01-RNN-Example

October 19, 2022

For more information, visit us at www.pieriandata.com

1 RNN Example for Time Series

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
```

1.1 Data

https://fred.stlouisfed.org/series/S4248SM144NCEN

[2]:
```
df = pd.read_csv('../Data/Alcohol_Sales.csv',index_col='DATE',parse_dates=True)
df.index.freq = 'MS'
```

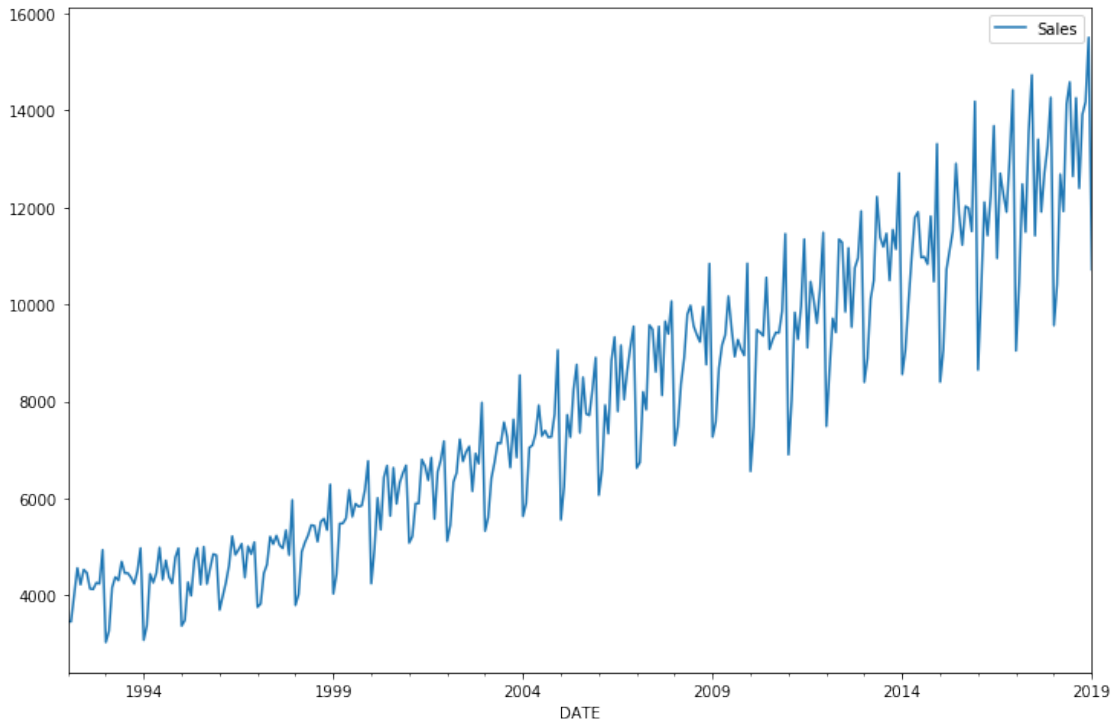
[3]:
```
df.head()
```

[3]:
```
            S4248SM144NCEN
DATE
1992-01-01            3459
1992-02-01            3458
1992-03-01            4002
1992-04-01            4564
1992-05-01            4221
```

[4]:
```
df.columns = ['Sales']
```

[5]:
```
df.plot(figsize=(12,8))
```

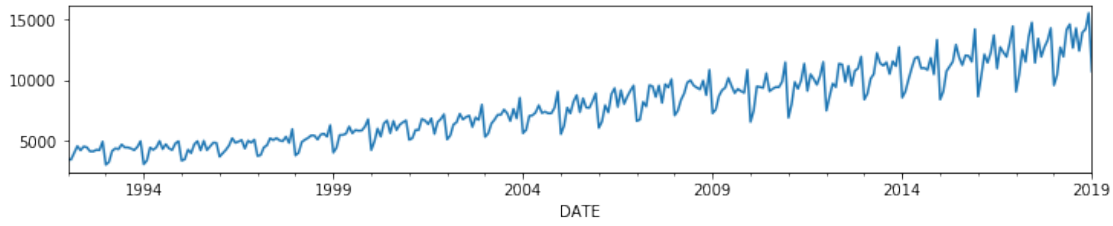
[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29c4949e208>
```



[6]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

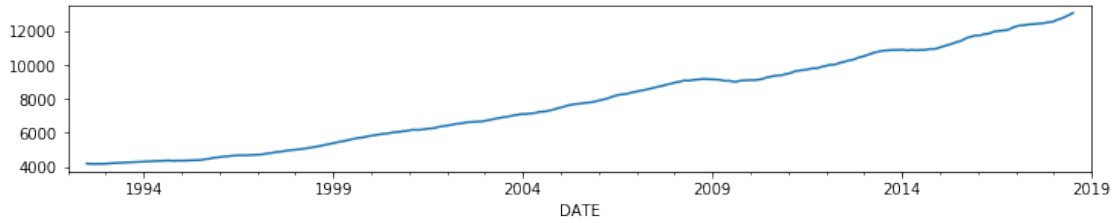
[7]:
```
results = seasonal_decompose(df['Sales'])
results.observed.plot(figsize=(12,2))
```

[7]: <matplotlib.axes._subplots.AxesSubplot at 0x29c4b5852e8>



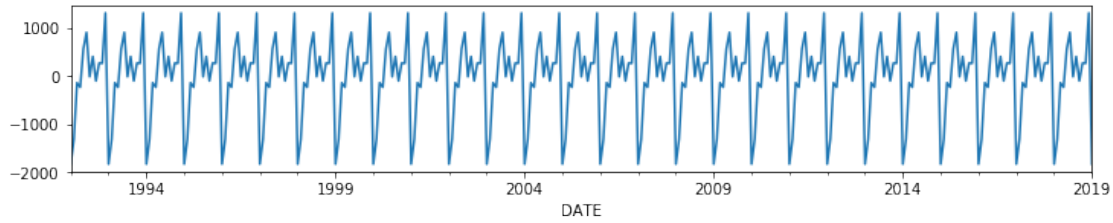
[8]:
```
results.trend.plot(figsize=(12,2))
```

[8]: <matplotlib.axes._subplots.AxesSubplot at 0x29c4b625f28>



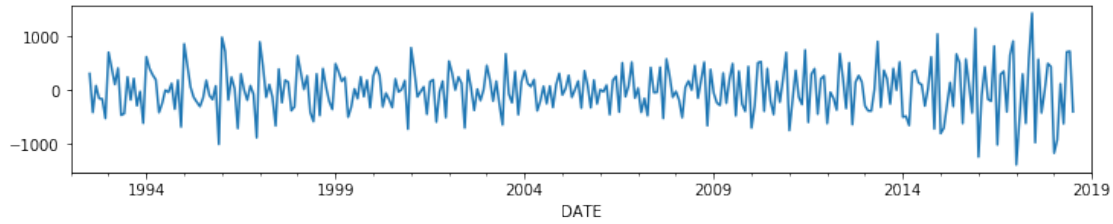
[9]:
```
results.seasonal.plot(figsize=(12,2))
```

[9]: <matplotlib.axes._subplots.AxesSubplot at 0x29c497f5748>



[10]:
```
results.resid.plot(figsize=(12,2))
```

[10]: <matplotlib.axes._subplots.AxesSubplot at 0x29c4b72dba8>



1.2 Train Test Split

[11]:
```
len(df)
```

[11]: 325

[12]:
```
325-12
```

[12]: 313

[13]:
```
train = df.iloc[:313]
test = df.iloc[313:]
```

[14]:
```
len(test)
```

[14]:
```
12
```

1.3 Scale Data

[15]:
```
from sklearn.preprocessing import MinMaxScaler
```

[16]:
```
scaler = MinMaxScaler()
```

[17]:
```
# IGNORE WARNING ITS JUST CONVERTING TO FLOATS
# WE ONLY FIT TO TRAININ DATA, OTHERWISE WE ARE CHEATING ASSUMING INFO ABOUT␣
 ↪TEST SET
scaler.fit(train)
```

```
C:\Users\Marcial\Anaconda3\lib\site-packages\sklearn\preprocessing\data.py:323:
DataConversionWarning: Data with input dtype int64 were all converted to float64
by MinMaxScaler.
  return self.partial_fit(X, y)
```

[17]:
```
MinMaxScaler(copy=True, feature_range=(0, 1))
```

[18]:
```
scaled_train = scaler.transform(train)
scaled_test = scaler.transform(test)
```

2 Time Series Generator

This class takes in a sequence of data-points gathered at equal intervals, along with time series parameters such as stride, length of history, etc., to produce batches for training/validation.

Arguments

```
data: Indexable generator (such as list or Numpy array)
    containing consecutive data points (timesteps).
    The data should be at 2D, and axis 0 is expected
    to be the time dimension.
targets: Targets corresponding to timesteps in `data`.
    It should have same length as `data`.
length: Length of the output sequences (in number of timesteps).
sampling_rate: Period between successive individual timesteps
    within sequences. For rate `r`, timesteps
```

```
    `data[i]`, `data[i-r]`, ... `data[i - length]`
    are used for create a sample sequence.
stride: Period between successive output sequences.
    For stride `s`, consecutive output samples would
    be centered around `data[i]`, `data[i+s]`, `data[i+2*s]`, etc.
start_index: Data points earlier than `start_index` will not be used
    in the output sequences. This is useful to reserve part of the
    data for test or validation.
end_index: Data points later than `end_index` will not be used
    in the output sequences. This is useful to reserve part of the
    data for test or validation.
shuffle: Whether to shuffle output samples,
    or instead draw them in chronological order.
reverse: Boolean: if `true`, timesteps in each output sample will be
    in reverse chronological order.
batch_size: Number of timeseries samples in each batch
    (except maybe the last one).
```

[19]:
```
from keras.preprocessing.sequence import TimeseriesGenerator
```

```
Using TensorFlow backend.
```

[20]:
```
scaled_train
```

[20]:
```
array([[0.03658432],
       [0.03649885],
       [0.08299855],
       [0.13103684],
       [0.1017181 ],
       [0.12804513],
       [0.12266006],
       [0.09453799],
       [0.09359774],
       [0.10496624],
       [0.10334217],
       [0.16283443],
       [0.        ],
       [0.0196598 ],
       [0.09650397],
       [0.11505257],
       [0.10906915],
       [0.14231986],
       [0.12197624],
       [0.12189076],
       [0.11394136],
       [0.10300026],
       [0.12556629],
       [0.16608257],
```

```
[0.00376101],
[0.02957518],
[0.12069408],
[0.10513719],
[0.12214719],
[0.16702282],
[0.11052227],
[0.14428584],
[0.11479614],
[0.10402599],
[0.14984187],
[0.16582614],
[0.02897684],
[0.03872126],
[0.10582101],
[0.08231473],
[0.14394393],
[0.16608257],
[0.10188905],
[0.16830498],
[0.10291478],
[0.13018207],
[0.15556885],
[0.15343192],
[0.0570989 ],
[0.08137448],
[0.10522267],
[0.1357381 ],
[0.18702453],
[0.15428669],
[0.16326182],
[0.17360458],
[0.11402684],
[0.16933071],
[0.15548337],
[0.17659629],
[0.06214206],
[0.06786905],
[0.12163433],
[0.13710574],
[0.18625524],
[0.17317719],
[0.18805026],
[0.17121121],
[0.16574066],
[0.19753825],
[0.1538593 ],
```

```
[0.25079067],
[0.0653902 ],
[0.08445166],
[0.15958629],
[0.17599795],
[0.18856313],
[0.20651338],
[0.20548765],
[0.17745106],
[0.21232584],
[0.2181383 ],
[0.19788016],
[0.27822891],
[0.08556287],
[0.12001026],
[0.20924865],
[0.20959056],
[0.21848021],
[0.2688264 ],
[0.22138644],
[0.24429438],
[0.23908026],
[0.24087529],
[0.2691683 ],
[0.31968544],
[0.1035986 ],
[0.16420207],
[0.25446619],
[0.1984785 ],
[0.29096504],
[0.31130866],
[0.2226686 ],
[0.30763313],
[0.24412343],
[0.28130609],
[0.29823062],
[0.31173605],
[0.17531413],
[0.1867681 ],
[0.24463629],
[0.24472177],
[0.32207881],
[0.31079579],
[0.28575092],
[0.32558338],
[0.21745448],
[0.30036755],
```

```
[0.32122404],
[0.3546457 ],
[0.17830584],
[0.20608599],
[0.28258826],
[0.298658  ],
[0.35772288],
[0.31883067],
[0.33566972],
[0.34524318],
[0.26643303],
[0.33276348],
[0.31498419],
[0.42260022],
[0.19617061],
[0.22036071],
[0.28917001],
[0.31712112],
[0.35156851],
[0.35062826],
[0.38781092],
[0.36199675],
[0.30797504],
[0.39276861],
[0.32583982],
[0.47089495],
[0.22207026],
[0.24506368],
[0.34310625],
[0.34729464],
[0.36772374],
[0.41772801],
[0.36396273],
[0.37310881],
[0.36139841],
[0.36216771],
[0.40174374],
[0.51517224],
[0.21591589],
[0.27404052],
[0.40105992],
[0.36165484],
[0.44533721],
[0.48944354],
[0.36934781],
[0.46713394],
[0.40259851],
```

```
[0.39994871],
[0.44585007],
[0.50183776],
[0.25942388],
[0.30421404],
[0.4181554 ],
[0.36789469],
[0.4967946 ],
[0.53816566],
[0.40695786],
[0.52354902],
[0.42789982],
[0.47944269],
[0.52115565],
[0.55679973],
[0.3073767 ],
[0.31729208],
[0.44140525],
[0.41003505],
[0.55893666],
[0.55158561],
[0.47670741],
[0.55662877],
[0.43525088],
[0.56568938],
[0.54355073],
[0.60124797],
[0.34720916],
[0.38054535],
[0.4559364 ],
[0.50123942],
[0.5780836 ],
[0.59372596],
[0.55748355],
[0.54226857],
[0.52944696],
[0.59124712],
[0.48952902],
[0.66740747],
[0.36199675],
[0.3886657 ],
[0.4835456 ],
[0.52406189],
[0.54175571],
[0.60996666],
[0.55355159],
[0.50363279],
```

```
[0.5334644 ],
[0.51662535],
[0.5058552 ],
[0.66774938],
[0.30147876],
[0.38037439],
[0.55081631],
[0.54645696],
[0.54021711],
[0.64287546],
[0.51679631],
[0.53354988],
[0.54611505],
[0.54551671],
[0.58423797],
[0.72006154],
[0.3307975 ],
[0.42593384],
[0.58133174],
[0.5342337 ],
[0.59287119],
[0.71057355],
[0.51927515],
[0.63578084],
[0.60295752],
[0.56252671],
[0.62372852],
[0.72245491],
[0.38080178],
[0.47952816],
[0.57081802],
[0.54637148],
[0.7104026 ],
[0.70459014],
[0.58244294],
[0.69510215],
[0.55568852],
[0.66014189],
[0.67715189],
[0.75997949],
[0.45850073],
[0.50064108],
[0.60509445],
[0.63783229],
[0.78528079],
[0.71407813],
[0.69706813],
```

```
[0.72065989],
[0.63791777],
[0.72732712],
[0.69296521],
[0.82725019],
[0.47234806],
[0.51525771],
[0.6003932 ],
[0.67920335],
[0.74886742],
[0.75844089],
[0.67817762],
[0.67954526],
[0.66646722],
[0.75100436],
[0.63586631],
[0.8786221 ],
[0.45892811],
[0.51551415],
[0.65740662],
[0.69031541],
[0.72459185],
[0.84391828],
[0.75544918],
[0.70031627],
[0.76852722],
[0.76519361],
[0.72442089],
[0.95324387],
[0.48012651],
[0.62313018],
[0.7757928 ],
[0.71706984],
[0.78699034],
[0.91033422],
[0.67689546],
[0.8264809 ],
[0.78989657],
[0.75852637],
[0.85349175],
[0.97358749],
[0.51389008],
[0.63432772],
[0.80776135],
[0.72313873],
[0.89870929],
[1.        ],
```

```
       [0.71672793],
       [0.88648602],
       [0.75869732],
       [0.82742115],
       [0.87443371],
       [0.96025301],
       [0.5584238 ]])
```

[21]:
```
# define generator
n_input = 2
n_features = 1
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input,␣
 ↪batch_size=1)
```

[22]:
```
len(scaled_train)
```

[22]:
```
313
```

[23]:
```
len(generator) # n_input = 2
```

[23]:
```
311
```

[24]:
```
scaled_train
```

[24]:
```
array([[0.03658432],
       [0.03649885],
       [0.08299855],
       [0.13103684],
       [0.1017181 ],
       [0.12804513],
       [0.12266006],
       [0.09453799],
       [0.09359774],
       [0.10496624],
       [0.10334217],
       [0.16283443],
       [0.        ],
       [0.0196598 ],
       [0.09650397],
       [0.11505257],
       [0.10906915],
       [0.14231986],
       [0.12197624],
       [0.12189076],
       [0.11394136],
       [0.10300026],
       [0.12556629],
```

```
[0.16608257],
[0.00376101],
[0.02957518],
[0.12069408],
[0.10513719],
[0.12214719],
[0.16702282],
[0.11052227],
[0.14428584],
[0.11479614],
[0.10402599],
[0.14984187],
[0.16582614],
[0.02897684],
[0.03872126],
[0.10582101],
[0.08231473],
[0.14394393],
[0.16608257],
[0.10188905],
[0.16830498],
[0.10291478],
[0.13018207],
[0.15556885],
[0.15343192],
[0.0570989 ],
[0.08137448],
[0.10522267],
[0.1357381 ],
[0.18702453],
[0.15428669],
[0.16326182],
[0.17360458],
[0.11402684],
[0.16933071],
[0.15548337],
[0.17659629],
[0.06214206],
[0.06786905],
[0.12163433],
[0.13710574],
[0.18625524],
[0.17317719],
[0.18805026],
[0.17121121],
[0.16574066],
[0.19753825],
```

```
[0.1538593 ],
[0.25079067],
[0.0653902 ],
[0.08445166],
[0.15958629],
[0.17599795],
[0.18856313],
[0.20651338],
[0.20548765],
[0.17745106],
[0.21232584],
[0.2181383 ],
[0.19788016],
[0.27822891],
[0.08556287],
[0.12001026],
[0.20924865],
[0.20959056],
[0.21848021],
[0.2688264 ],
[0.22138644],
[0.24429438],
[0.23908026],
[0.24087529],
[0.2691683 ],
[0.31968544],
[0.1035986 ],
[0.16420207],
[0.25446619],
[0.1984785 ],
[0.29096504],
[0.31130866],
[0.2226686 ],
[0.30763313],
[0.24412343],
[0.28130609],
[0.29823062],
[0.31173605],
[0.17531413],
[0.1867681 ],
[0.24463629],
[0.24472177],
[0.32207881],
[0.31079579],
[0.28575092],
[0.32558338],
[0.21745448],
```

```
[0.30036755],
[0.32122404],
[0.3546457 ],
[0.17830584],
[0.20608599],
[0.28258826],
[0.298658  ],
[0.35772288],
[0.31883067],
[0.33566972],
[0.34524318],
[0.26643303],
[0.33276348],
[0.31498419],
[0.42260022],
[0.19617061],
[0.22036071],
[0.28917001],
[0.31712112],
[0.35156851],
[0.35062826],
[0.38781092],
[0.36199675],
[0.30797504],
[0.39276861],
[0.32583982],
[0.47089495],
[0.22207026],
[0.24506368],
[0.34310625],
[0.34729464],
[0.36772374],
[0.41772801],
[0.36396273],
[0.37310881],
[0.36139841],
[0.36216771],
[0.40174374],
[0.51517224],
[0.21591589],
[0.27404052],
[0.40105992],
[0.36165484],
[0.44533721],
[0.48944354],
[0.36934781],
[0.46713394],
```

```
[0.40259851],
[0.39994871],
[0.44585007],
[0.50183776],
[0.25942388],
[0.30421404],
[0.4181554 ],
[0.36789469],
[0.4967946 ],
[0.53816566],
[0.40695786],
[0.52354902],
[0.42789982],
[0.47944269],
[0.52115565],
[0.55679973],
[0.3073767 ],
[0.31729208],
[0.44140525],
[0.41003505],
[0.55893666],
[0.55158561],
[0.47670741],
[0.55662877],
[0.43525088],
[0.56568938],
[0.54355073],
[0.60124797],
[0.34720916],
[0.38054535],
[0.4559364 ],
[0.50123942],
[0.5780836 ],
[0.59372596],
[0.55748355],
[0.54226857],
[0.52944696],
[0.59124712],
[0.48952902],
[0.66740747],
[0.36199675],
[0.3886657 ],
[0.4835456 ],
[0.52406189],
[0.54175571],
[0.60996666],
[0.55355159],
```

```
[0.50363279],
[0.5334644 ],
[0.51662535],
[0.5058552 ],
[0.66774938],
[0.30147876],
[0.38037439],
[0.55081631],
[0.54645696],
[0.54021711],
[0.64287546],
[0.51679631],
[0.53354988],
[0.54611505],
[0.54551671],
[0.58423797],
[0.72006154],
[0.3307975 ],
[0.42593384],
[0.58133174],
[0.5342337 ],
[0.59287119],
[0.71057355],
[0.51927515],
[0.63578084],
[0.60295752],
[0.56252671],
[0.62372852],
[0.72245491],
[0.38080178],
[0.47952816],
[0.57081802],
[0.54637148],
[0.7104026 ],
[0.70459014],
[0.58244294],
[0.69510215],
[0.55568852],
[0.66014189],
[0.67715189],
[0.75997949],
[0.45850073],
[0.50064108],
[0.60509445],
[0.63783229],
[0.78528079],
[0.71407813],
```

```
[0.69706813],
[0.72065989],
[0.63791777],
[0.72732712],
[0.69296521],
[0.82725019],
[0.47234806],
[0.51525771],
[0.6003932 ],
[0.67920335],
[0.74886742],
[0.75844089],
[0.67817762],
[0.67954526],
[0.66646722],
[0.75100436],
[0.63586631],
[0.8786221 ],
[0.45892811],
[0.51551415],
[0.65740662],
[0.69031541],
[0.72459185],
[0.84391828],
[0.75544918],
[0.70031627],
[0.76852722],
[0.76519361],
[0.72442089],
[0.95324387],
[0.48012651],
[0.62313018],
[0.7757928 ],
[0.71706984],
[0.78699034],
[0.91033422],
[0.67689546],
[0.8264809 ],
[0.78989657],
[0.75852637],
[0.85349175],
[0.97358749],
[0.51389008],
[0.63432772],
[0.80776135],
[0.72313873],
[0.89870929],
```

```
       [1.        ],
       [0.71672793],
       [0.88648602],
       [0.75869732],
       [0.82742115],
       [0.87443371],
       [0.96025301],
       [0.5584238 ]])
```

[25]:
```
# What does the first batch look like?
X,y = generator[0]
```

[26]:
```
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.03658432 0.03649885]
Predict this y:
 [[0.08299855]]
```

[27]:
```
# Let's redefine to get 12 months back and then predict the next month out
n_input = 12
generator = TimeseriesGenerator(scaled_train, scaled_train, length=n_input,␣
 ↪batch_size=1)
```

[28]:
```
# What does the first batch look like?
X,y = generator[0]
```

[29]:
```
print(f'Given the Array: \n{X.flatten()}')
print(f'Predict this y: \n {y}')
```

```
Given the Array:
[0.03658432 0.03649885 0.08299855 0.13103684 0.1017181  0.12804513
 0.12266006 0.09453799 0.09359774 0.10496624 0.10334217 0.16283443]
Predict this y:
 [[0.]]
```

2.0.1 Create the Model

[30]:
```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
```

[31]:
```
# define model
model = Sequential()
model.add(LSTM(100, activation='relu', input_shape=(n_input, n_features)))
```

```
model.add(Dense(1))
model.compile(optimizer='adam', loss='mse')
```

[32]:
```
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

[33]:
```
# fit model
model.fit_generator(generator,epochs=50)
```

```
Epoch 1/50
301/301 [==============================] - 6s 21ms/step - loss: 0.0226
Epoch 2/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0090
Epoch 3/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0081
Epoch 4/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0084
Epoch 5/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0075
Epoch 6/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0065
Epoch 7/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0055
Epoch 8/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0050
Epoch 9/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0039
Epoch 10/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0035
Epoch 11/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0030
Epoch 12/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0030
Epoch 13/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0025
Epoch 14/50
```

```
301/301 [==============================] - 5s 16ms/step - loss: 0.0026
Epoch 15/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0019
Epoch 16/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0022
Epoch 17/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0015
Epoch 18/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0019
Epoch 19/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0018
Epoch 20/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0018
Epoch 21/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0016
Epoch 22/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0016
Epoch 23/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0016
Epoch 24/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0017
Epoch 25/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0018
Epoch 26/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0016
Epoch 27/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0016
Epoch 28/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0014
Epoch 29/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0013
Epoch 30/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0017
Epoch 31/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0015
Epoch 32/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0016
Epoch 33/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0016
Epoch 34/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0014
Epoch 35/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0014
Epoch 36/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0015
Epoch 37/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0015
Epoch 38/50
```

```
301/301 [==============================] - 5s 17ms/step - loss: 0.0012
Epoch 39/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0016
Epoch 40/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0014
Epoch 41/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0016
Epoch 42/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0015
Epoch 43/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0014
Epoch 44/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0014
Epoch 45/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0014
Epoch 46/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0015
Epoch 47/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0014
Epoch 48/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0014
Epoch 49/50
301/301 [==============================] - 5s 17ms/step - loss: 0.0013
Epoch 50/50
301/301 [==============================] - 5s 16ms/step - loss: 0.0014
```

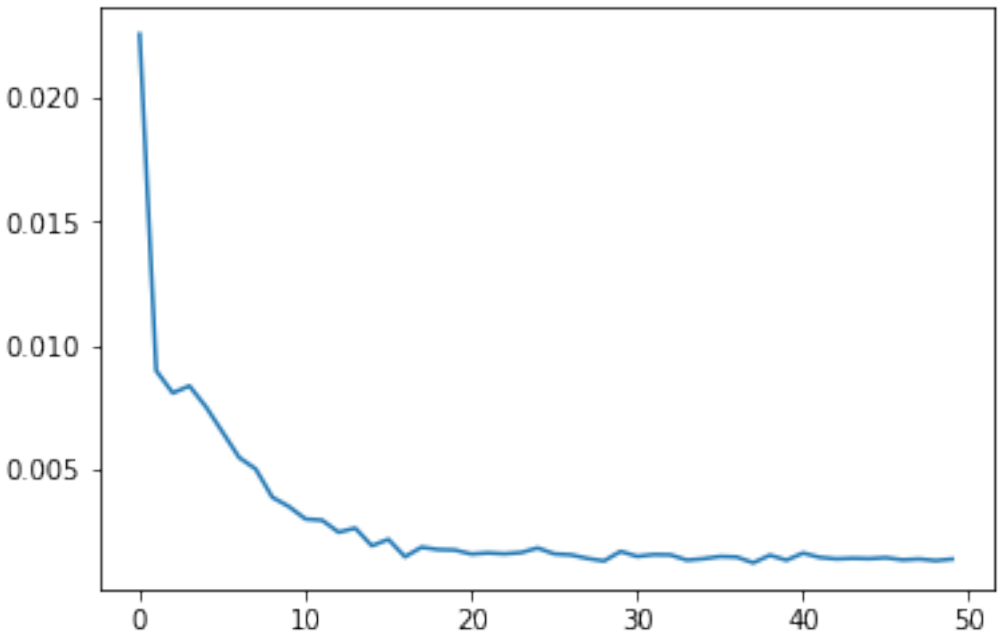
[33]:
```
<keras.callbacks.History at 0x29c52be3f60>
```

[34]:
```
model.history.history.keys()
```

[34]:
```
dict_keys(['loss'])
```

[35]:
```
loss_per_epoch = model.history.history['loss']
plt.plot(range(len(loss_per_epoch)),loss_per_epoch)
```

[35]:
```
[<matplotlib.lines.Line2D at 0x29e76fddc88>]
```



2.1 Evaluate on Test Data

[36]:
```
first_eval_batch = scaled_train[-12:]
```

[37]:
```
first_eval_batch
```

[37]:
```
array([[0.63432772],
       [0.80776135],
       [0.72313873],
       [0.89870929],
       [1.        ],
       [0.71672793],
       [0.88648602],
       [0.75869732],
       [0.82742115],
       [0.87443371],
       [0.96025301],
       [0.5584238 ]])
```

[38]:
```
first_eval_batch = first_eval_batch.reshape((1, n_input, n_features))
```

[39]:
```
model.predict(first_eval_batch)
```

[39]:
```
array([[0.72332144]], dtype=float32)
```

[40]:
```
scaled_test[0]
```

[40]:
```
array([0.63116506])
```

Now let's put this logic in a for loop to predict into the future for the entire test range.

---

[41]:
```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))
```

[42]:
```
current_batch.shape
```

[42]:
```
(1, 12, 1)
```

[43]:
```
current_batch
```

[43]:
```
array([[[0.63432772],
        [0.80776135],
        [0.72313873],
        [0.89870929],
        [1.        ],
        [0.71672793],
        [0.88648602],
        [0.75869732],
        [0.82742115],
        [0.87443371],
        [0.96025301],
        [0.5584238 ]]])
```

[44]:
```
np.append(current_batch[:,1:,:],[[[99]]],axis=1)
```

[44]:
```
array([[[ 0.80776135],
        [ 0.72313873],
        [ 0.89870929],
        [ 1.        ],
        [ 0.71672793],
        [ 0.88648602],
        [ 0.75869732],
        [ 0.82742115],
        [ 0.87443371],
        [ 0.96025301],
        [ 0.5584238 ],
        [99.        ]]])
```

NOTE: PAY CLOSE ATTENTION HERE TO WHAT IS BEING OUTPUTED AND IN WHAT DIMENSIONS. ADD YOUR OWN PRINT() STATEMENTS TO SEE

WHAT IS TRULY GOING ON!!

[45]:
```
test_predictions = []

first_eval_batch = scaled_train[-n_input:]
current_batch = first_eval_batch.reshape((1, n_input, n_features))

for i in range(len(test)):

    # get prediction 1 time stamp ahead ([0] is for grabbing just the number␣
 ↪instead of [array])
    current_pred = model.predict(current_batch)[0]

    # store prediction
    test_predictions.append(current_pred)

    # update batch to now include prediction and drop first value
    current_batch = np.append(current_batch[:,1:,:],[[current_pred]],axis=1)
```

[46]:
```
test_predictions
```

[46]:
```
[array([0.72332144], dtype=float32),
 array([0.8512236], dtype=float32),
 array([0.79676867], dtype=float32),
 array([0.9748083], dtype=float32),
 array([1.0563692], dtype=float32),
 array([0.7824111], dtype=float32),
 array([0.9462028], dtype=float32),
 array([0.8080985], dtype=float32),
 array([0.8912762], dtype=float32),
 array([0.9375939], dtype=float32),
 array([1.015485], dtype=float32),
 array([0.6268751], dtype=float32)]
```

[47]:
```
scaled_test
```

[47]:
```
array([[0.63116506],
       [0.82502778],
       [0.75972305],
       [0.94939738],
       [0.98743482],
       [0.82135225],
       [0.95956919],
       [0.80049577],
       [0.93025045],
       [0.95247457],
       [1.0661595 ],
       [0.65706471]])
```

2.2 Inverse Transformations and Compare

[48]:
```
true_predictions = scaler.inverse_transform(test_predictions)
```

[49]:
```
true_predictions
```

[49]:
```
array([[11493.13750124],
       [12989.46475589],
       [12352.39661551],
       [14435.28201741],
       [15389.46310055],
       [12184.42744112],
       [14100.62672776],
       [12484.94429308],
       [13458.04003853],
       [13999.91077071],
       [14911.15957999],
       [10364.81182438]])
```

[50]:
```
test
```

[50]:

DATE	Sales
2018-02-01	10415
2018-03-01	12683
2018-04-01	11919
2018-05-01	14138
2018-06-01	14583
2018-07-01	12640
2018-08-01	14257
2018-09-01	12396
2018-10-01	13914
2018-11-01	14174
2018-12-01	15504
2019-01-01	10718

[51]:
```
# IGNORE WARNINGS
test['Predictions'] = true_predictions
```

```
C:\Users\Marcial\Anaconda3\lib\site-packages\ipykernel_launcher.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
```

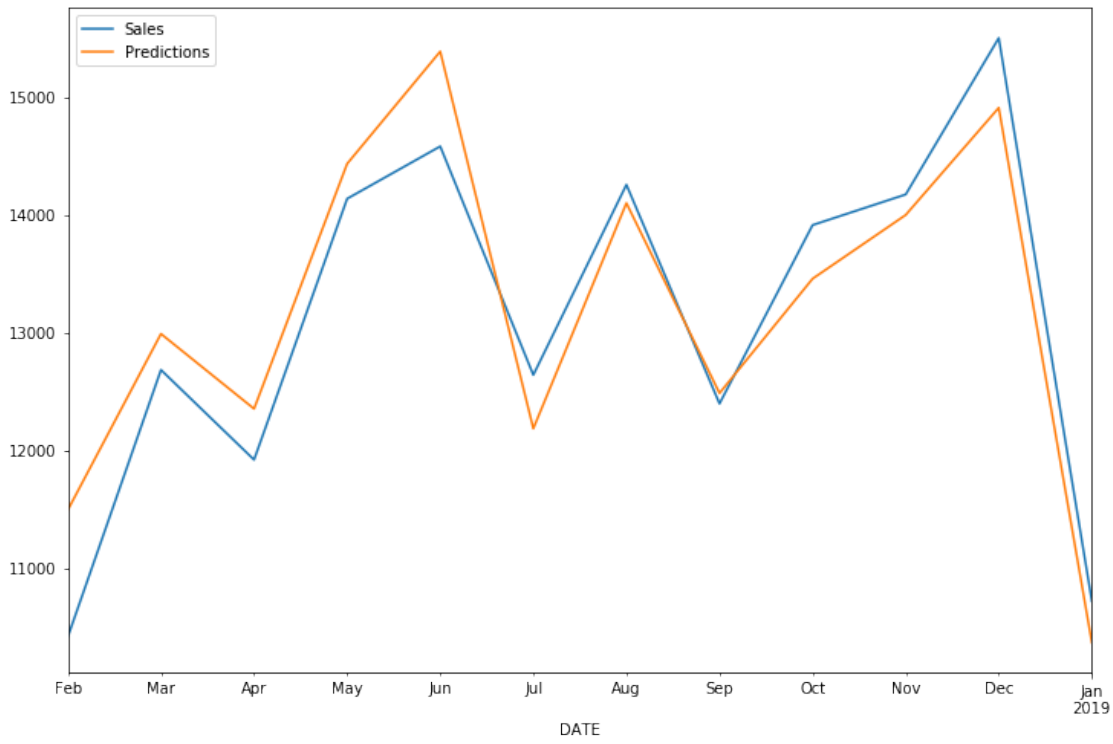
[52]:
```
test
```

[52]:

DATE	Sales	Predictions
2018-02-01	10415	11493.137501
2018-03-01	12683	12989.464756
2018-04-01	11919	12352.396616
2018-05-01	14138	14435.282017
2018-06-01	14583	15389.463101
2018-07-01	12640	12184.427441
2018-08-01	14257	14100.626728
2018-09-01	12396	12484.944293
2018-10-01	13914	13458.040039
2018-11-01	14174	13999.910771
2018-12-01	15504	14911.159580
2019-01-01	10718	10364.811824

[53]:
```
test.plot(figsize=(12,8))
```

[53]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29c7e955f28>
```



3 Saving and Loading Models

[54]:
```
model.save('my_rnn_model.h5')
```

3.1 load a model

[55]:
```
from keras.models import load_model
new_model = load_model('my_rnn_model.h5')
```

[56]:
```
new_model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_1 (LSTM)                (None, 100)               40800
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 101
=================================================================
Total params: 40,901
Trainable params: 40,901
Non-trainable params: 0
_________________________________________________________________
```

[ ]:
```

```