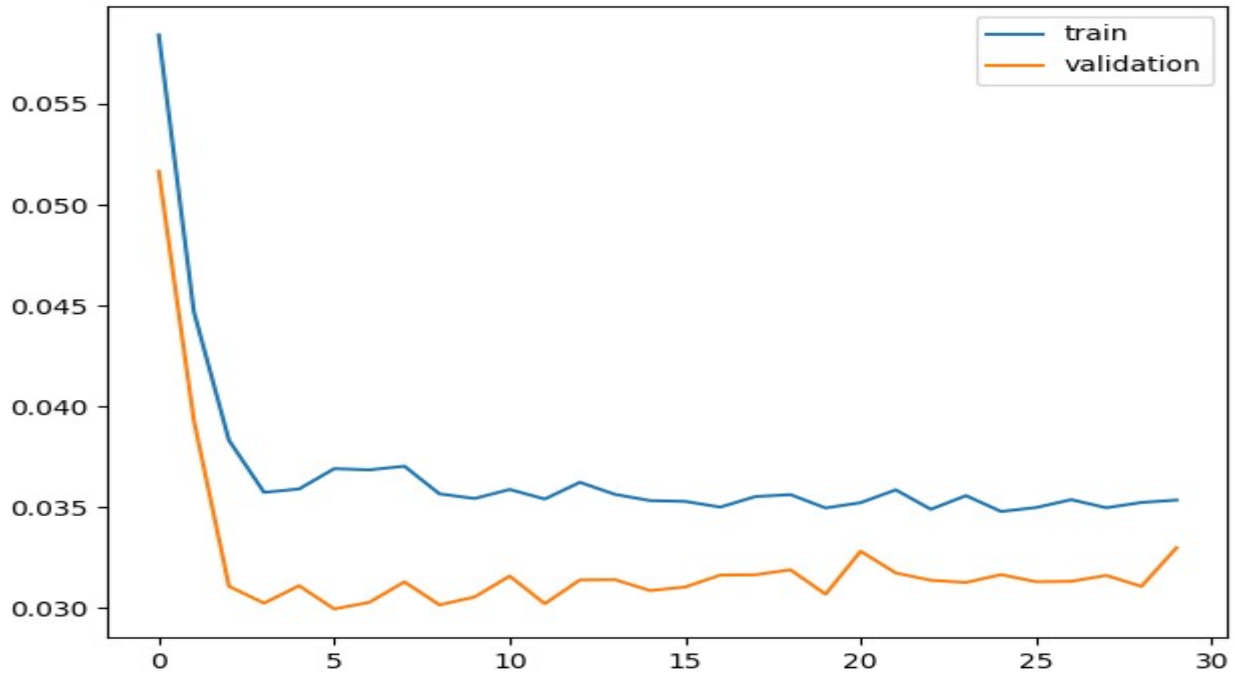
همان‌طور که میبینیم در ۲ سری شبکه فوق و سری اول یعنی به ترتیب ساعت‌های یک روز . یه ساعت در هفته برای ۶ روز و ۱ ساعت و روز در ماه برای ۳ هفته انجام دادیم
و دیدیم در تقریباً همه ی آن‌ها شبکه LSTM بیشترین دقت یعنی کمترین لاس را دارا بوده است
و برای هر حالت مقدار خطا از واقعیت نمودار لاس و مقادیر واقعی و حدس زده شده را رسم کرده ایم

5) برای پیدا کردن تأثیر dropout روی شبکه یک شبکه را امتحان می‌کنیم و خواهیم دید که:
این کار را روی شبکه با LSTM MAE RMSProp انجام دادیم که بهترین نتیجه را از قبل گرفته بود و کی بینیم

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2]), dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
```

```
Epoch 30/30
 - 1s - loss: 0.0354 - mse: 0.0046 - val_loss: 0.0330 - val_mse: 0.0022
Elapsed Time for fitiing:  34.759607553482056
Test RMSE: 36.910
```

و برای نمودار ها داریم:

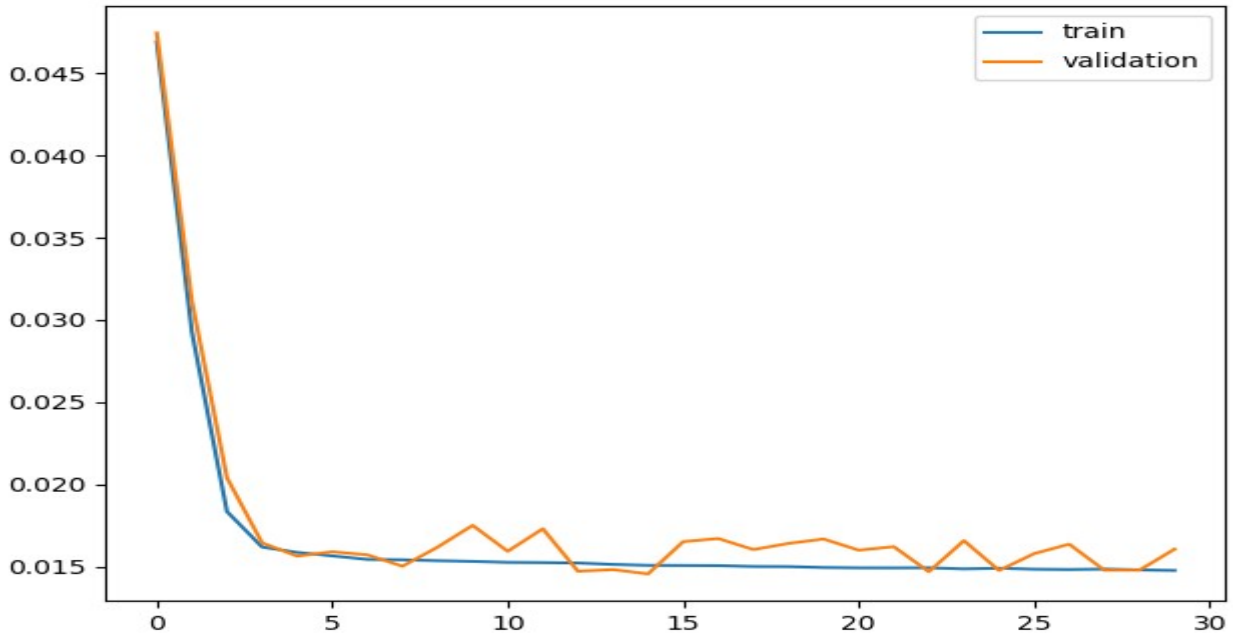


که همان‌طور که میبینیم اگر دقت کمی پایین‌تر رفته ولی اگر تعداد epoch را بیشتر می‌کردیم دو نمودار نزدیک‌تر می‌شدند و همچنین می‌بینیم که نمودار در هیچ حالتی overfit نمی‌شود چرا که با لایه ی dropout این اتفاق نمی‌افتد

حال اگر همین شبکه بالا را با recurrent dropout درست کنیم داریم:

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2]), recurrent_dropout=0.3))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
```

```
Epoch 30/30
 - 1s - loss: 0.0148 - mse: 9.1702e-04 - val_loss: 0.0161 - val_mse: 8.1388e-04
Elapsed Time for fitiing:  33.141114711761475
Test RMSE: 25.990
```



می‌بینیم دقت بسیار بهتر شده است چرا که این dropout از لایه‌های خو LSTM تأثیر می‌پذیرد و برای داده‌های تست بهتر عمل می‌کند و جلوی overfit را می‌گیرد

که دیدم recurrent dropout از dropout بهتر عمل کرده است

(6

7) در این قسمت می‌خواهیم ۲ ستون را که بیشترین ارتباط با خروجی ما یعنی مقدار آلودگی دارد را پیدا کنیم

برای پیدا کردن بیشترین آلودگی می‌توانیم با یکی از دو روش زیر استفاده کنیم:
۱) می تونیم با استفاده از pca ابعاد داده‌ها را کم کنیم و ارتباط هر ستون داده با خروجی را بیابیم

۲) می‌توانیم از کورولیشن بین داده استفاده کنیم به این معنی که ستونی که بیشترین کورولیشن را با خروجی دارد درواقع بیشترین ارتباط و هم بستگی را با آن دارد

در اینجا از روش ۲ استفاده می‌کنیم و مانند زیر عمل می کنیم:

```
def find_corolation(df):
    import pandas as pd
    pd.set_option('display.max_rows', 500)
    pd.set_option('display.max_columns', 500)
    pd.set_option('display.width', 1000)
    df = DataFrame(df)
    df['wnd_dir'] = df['wnd_dir'].astype('category').cat.codes
    print(df.corr())
```

که در آن کورولیشن بین ستون‌های مختلف را محاسبه می‌کنیم و در خروجی داریم:

	pollution	dew	temp	press	wnd_dir	wnd_spd	snow	rain
pollution	1.000000	0.157585	-0.090798	-0.045544	0.187459	-0.234362	0.022226	-0.049045
dew	0.157585	1.000000	0.824432	-0.778737	0.232960	-0.296720	-0.034484	0.125053
temp	-0.090798	0.824432	1.000000	-0.827205	0.175626	-0.154902	-0.092726	0.049037
press	-0.045544	-0.778737	-0.827205	1.000000	-0.168986	0.185380	0.069031	-0.079840
wnd_dir	0.187459	0.232960	0.175626	-0.168986	1.000000	-0.200031	0.010356	-0.048323
wnd_spd	-0.234362	-0.296720	-0.154902	0.185380	-0.200031	1.000000	0.021876	-0.010137
snow	0.022226	-0.034484	-0.092726	0.069031	0.010356	0.021876	1.000000	-0.009553
rain	-0.049045	0.125053	0.049037	-0.079840	-0.048323	-0.010137	-0.009553	1.000000

همان‌طور که می‌بینیم ستون‌های dev و wnd_dir بیشترین ارتباط را با خروجی دارند که در اینجا این ۲ ستون را انتخاب می‌کنیم

8) در اینجا ۲ ستون انتخاب شده را با خروجی برداشته و برای ۳ شبکه آموزش می‌دهیم
در اینجا از تمام داده‌های فایل استفاده کرده‌ایم و مانند قسمت اول از ۱۱ ساعت قبل استفاده کرده و ساعت ۱۲ را پیش‌بینی می‌کنیم

```
# Feature Selection : select 2
dataset = dataset[['pollution', 'dew', 'wnd_dir']]
```

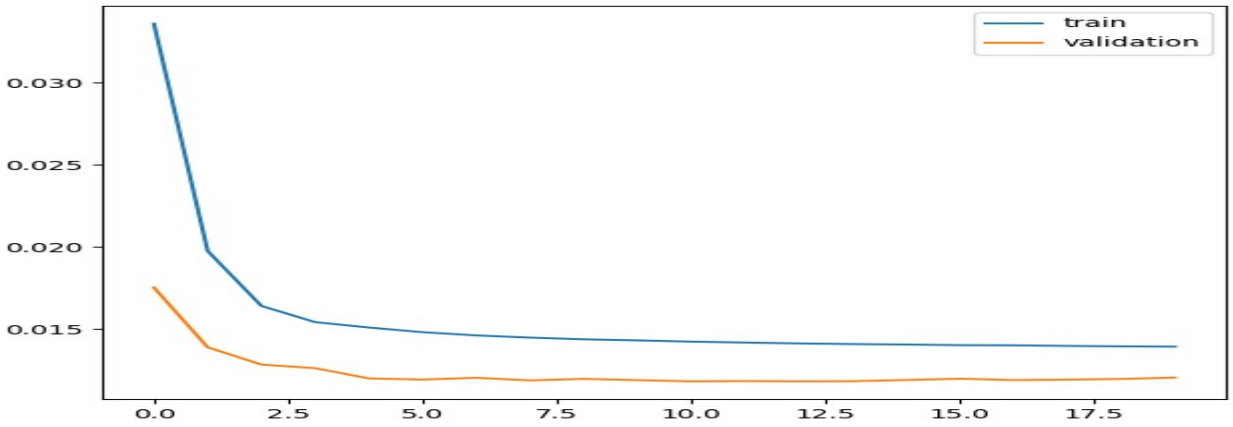
```
n_hours = 11
n_features = 3
```

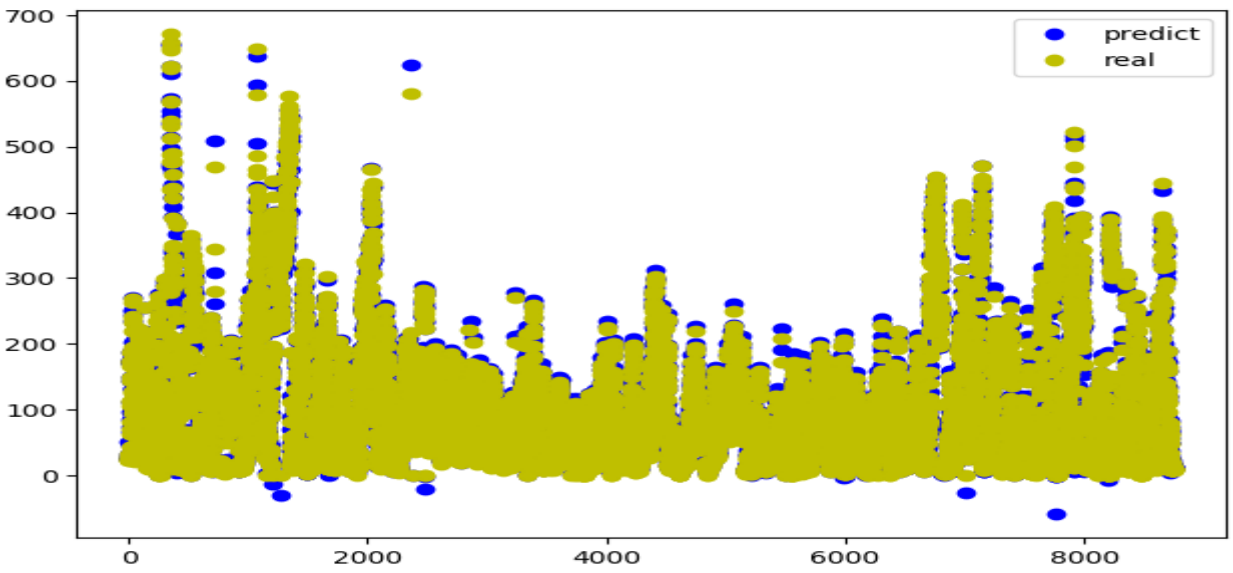
حال می‌توانیم ۳ شبکه را با این داده‌ها آموزش دهیم پس داریم:

LSTM RMSProp MAE (۱

```
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 20/20
 - 10s - loss: 0.0139 - mse: 8.8372e-04 - val_loss: 0.0120 - val_mse: 5.0008e-04
Elapsed Time for fitiing:  187.4927203655243
Test RMSE: 24.120
```

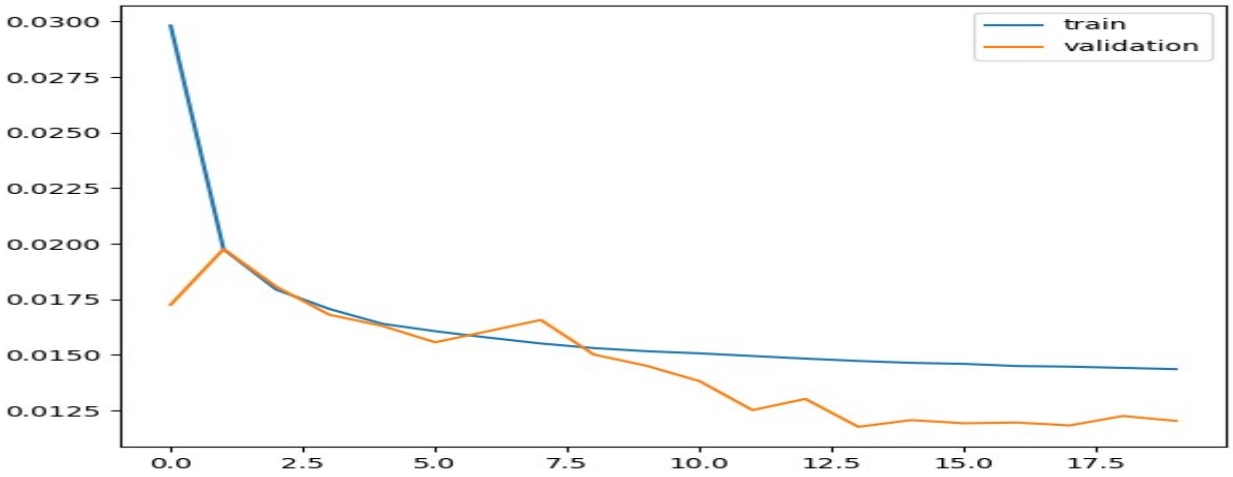


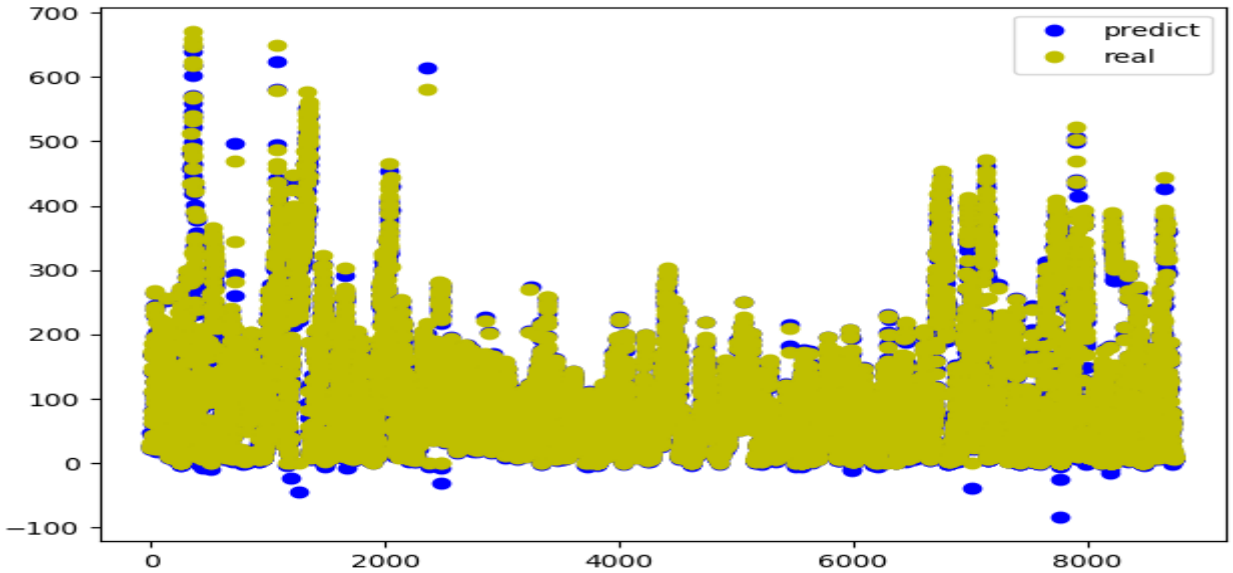


GRU RMSprop MAE (2

```
# design network
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
```

```
Epoch 20/20
 - 12s - loss: 0.0144 - mse: 8.9577e-04 - val_loss: 0.0120 - val_mse: 4.9411e-04
Elapsed Time for fitiing:  234.440105676651
Test RMSE: 24.103
```

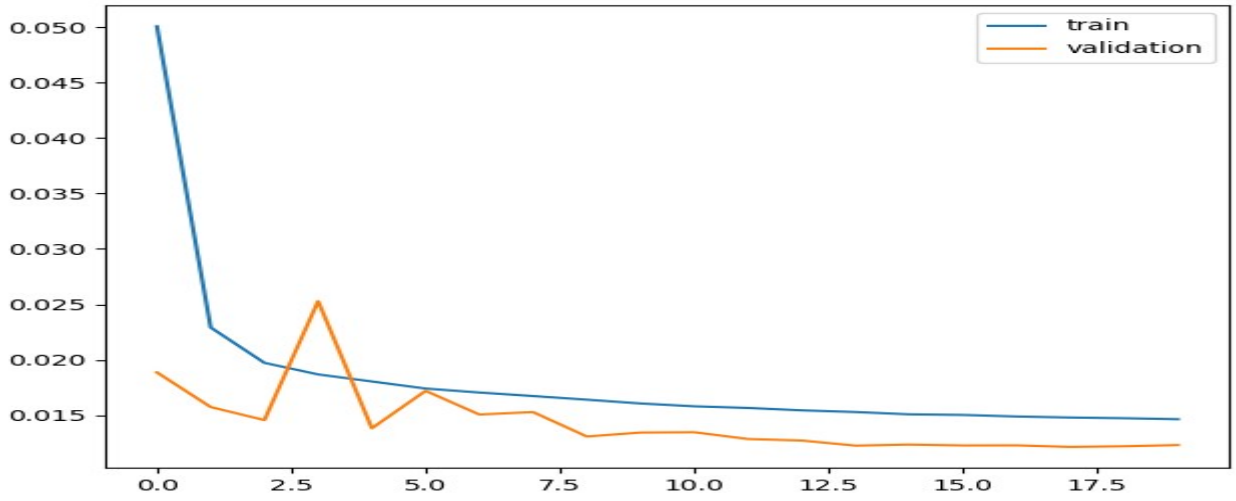


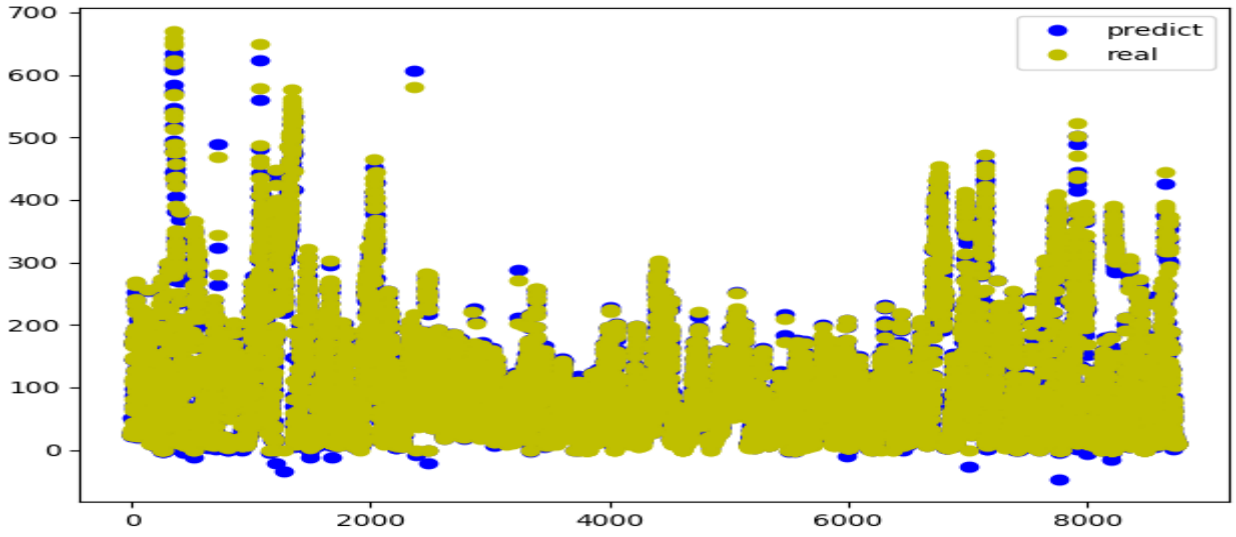


RNN RMSProp MAE (3

```
# design network
model = Sequential()
model.add(SimpleRNN(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
```

```
Epoch 20/20
 - 4s - loss: 0.0146 - mse: 8.8956e-04 - val_loss: 0.0123 - val_mse: 5.0870e-04
Elapsed Time for fitiing:  97.21206617355347
Test RMSE: 24.266
```





دقت ۳ شبکه تقریباً برابر است ولی برای حالت کلی LSTM توانسته بهتر عمل کند ولی زمان بیشتری برای آموزش نیاز داشته است چرا که تعداد پارامتر های آن بیشتر است

سؤال ۲)

ابتدا برای هر ستون از داده‌ها ۲۰ درصد از آن‌ها را حذف می‌کنیم مانند زیر:

```
# MAke 20 percent of each column miss
for key, value in dataset.iteritems():
    dataset.loc[dataset[key].sample(frac=.2).index, key] = np.nan

print(dataset)
print(dataset['dew'].isna().sum())
```

که مانند زیر خروجی می‌شود

```
                     pollution   dew  temp   press wnd_dir  wnd_spd  snow  rain
date
2010-01-02 00:00:00      129.0   NaN  -4.0     NaN      SE     1.79   0.0   0.0
2010-01-02 01:00:00      148.0 -15.0  -4.0  1020.0      SE     2.68   0.0   0.0
2010-01-02 02:00:00      159.0   NaN  -5.0  1021.0      SE     3.57   0.0   0.0
2010-01-02 03:00:00        NaN  -7.0  -5.0  1022.0      SE     5.36   1.0   0.0
2010-01-02 04:00:00      138.0   NaN  -5.0  1022.0     NaN     6.25   NaN   0.0
...                        ...   ...   ...     ...     ...      ...   ...   ...
2014-12-31 19:00:00        8.0   NaN  -2.0     NaN     NaN   231.97   0.0   NaN
2014-12-31 20:00:00        NaN -22.0  -3.0  1034.0     NaN   237.78   0.0   0.0
2014-12-31 21:00:00        NaN -22.0  -3.0  1034.0      NW   242.70   0.0   0.0
2014-12-31 22:00:00        8.0 -22.0  -4.0     NaN      NW   246.72   0.0   NaN
2014-12-31 23:00:00       12.0 -21.0  -3.0  1034.0      NW   249.85   0.0   NaN

[43800 rows x 8 columns]
8760
```

۳) حال برای رفع این نقصان داده‌ها باید یک روش را انتخاب کنیم برای این کار ابتدا به بررسی ۳ روش زیر می پردازیم:

روش ۱) در این روش می‌توانیم مقادیر را با میانه ی همان ستون یا میانگین آن جایگزین کنیم
این کار راحت و سریع است و برای دیتا ست های عددی کوچک می‌تواند خوب عمل کرد ولی بدی آن این است که هم دقیق نیست و هم با ارتباط به ستون‌های دیگر نیست و فقط به ستون خودش مربوط می‌شود و برای داده‌های کتگوری نمی‌توان از آن استفاده کرد

روش ۲) جایگزین کردن داده‌ها با داده‌ای که بیشترین تکرار را دارد و یا جایگزین کردن آن با ۰ یا یک عدد ثابت
این روش برای داده‌های کتگوری به خوبی کار می‌کند بر عکس روش قبل ولی می‌تواند نویز زیادی وارد داده‌ها کند و مانند روش قبل در ارتباط با ستون‌های دیگر نیست

روش ۳)استفاده از K_NN در این روش برای جایگزین کردن داده‌ها از N داده ی نزدیک به آن استفاده می‌شود و می‌تواند داده حذف شده را پیش‌بینی کند

در اینجا برای حل نقصان دادگان از روش میانگین استفاده می‌کنیم
برای این کار به جای هر یک از دادگان حذف شده مقدار میانگین آن ستون بعد از پیش پردازش را جایگزین می‌کنیم
برای این کار مانند زیر عمل می کنیم:

```
# Fill Missing Data
df = DataFrame(scaled)
df = df.fillna(df.mean())
print(df)
```

که می‌بینیم داده‌های حذف شده جایگزین شده‌اند

	0	1	2	...	5	6	7
0	0.096565	0.608401	0.523349	...	0.040481	0.002041	0.005205
1	0.152263	0.358209	0.250000	...	0.040481	0.002041	0.000000
2	0.163580	0.417910	0.233333	...	0.040481	0.000000	0.000000
3	0.096565	0.608401	0.523349	...	0.040481	0.002041	0.005205
4	0.141975	0.477612	0.523349	...	0.009912	0.076923	0.000000
...	...	...	...	...	...	...	...
43795	0.096565	0.238806	0.523349	...	0.395659	0.000000	0.000000
43796	0.010288	0.608401	0.523349	...	0.405588	0.000000	0.000000
43797	0.096565	0.253731	0.523349	...	0.040481	0.000000	0.000000
43798	0.096565	0.608401	0.523349	...	0.040481	0.002041	0.005205
43799	0.012346	0.608401	0.266667	...	0.426216	0.002041	0.000000

و مقادیر میانگین آن‌ها برابر بوده است با:

```
[43800 rows x 8 columns]
0    0.096565
1    0.608401
2    0.523349
3    0.462537
4    0.545120
5    0.040481
6    0.002041
7    0.005205
```

حال می‌توانیم خطای این مقادیر را با مقادیر اصلی که بوده‌اند با روش MSE پیدا کنیم و ببینیم که چقدر تفاوت مقادیر وجود دارد برای این کار مانند زیر عمل می کنیم:

```
print("pollution MSE: ", mean_squared_error(df.iloc[:, 0], df_new.iloc[:, 0]))
print("dew MSE: ", mean_squared_error(df.iloc[:, 1], df_new.iloc[:, 1]))
print("temp MSE: ", mean_squared_error(df.iloc[:, 2], df_new.iloc[:, 2]))
print("press MSE: ", mean_squared_error(df.iloc[:, 3], df_new.iloc[:, 3]))
print("wnd_dir: ", mean_squared_error(df.iloc[:, 4], df_new.iloc[:, 4]))
print("wnd_speed MSE: ", mean_squared_error(df.iloc[:, 5], df_new.iloc[:, 5]))
print("snow MSE: ", mean_squared_error(df.iloc[:, 6], df_new.iloc[:, 6]))
print("rain MSE: ", mean_squared_error(df.iloc[:, 7], df_new.iloc[:, 7]))
```

و در خروجی داریم:

```
pollution MSE:  0.0031058332
dew MSE:  0.016202146
temp MSE:  0.014445615
press MSE:  0.01252974
wnd_dir:  0.023768868
wnd_speed MSE:  0.0025056782
snow MSE:  0.00026352733
rain MSE:  0.00060844544
```

که همان خطای مقادیر بدست آمده با مقادیر اصلی است

بعد از هر با اجرا مقادیر ممکن است کمی تغییر کنند چرا که مقادیر حذف شده به صورت رندوم انتخاب شده بودند ولی در آخر مقادیر نزدیک به همین مقادیر می‌شوند

حال می‌خواهیم با دیتا ست بدست آمده مقادیر را حدس بزنیم در اینجا بازه زمانی را برابر با ۱۱ در نظر می‌گیریم تا مانند قسمت اول بتوانیم با هم مقایسه کنیم و از دیتا ست جدید برای اینکار استفاده می کنیم:

```
# specify the number of lag hours
n_hours = 11
n_features = 8
```

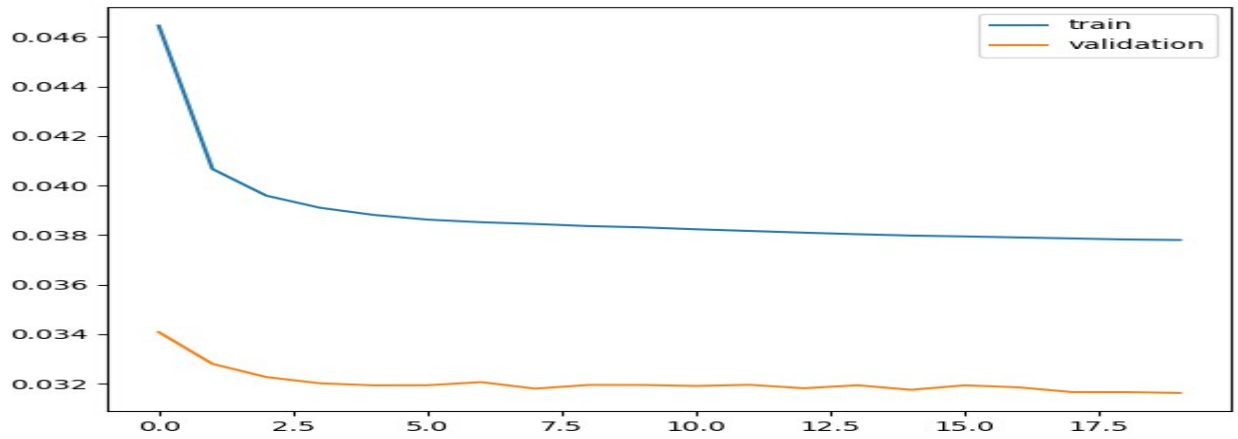
```
# frame as supervised learning
reframed = series_to_supervised(df.values, n_hours, 1)
print(reframed)
# split into train and test sets
values = reframed.values
split = int(len(dataset.index) * 0.8)
train = values[:split, :]
test = values[split:, :]

# split into input and outputs
n_obs = n_hours * n_features
train_X, train_y = train[:, :n_obs], train[:, -n_features]
test_X, test_y = test[:, :n_obs], test[:, -n_features]
print(train_X.shape, len(train_X), train_y.shape)
```

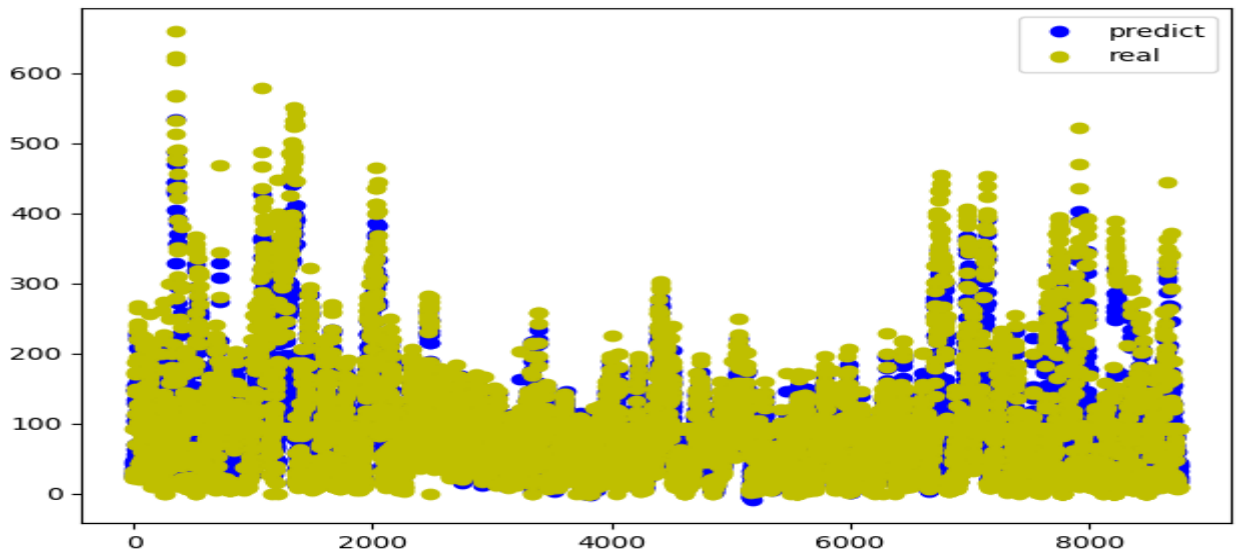
حال برای ۲ شبکه با LSTM و GRU مقادیر را اجرا می‌کنیم و داریم:

:LSTM RMSProp MAE

```
# design network
model = Sequential()
model.add(LSTM(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 20/20
 - 9s - loss: 0.0378 - mse: 0.0034 - val_loss: 0.0316 - val_mse: 0.0022
Elapsed Time for fitiing:  177.01649689674377
Test RMSE: 56.954
```

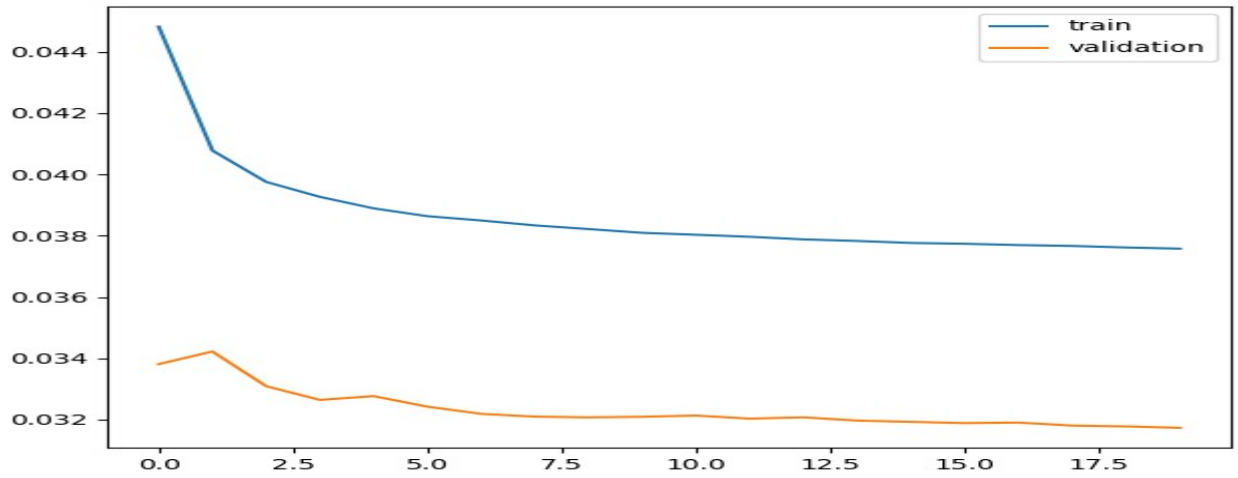


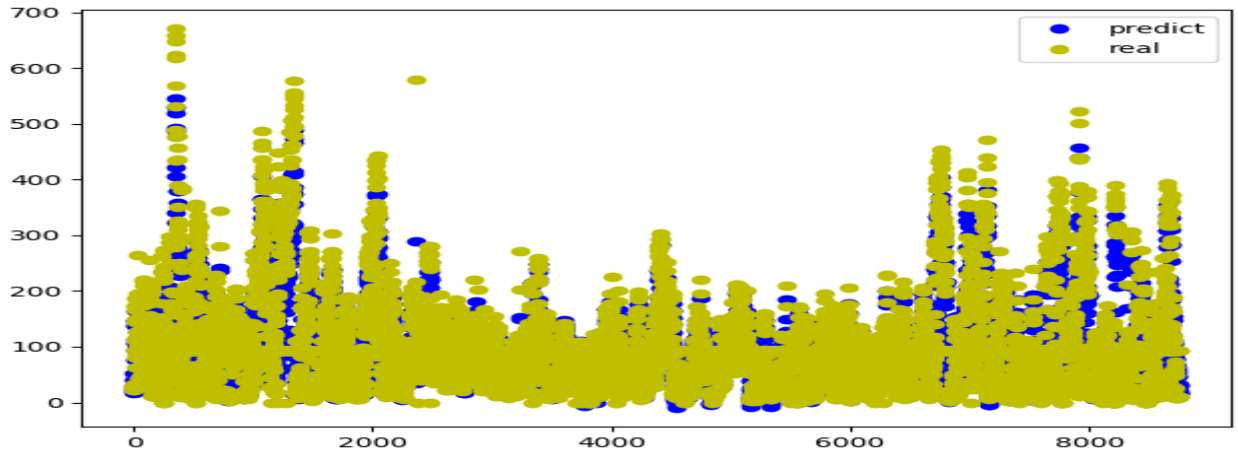


GRU RMSProp MAE

```
# design network
model = Sequential()
model.add(GRU(50, input_shape=(train_X.shape[1], train_X.shape[2])))
model.add(Dense(1))
model.compile(loss='mae', optimizer='rmsprop', metrics=['mse'])
# fit network
start = time.time()
history = model.fit(train_X, train_y, epochs=20, batch_size=64, validation_split=0.2, verbose=2,
                    shuffle=False)
print("Elapsed Time for fitiing: ", time.time()-start)
```

```
Epoch 20/20
 - 12s - loss: 0.0376 - mse: 0.0035 - val_loss: 0.0317 - val_mse: 0.0023
Elapsed Time for fitiing:  247.2066159248352
Test RMSE: 56.704
```





همان‌طور که میبینیم در این قسمت که بخشی از دادگان حذف شده بودند
هر دو شبکه نسبت به حالت قبل لاس کمتر و MSE پایین‌تری دارند و توانسته اند برای داده‌هایی که ندیده اند به خوبی عمل کنند
این شبکه‌ها دقت بالاتری داشته‌اند چرا که قابلیت generalize آن‌ها بیشتر است و می‌توانند در برابر داده‌هایی که ندیده اند بهتر عمل کنند
این قضیه برای هر دو شبکه نیز برقرار بوده است