HW4 - ans

April 12, 2022

1 LSTM model of StockData

Intro to LSTM

In this notebook we will go through a basic Long Short Term Memory (LSTM) model for time series. The notebooks does the following things: * First load in the data. The preproccessing only consist of normalization and the creation of windows. * Creation of the LSTM model * Training the LSTM model * Testing the LSTM model with 1 time step and with 1 window

1.1 Importing libraries and loading in the data

1.1.1 Import libraries

[1]:
```
import warnings
warnings.filterwarnings("ignore")
import tensorflow as tf
import re
import math
```

[2]:
```
import matplotlib.pyplot as plt
import statsmodels.tsa.seasonal as smt
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import random
import datetime as dt
from sklearn import linear_model
from sklearn.metrics import mean_absolute_error
import plotly
import os
from subprocess import check_output

# import the relevant Keras modules
from keras.models import Sequential
from keras.layers import Activation, Dense
from keras.layers import LSTM
from keras.layers import Dropout
```

[3]:
```
# Use CPU to run the model
os.environ["CUDA_VISIBLE_DEVICES"]="-1"
```

1.1.2 1) Loading in the data

Select a stock or ETF for this HW from the ones available.

[4]:
```
filename = "ETFs\\adre.us.txt"#"<insert file name>"
print(filename)

df = pd.read_csv(filename, sep=",")
df['Label'] = filename
df['Date'] = pd.to_datetime(df['Date'])
df.head()
```

```
ETFs\adre.us.txt
```

[4]:
```
        Date    Open    High     Low   Close  Volume  OpenInt  \
0 2005-02-25  19.065  19.416  19.065  19.416   72019        0
1 2005-02-28  20.172  20.172  19.312  19.380  101346        0
2 2005-03-01  19.798  19.798  19.209  19.268   53671        0
3 2005-03-02  19.109  19.195  19.042  19.160   23894        0
4 2005-03-03  19.744  19.744  19.127  19.187   28870        0

             Label
0  ETFs\adre.us.txt
1  ETFs\adre.us.txt
2  ETFs\adre.us.txt
3  ETFs\adre.us.txt
4  ETFs\adre.us.txt
```

1.1.3 Visualize the data

Plotly is a nice visualization library that has some interactive plot options.

[5]:
```
r = lambda: random.randint(0,255)
traces = []

#for df in data:
clr = str(r()) + str(r()) + str(r())
df = df.sort_values('Date')
label = df['Label'].iloc[0]

trace = plotly.graph_objs.Scattergl(
    x=df['Date'],
    y=df['Close']#,mode='line',line=dict(color = "blue")
)
traces.append(trace)

layout = plotly.graph_objs.Layout(
    title='Plot '+label.split(sep='.')[0].upper(),
)
```

```
fig = plotly.graph_objs.Figure(data=traces, layout=layout)

plotly.offline.init_notebook_mode(connected=True)
plotly.offline.iplot(fig, filename='dataplot')
```

1.1.4 2) Creating windows and normalizing the data

The default window here is 10. The final question will ask you to consider this parameter in your final analysis and how it might impact your results.

[6]:
```
window_len = 10

#Create a data point (i.e. a date) which splits the training and testing set
split_date = list(df["Date"][-(2*window_len+1):])[0]
print("split_date:",split_date)


#Split the training and test set
training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
↪split_date]
training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
test_set = test_set.drop(['Date','Label','OpenInt'], 1)

#Create windows for training
LSTM_training_inputs = []
for i in range(len(training_set)-window_len):
    temp_set = training_set[i:(i+window_len)].copy()

    for col in list(temp_set):
        temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

    LSTM_training_inputs.append(temp_set)
LSTM_training_inputs
LSTM_training_outputs = (training_set['Close'][window_len:].values/training_set[
    'Close'][:-window_len].values)-1

LSTM_training_inputs = [np.array(LSTM_training_input) for LSTM_training_input␣
↪in LSTM_training_inputs]
LSTM_training_inputs = np.array(LSTM_training_inputs)

# Create windows for testing
LSTM_test_inputs = []
for i in range(len(test_set)-window_len):
    temp_set = test_set[i:(i+window_len)].copy()

    for col in list(temp_set):
        temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1
```

```
    LSTM_test_inputs.append(temp_set)
LSTM_test_outputs = (test_set['Close'][window_len:].values/test_set['Close'][:
 ↪-window_len].values)-1

LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
 ↪LSTM_test_inputs]
LSTM_test_inputs = np.array(LSTM_test_inputs)
```

```
split_date: 2017-10-13 00:00:00
```

1.2 3) LSTM model definition

LSTM's have a set of parameters that can be tuned to your data set. Consider these inputs: **activation function, loss function, dropout rate, optimizer, nn layers/architecture** and review your options in the documentation.

Keras Docs

[7]:
```
def build_model(inputs, output_size, neurons, activ_func="linear",
                dropout=0.10, loss="mae", optimizer="adam"):

    model = Sequential()

    model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))

    model.compile(loss=loss, optimizer=optimizer)
    return model
```

1.3 4) Training of the LSTM model

Just like most ML models choosing a stopping condition is important. Here we use **Epochs** or iterations to set this stopping condition where we also monitor the loss at each step. Consider **Epochs** as a parameter to adjust.

[8]:
```
# initialise model architecture
nn_model = build_model(LSTM_training_inputs, output_size=1, neurons = 32)
# model output is next price normalised to 10th previous closing price
# train model on data
# note: eth_history contains information on the training error per epoch
nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                            epochs=5, batch_size=1, verbose=2, shuffle=True)
```

```
Epoch 1/5
3170/3170 - 5s - loss: 0.0240 - 5s/epoch - 2ms/step
Epoch 2/5
```

```
3170/3170 - 4s - loss: 0.0157 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```

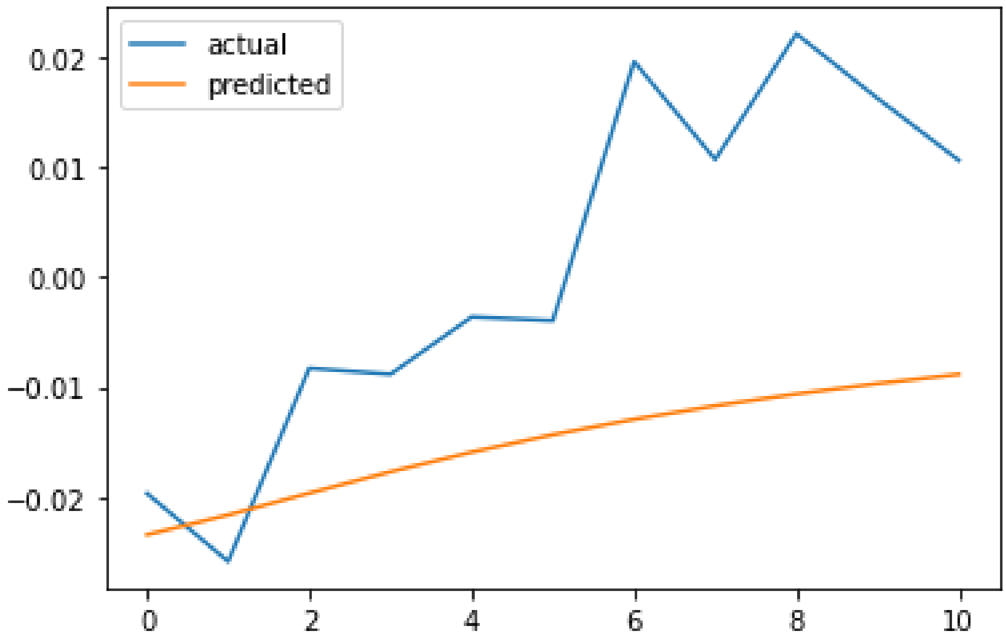
1.3.1 Plot of prediction of one data point ahead

As can be seen in the plot, one step prediction is not bad. The scale is a bit off, because the data is normalized.

1.3.2 Prediction of one window (n steps) ahead

As can be seen in the plot below, the performance degrades when predicting multiple time points ahead. However, compered to something like linear regression the performance is better.

[9]:
```python
def predict_sequence_full(model, data, window_size):
    #Shift the window by 1 new prediction each time, re-run predictions on new␣
↪window
    curr_frame = data[0]
    predicted = []
    for i in range(len(data)):
        predicted.append(model.predict(curr_frame[np.newaxis,:,:])[0,0])
        curr_frame = curr_frame[1:]
        curr_frame = np.insert(curr_frame, [window_size-1], predicted[-1],␣
↪axis=0)
    return predicted

predictions = predict_sequence_full(nn_model, LSTM_test_inputs, 10)

plt.plot(LSTM_test_outputs, label="actual")
plt.plot(predictions, label="predicted")
plt.legend()
plt.show()
MAE = mean_absolute_error(LSTM_test_outputs, predictions)
print('The Mean Absolute Error is: {}'.format(MAE))
```




```
The Mean Absolute Error is: 0.016746579302144234
```

1.4 Conclusion

For this HW your task is to select a stock or ETF from the provided folder. You will run the code above with the default parameters and understand the logic and flow of the program. Once you are confident the code runs you are to test different parameter settings. You are to report the best set of parameters that you find and explain the importance of each parameter and how it impacts the training of the model. To best know how much to adjust and how to interpret the impacts I suggest changing one parameter at a at a time. This is a manual grid search you are performing so that you can become familiar with each parameter. In the future you can have grid search algorithms find the best set for you.

For each define the parameter and it's impact to the model. Report the set of values tested and the best parameter setting you found for each.

1) Window Length
2) LSTM Parameter: activation function
3) LSTM Parameter: loss function
4) LSTM Parameter: dropout rate
5) LSTM Parameter: optimizer
6) LSTM Parameter: nn layers/architecture
7) Epochs

The last cell has the questions 1-7 which expects a two part response, 2.pts each question, 1pt for the explanation of the parameter and 1 point for set of values tested with the best value for that particular parameters, this can be reported as an optimal set of parameters at the end. The 8th quest is submitted separately and asks you to perform back propagation on a simple NN for the final 1 pt. In all there are 15 points possible. Please download this notebook, import it into your workspace, download the stock or EFT in the files. You need to select one to train the NN on. Your completed HW should be a notebook with the optimal set of parameters and your answers in the last cell as markdown syntax. Then download the back propagation document and submit that answer as a separate document.

2 Here are my answers to the above questions

[10]:
```
filename = "ETFs\\adre.us.txt"
df = pd.read_csv(filename, sep=",")
df['Label'] = filename
df['Date'] = pd.to_datetime(df['Date'])
df.head()
```

[10]:
```
        Date    Open    High     Low   Close  Volume  OpenInt  \
0 2005-02-25  19.065  19.416  19.065  19.416   72019        0
1 2005-02-28  20.172  20.172  19.312  19.380  101346        0
2 2005-03-01  19.798  19.798  19.209  19.268   53671        0
3 2005-03-02  19.109  19.195  19.042  19.160   23894        0
4 2005-03-03  19.744  19.744  19.127  19.187   28870        0

             Label
0  ETFs\adre.us.txt
1  ETFs\adre.us.txt
2  ETFs\adre.us.txt
3  ETFs\adre.us.txt
4  ETFs\adre.us.txt
```

[11]:
```
def build_model(inputs, output_size, neurons, activ_func="linear",
                dropout=0.10, loss="mae", optimizer="adam"):

    model = Sequential()

    model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))

    model.compile(loss=loss, optimizer=optimizer)
    return model
```

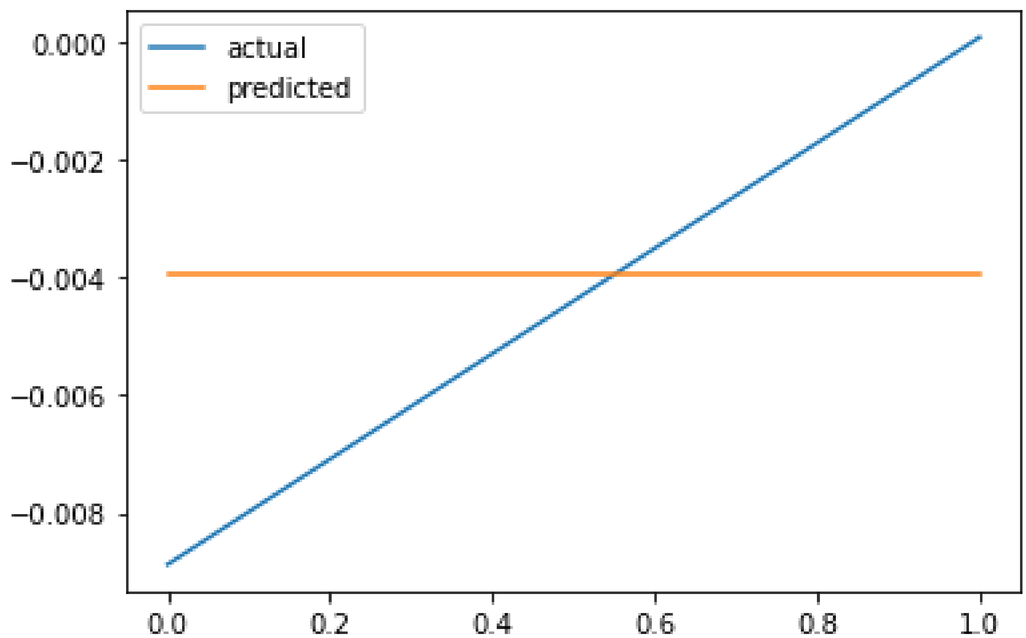
2.1 Window Length

[12]:
```
window_len_lst = list(range(1, 20))
window_dic = {}
```

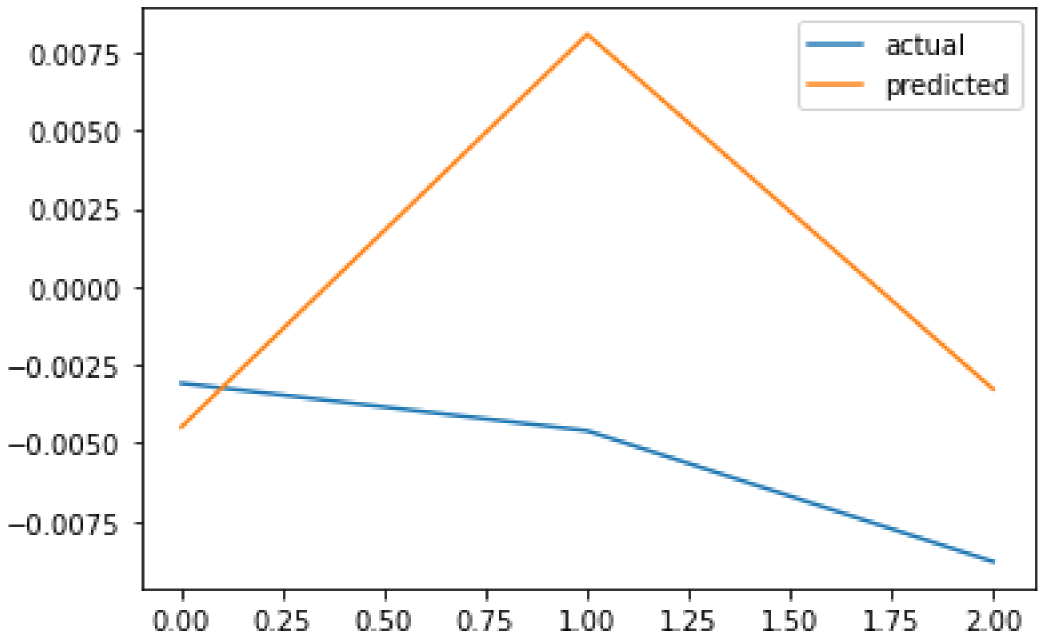
```
for window_len in window_len_lst:
    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)

    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
 ↪split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
    LSTM_training_outputs = (training_set['Close'][window_len:].values/
 ↪training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for␣
 ↪LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/
 ↪test_set['Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
 ↪LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)

    # initialise model architecture
    nn_model = build_model(LSTM_training_inputs, output_size=1, neurons = 32)
```

```
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                              epochs=5, batch_size=1, verbose=2, shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    MAE = mean_absolute_error(LSTM_test_outputs, nn_model.
↪predict(LSTM_test_inputs))
    window_dic[window_len] = MAE
window_result = pd.DataFrame(window_dic.values(), window_dic.keys()).
↪rename(columns={0: "MAE"})
```

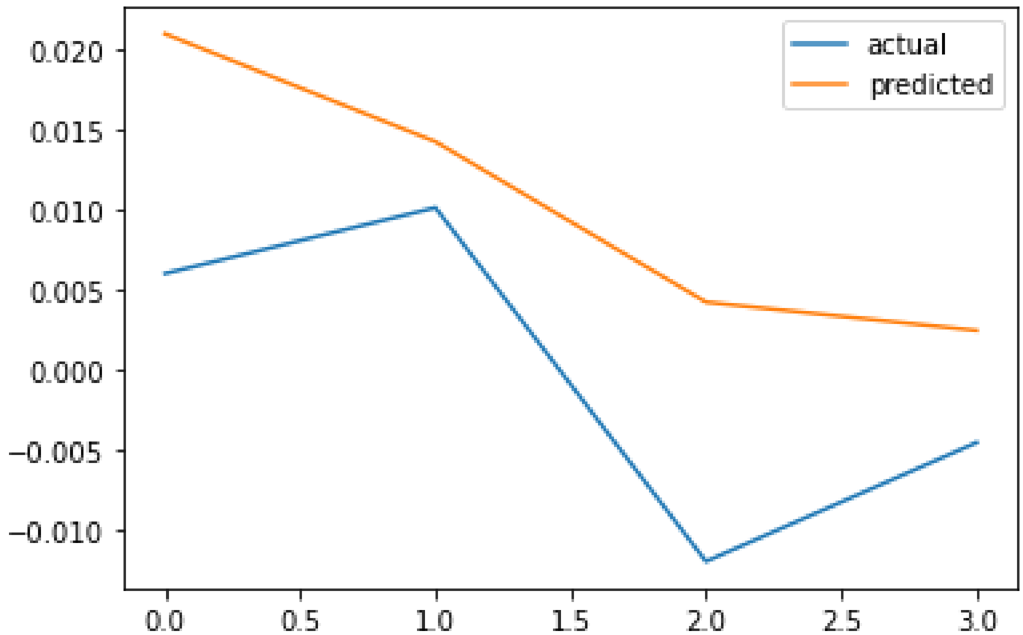
```
split_date: 2017-11-08 00:00:00
Epoch 1/5
3197/3197 - 3s - loss: 0.0125 - 3s/epoch - 839us/step
Epoch 2/5
3197/3197 - 2s - loss: 0.0125 - 2s/epoch - 584us/step
Epoch 3/5
3197/3197 - 2s - loss: 0.0125 - 2s/epoch - 622us/step
Epoch 4/5
3197/3197 - 2s - loss: 0.0124 - 2s/epoch - 645us/step
Epoch 5/5
3197/3197 - 2s - loss: 0.0124 - 2s/epoch - 617us/step
```



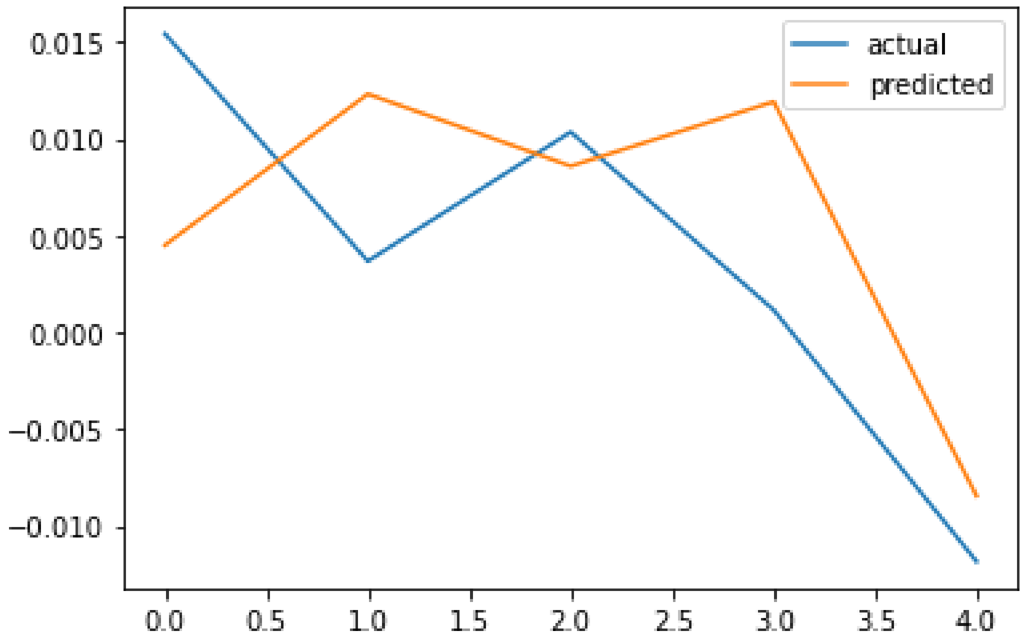
```
split_date: 2017-11-06 00:00:00
Epoch 1/5
3194/3194 - 3s - loss: 0.0167 - 3s/epoch - 1ms/step
Epoch 2/5
3194/3194 - 2s - loss: 0.0136 - 2s/epoch - 693us/step
Epoch 3/5
3194/3194 - 2s - loss: 0.0132 - 2s/epoch - 771us/step
Epoch 4/5
3194/3194 - 3s - loss: 0.0129 - 3s/epoch - 810us/step
Epoch 5/5
3194/3194 - 2s - loss: 0.0129 - 2s/epoch - 746us/step
```



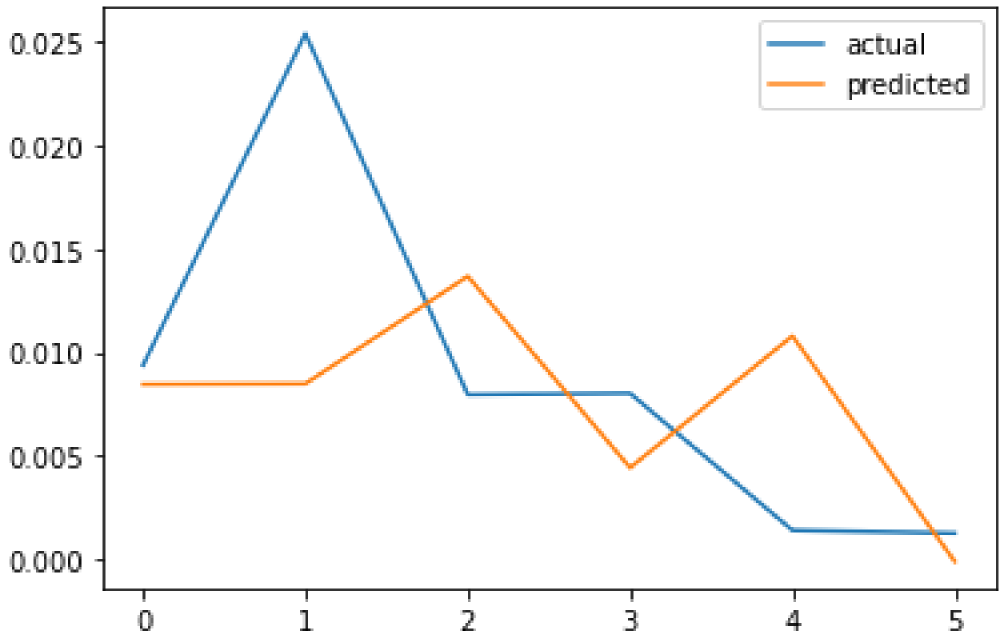

```
split_date: 2017-11-02 00:00:00
Epoch 1/5
3191/3191 - 4s - loss: 0.0200 - 4s/epoch - 1ms/step
Epoch 2/5
3191/3191 - 3s - loss: 0.0144 - 3s/epoch - 980us/step
Epoch 3/5
3191/3191 - 3s - loss: 0.0135 - 3s/epoch - 996us/step
Epoch 4/5
3191/3191 - 3s - loss: 0.0132 - 3s/epoch - 865us/step
Epoch 5/5
3191/3191 - 3s - loss: 0.0131 - 3s/epoch - 885us/step
```



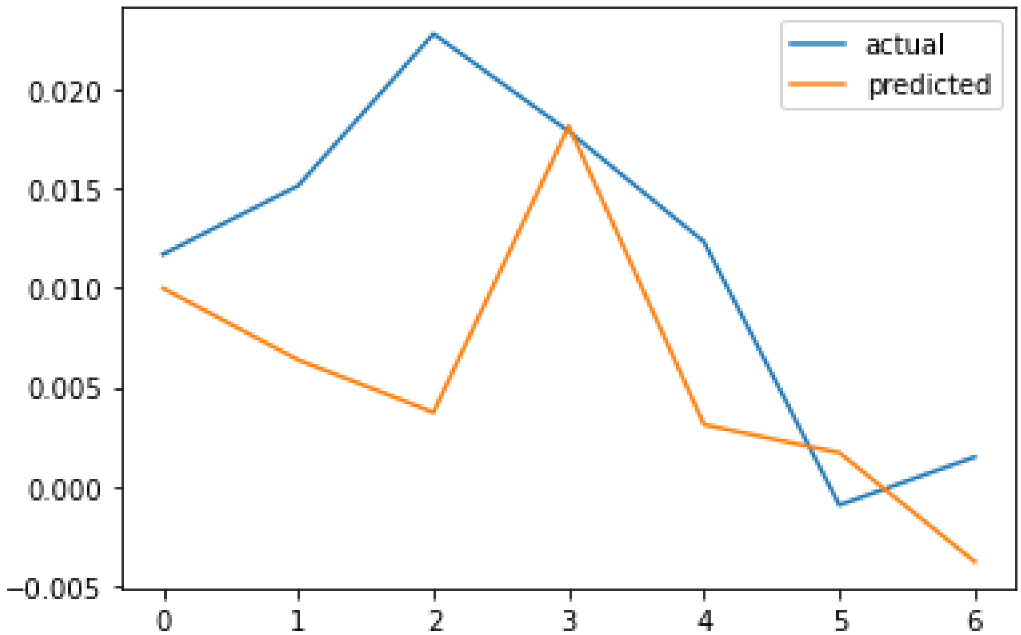

```
split_date: 2017-10-31 00:00:00
Epoch 1/5
3188/3188 - 4s - loss: 0.0210 - 4s/epoch - 1ms/step
Epoch 2/5
3188/3188 - 3s - loss: 0.0147 - 3s/epoch - 833us/step
Epoch 3/5
3188/3188 - 3s - loss: 0.0138 - 3s/epoch - 860us/step
Epoch 4/5
3188/3188 - 3s - loss: 0.0134 - 3s/epoch - 857us/step
Epoch 5/5
3188/3188 - 3s - loss: 0.0134 - 3s/epoch - 959us/step
WARNING:tensorflow:5 out of the last 18 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001BF67246820>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



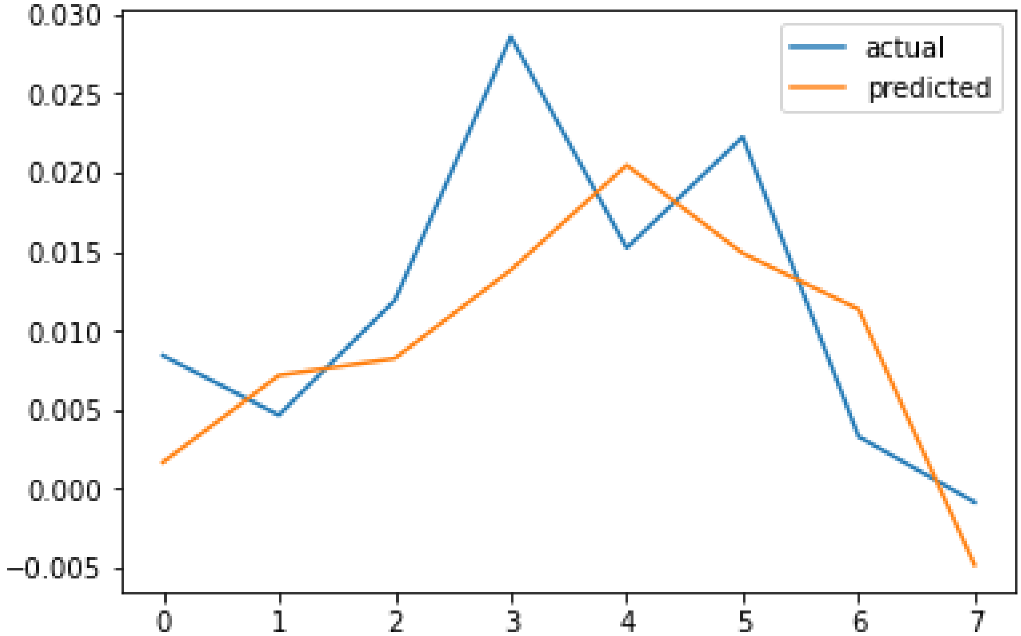

```
split_date: 2017-10-27 00:00:00
Epoch 1/5
3185/3185 - 4s - loss: 0.0197 - 4s/epoch - 1ms/step
Epoch 2/5
3185/3185 - 3s - loss: 0.0147 - 3s/epoch - 915us/step
Epoch 3/5
3185/3185 - 3s - loss: 0.0139 - 3s/epoch - 908us/step
Epoch 4/5
3185/3185 - 3s - loss: 0.0136 - 3s/epoch - 1ms/step
Epoch 5/5
3185/3185 - 3s - loss: 0.0135 - 3s/epoch - 1ms/step
WARNING:tensorflow:6 out of the last 20 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001BF671FDDC0>
triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2)
passing tensors with different shapes, (3) passing Python objects instead of
tensors. For (1), please define your @tf.function outside of the loop. For (2),
@tf.function has experimental_relax_shapes=True option that relaxes argument
shapes that can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
```



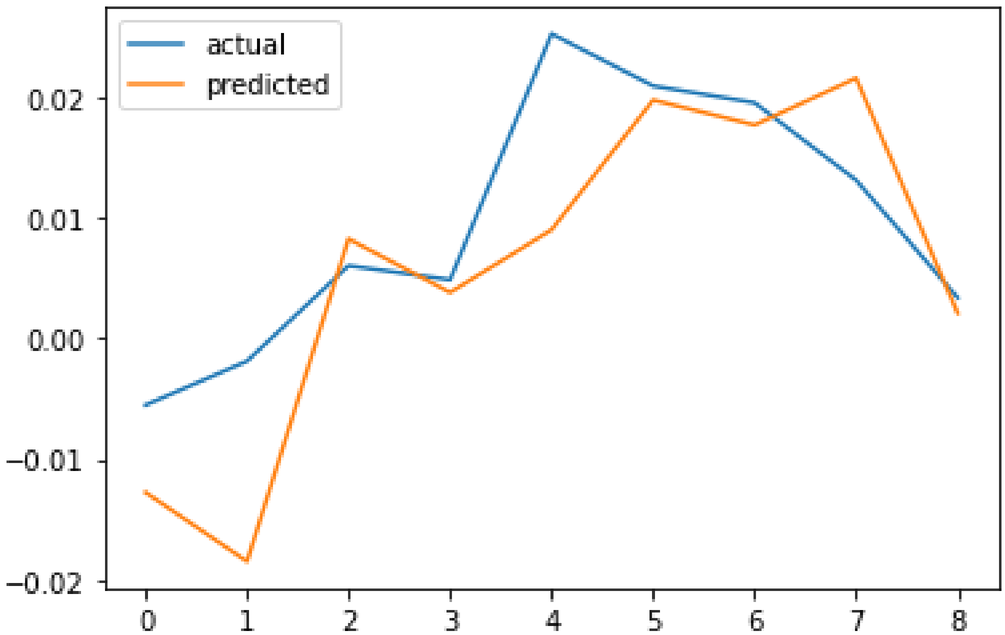

```
split_date: 2017-10-25 00:00:00
Epoch 1/5
3182/3182 - 4s - loss: 0.0209 - 4s/epoch - 1ms/step
Epoch 2/5
3182/3182 - 3s - loss: 0.0148 - 3s/epoch - 1ms/step
Epoch 3/5
3182/3182 - 3s - loss: 0.0142 - 3s/epoch - 1ms/step
Epoch 4/5
3182/3182 - 3s - loss: 0.0138 - 3s/epoch - 1ms/step
Epoch 5/5
3182/3182 - 3s - loss: 0.0135 - 3s/epoch - 1ms/step
```



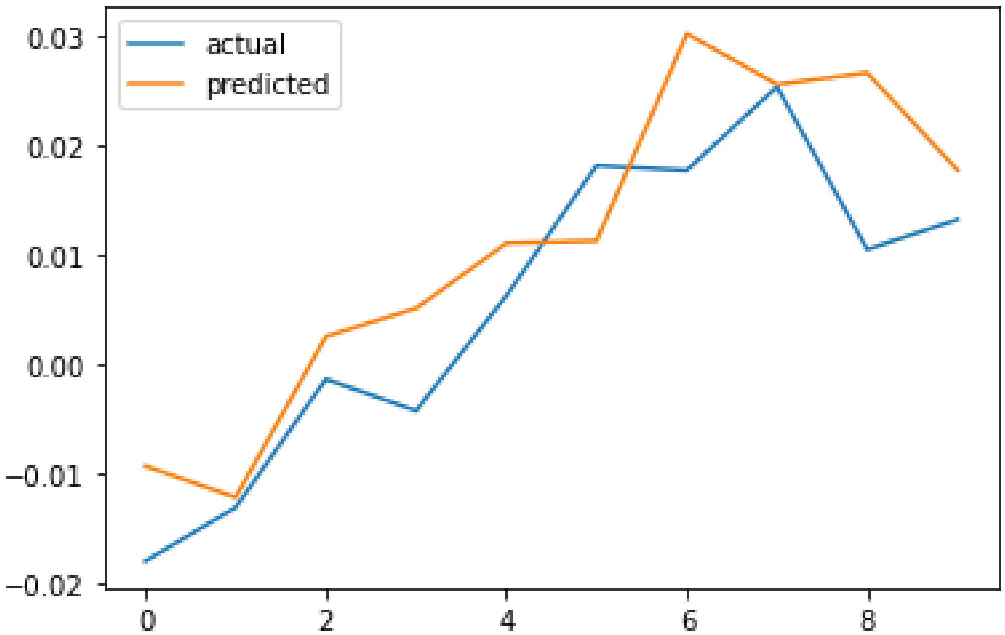

```
split_date: 2017-10-23 00:00:00
Epoch 1/5
3179/3179 - 4s - loss: 0.0233 - 4s/epoch - 1ms/step
Epoch 2/5
3179/3179 - 3s - loss: 0.0154 - 3s/epoch - 1ms/step
Epoch 3/5
3179/3179 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
Epoch 4/5
3179/3179 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
Epoch 5/5
3179/3179 - 4s - loss: 0.0136 - 4s/epoch - 1ms/step
```



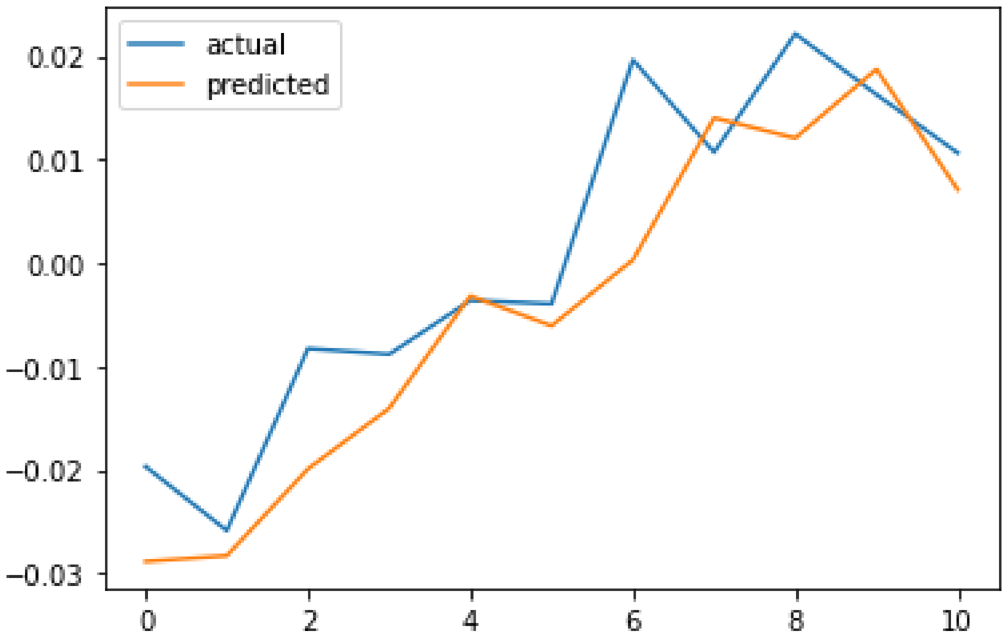

```
split_date: 2017-10-19 00:00:00
Epoch 1/5
3176/3176 - 5s - loss: 0.0213 - 5s/epoch - 2ms/step
Epoch 2/5
3176/3176 - 4s - loss: 0.0156 - 4s/epoch - 1ms/step
Epoch 3/5
3176/3176 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 4/5
3176/3176 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3176/3176 - 5s - loss: 0.0137 - 5s/epoch - 1ms/step
```



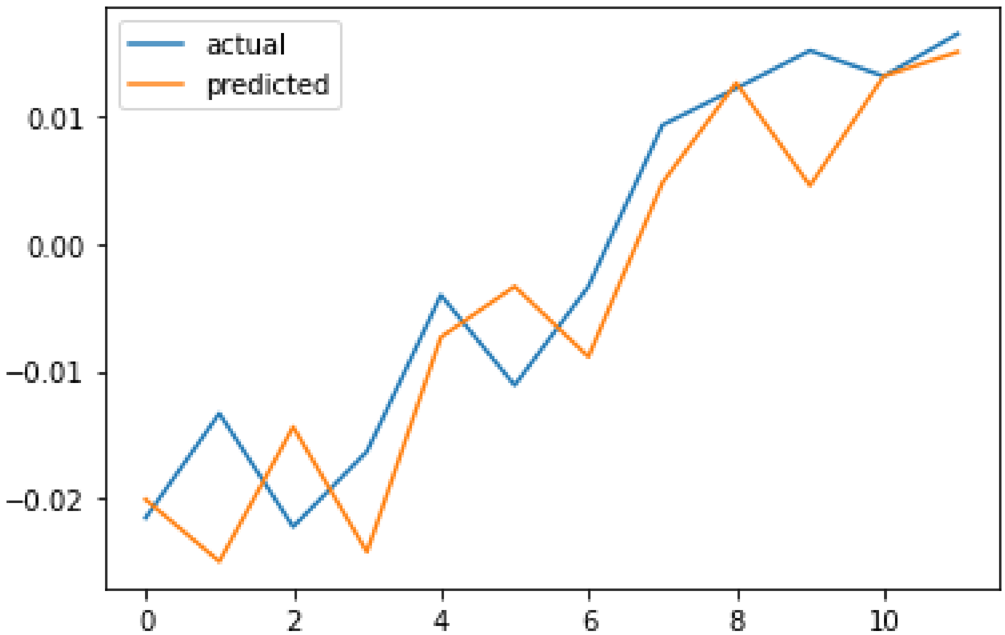

```
split_date: 2017-10-17 00:00:00
Epoch 1/5
3173/3173 - 5s - loss: 0.0246 - 5s/epoch - 2ms/step
Epoch 2/5
3173/3173 - 4s - loss: 0.0162 - 4s/epoch - 1ms/step
Epoch 3/5
3173/3173 - 4s - loss: 0.0148 - 4s/epoch - 1ms/step
Epoch 4/5
3173/3173 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3173/3173 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



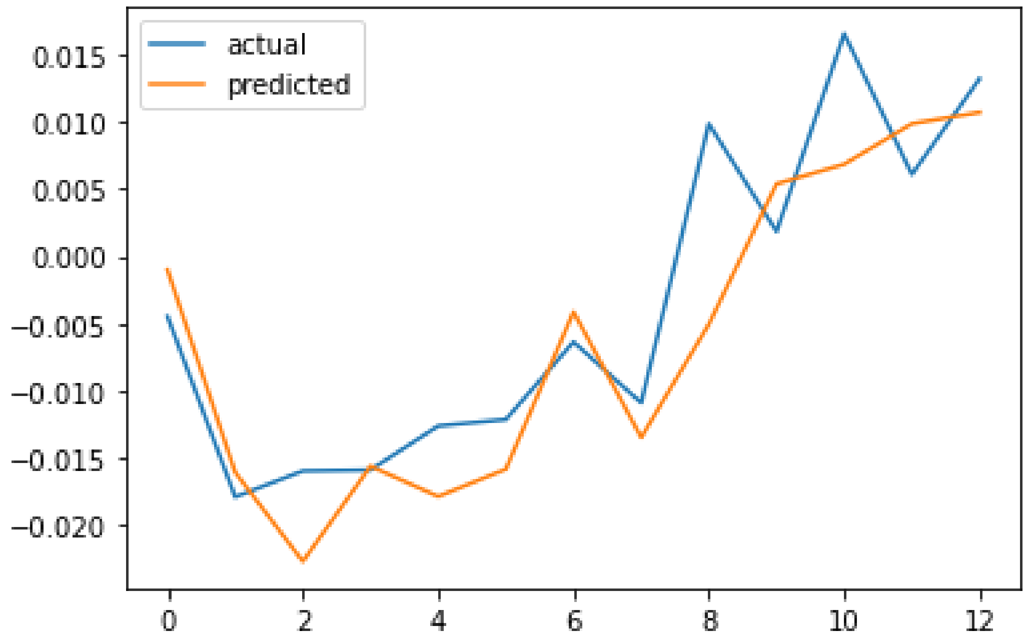

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0251 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0167 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0149 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
```



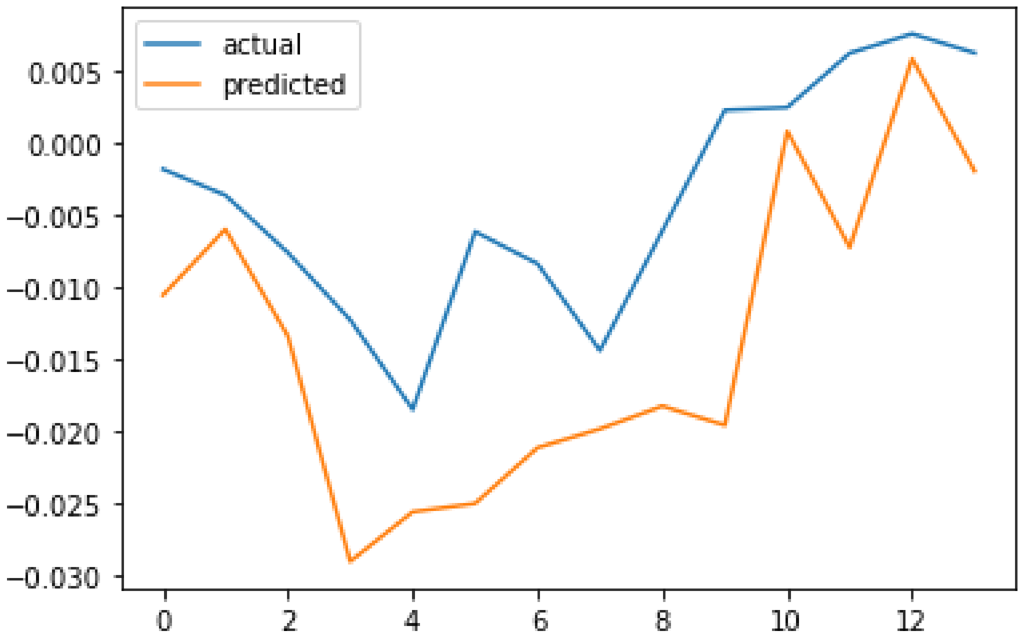

```
split_date: 2017-10-11 00:00:00
Epoch 1/5
3167/3167 - 5s - loss: 0.0221 - 5s/epoch - 2ms/step
Epoch 2/5
3167/3167 - 4s - loss: 0.0159 - 4s/epoch - 1ms/step
Epoch 3/5
3167/3167 - 5s - loss: 0.0146 - 5s/epoch - 1ms/step
Epoch 4/5
3167/3167 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3167/3167 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



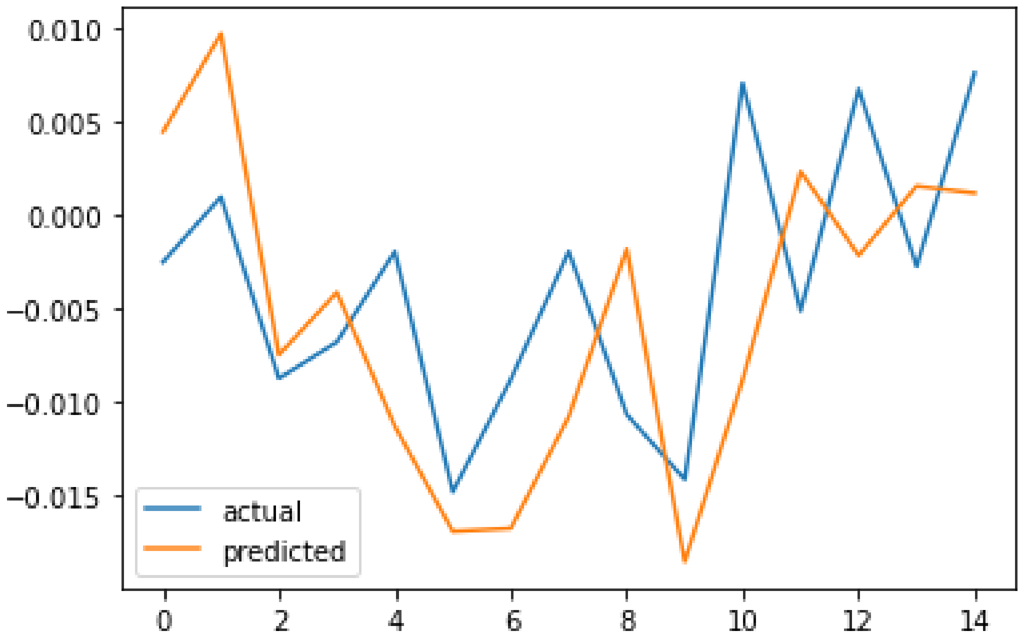

```
split_date: 2017-10-09 00:00:00
Epoch 1/5
3164/3164 - 6s - loss: 0.0287 - 6s/epoch - 2ms/step
Epoch 2/5
3164/3164 - 5s - loss: 0.0174 - 5s/epoch - 2ms/step
Epoch 3/5
3164/3164 - 5s - loss: 0.0151 - 5s/epoch - 2ms/step
Epoch 4/5
3164/3164 - 5s - loss: 0.0143 - 5s/epoch - 2ms/step
Epoch 5/5
3164/3164 - 5s - loss: 0.0138 - 5s/epoch - 2ms/step
```



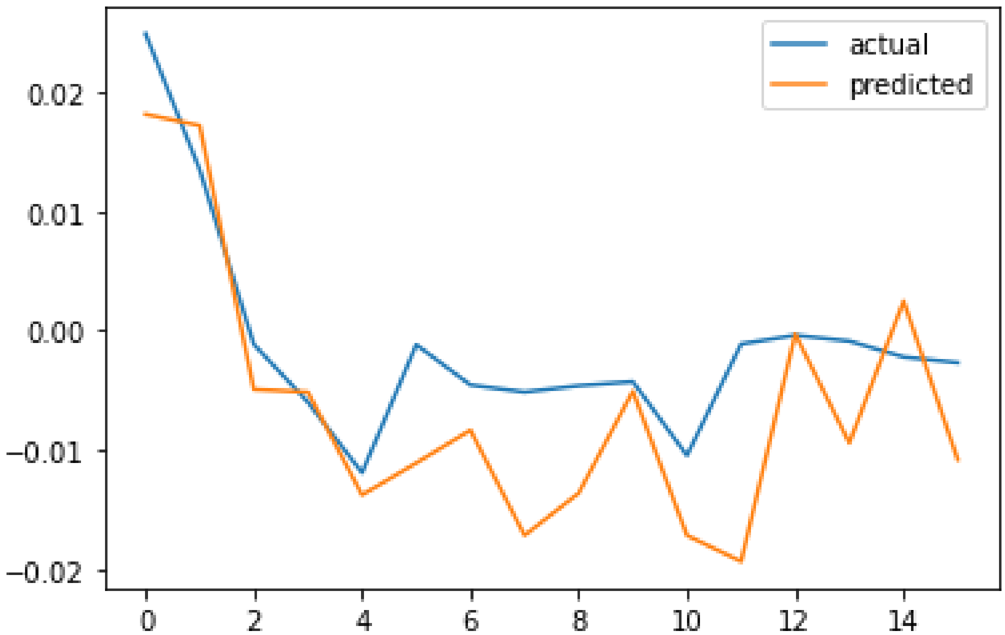

```
split_date: 2017-10-05 00:00:00
Epoch 1/5
3161/3161 - 6s - loss: 0.0244 - 6s/epoch - 2ms/step
Epoch 2/5
3161/3161 - 5s - loss: 0.0167 - 5s/epoch - 2ms/step
Epoch 3/5
3161/3161 - 5s - loss: 0.0151 - 5s/epoch - 2ms/step
Epoch 4/5
3161/3161 - 5s - loss: 0.0142 - 5s/epoch - 2ms/step
Epoch 5/5
3161/3161 - 5s - loss: 0.0139 - 5s/epoch - 2ms/step
```



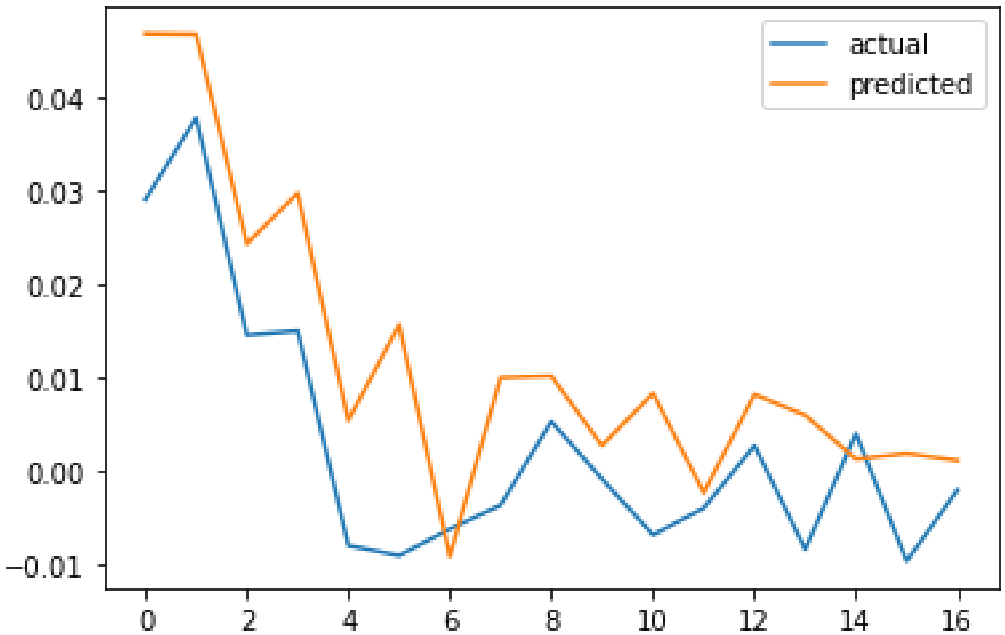

```
split_date: 2017-10-03 00:00:00
Epoch 1/5
3158/3158 - 6s - loss: 0.0248 - 6s/epoch - 2ms/step
Epoch 2/5
3158/3158 - 5s - loss: 0.0165 - 5s/epoch - 2ms/step
Epoch 3/5
3158/3158 - 5s - loss: 0.0147 - 5s/epoch - 2ms/step
Epoch 4/5
3158/3158 - 5s - loss: 0.0145 - 5s/epoch - 2ms/step
Epoch 5/5
3158/3158 - 6s - loss: 0.0140 - 6s/epoch - 2ms/step
```



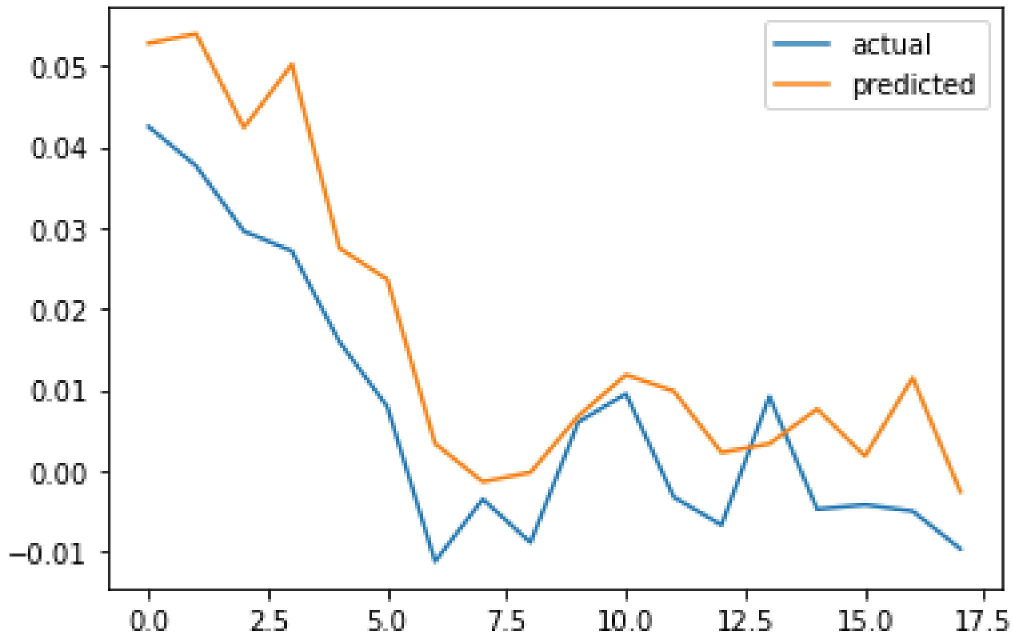

```
split_date: 2017-09-29 00:00:00
Epoch 1/5
3155/3155 - 6s - loss: 0.0237 - 6s/epoch - 2ms/step
Epoch 2/5
3155/3155 - 5s - loss: 0.0162 - 5s/epoch - 2ms/step
Epoch 3/5
3155/3155 - 6s - loss: 0.0153 - 6s/epoch - 2ms/step
Epoch 4/5
3155/3155 - 6s - loss: 0.0142 - 6s/epoch - 2ms/step
Epoch 5/5
3155/3155 - 5s - loss: 0.0142 - 5s/epoch - 2ms/step
```



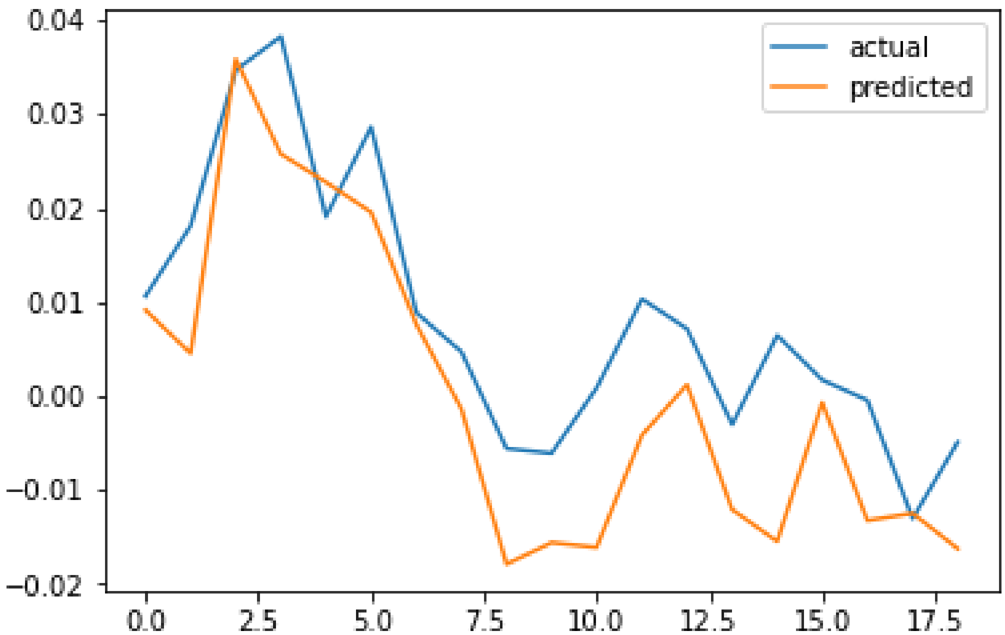

```
split_date: 2017-09-27 00:00:00
Epoch 1/5
3152/3152 - 6s - loss: 0.0250 - 6s/epoch - 2ms/step
Epoch 2/5
3152/3152 - 6s - loss: 0.0166 - 6s/epoch - 2ms/step
Epoch 3/5
3152/3152 - 6s - loss: 0.0149 - 6s/epoch - 2ms/step
Epoch 4/5
3152/3152 - 6s - loss: 0.0144 - 6s/epoch - 2ms/step
Epoch 5/5
3152/3152 - 6s - loss: 0.0141 - 6s/epoch - 2ms/step
```



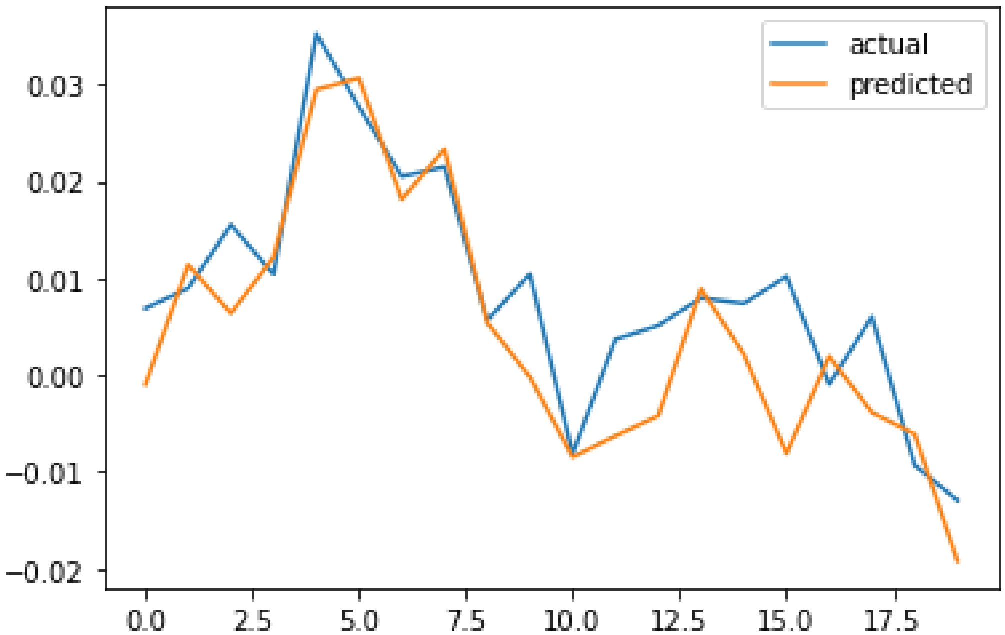

```
split_date: 2017-09-25 00:00:00
Epoch 1/5
3149/3149 - 7s - loss: 0.0253 - 7s/epoch - 2ms/step
Epoch 2/5
3149/3149 - 6s - loss: 0.0169 - 6s/epoch - 2ms/step
Epoch 3/5
3149/3149 - 6s - loss: 0.0154 - 6s/epoch - 2ms/step
Epoch 4/5
3149/3149 - 6s - loss: 0.0145 - 6s/epoch - 2ms/step
Epoch 5/5
3149/3149 - 6s - loss: 0.0143 - 6s/epoch - 2ms/step
```




```
split_date: 2017-09-21 00:00:00
Epoch 1/5
3146/3146 - 7s - loss: 0.0259 - 7s/epoch - 2ms/step
Epoch 2/5
3146/3146 - 6s - loss: 0.0174 - 6s/epoch - 2ms/step
Epoch 3/5
3146/3146 - 6s - loss: 0.0152 - 6s/epoch - 2ms/step
Epoch 4/5
3146/3146 - 6s - loss: 0.0148 - 6s/epoch - 2ms/step
Epoch 5/5
3146/3146 - 7s - loss: 0.0145 - 7s/epoch - 2ms/step
```



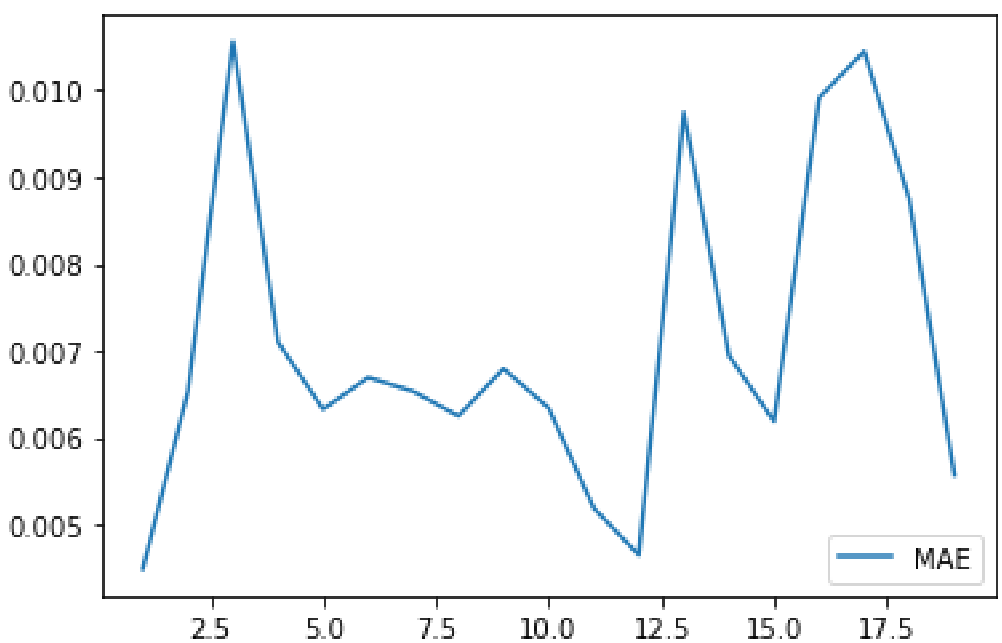

```
split_date: 2017-09-19 00:00:00
Epoch 1/5
3143/3143 - 7s - loss: 0.0254 - 7s/epoch - 2ms/step
Epoch 2/5
3143/3143 - 6s - loss: 0.0170 - 6s/epoch - 2ms/step
Epoch 3/5
3143/3143 - 6s - loss: 0.0153 - 6s/epoch - 2ms/step
Epoch 4/5
3143/3143 - 6s - loss: 0.0147 - 6s/epoch - 2ms/step
Epoch 5/5
3143/3143 - 6s - loss: 0.0143 - 6s/epoch - 2ms/step
```




```
[13]: print(window_result.loc[window_result["MAE"] == window_result["MAE"].min()])
      window_result.plot()
```

```
        MAE
1  0.004486
```

```
[13]: <AxesSubplot:>
```



The model parameter window denotes the number of days we use to trace back the prices, which hence represents the histrorical performance of the underlying asset.

It can be shown that using window = 1 would be the best choice with lowest mean absolute error, the result may due to the fact that, the price variation would not be so large between two consecutive trading days. We can observe from the plot that, the MAE firstly gets higher, then drops down and pulls up again.

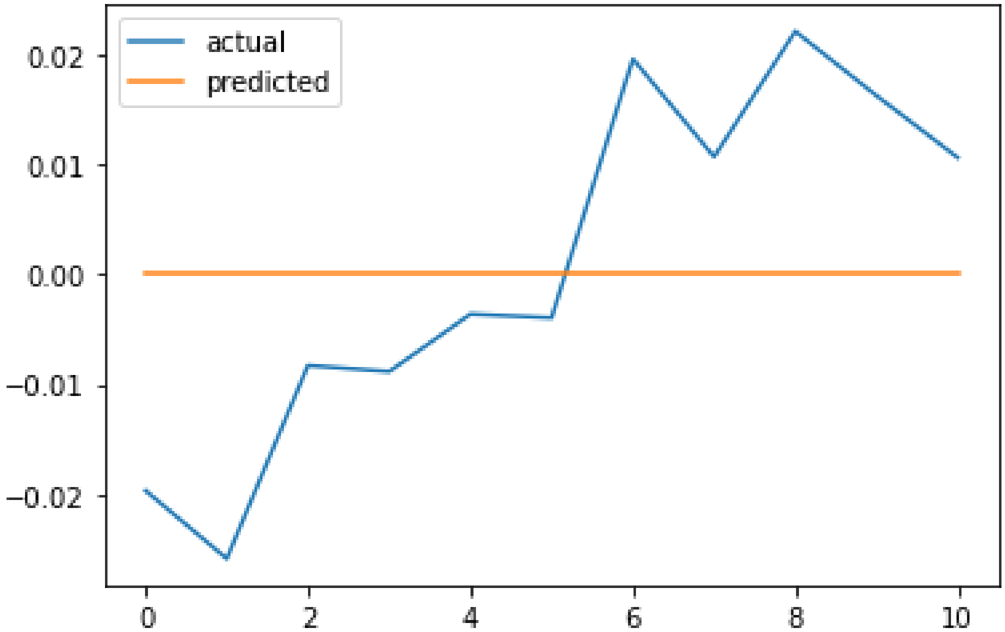
2.2 LSTM Parameter: activation function

[14]:
```
activation_lst = ["relu", "sigmoid", "softmax", "softplus", "softsign",
                  "tanh", "selu", "elu", "exponential", "linear"]
activation_dic = {}

window_len = 10
for activation in activation_lst:
    def this_build_model(inputs, output_size, neurons, activ_func=activation,
                         dropout=0.10, loss="mae", optimizer="adam"):
        model = Sequential()

        model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))

        model.compile(loss=loss, optimizer=optimizer)
        return model

    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)

    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
↪split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1
```

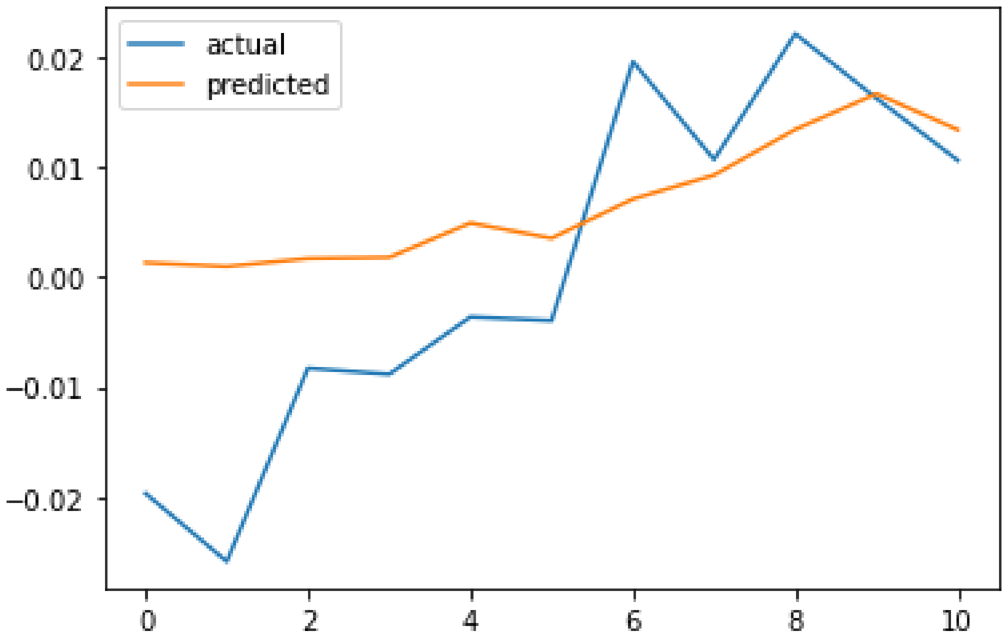
```
        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
    LSTM_training_outputs = (training_set['Close'][window_len:].values/
↪training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for␣
↪LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/
↪test_set['Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
↪LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)

    # initialise model architecture
    nn_model = this_build_model(LSTM_training_inputs, output_size=1, neurons =␣
↪32)
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                            epochs=5, batch_size=1, verbose=2, shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    MAE = mean_absolute_error(LSTM_test_outputs, nn_model.
↪predict(LSTM_test_inputs))
    activation_dic[activation] = MAE
activation_result = pd.DataFrame(activation_dic.values(), activation_dic.
↪keys()).rename(columns={0: "MAE"})
```

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0365 - 5s/epoch - 2ms/step
```

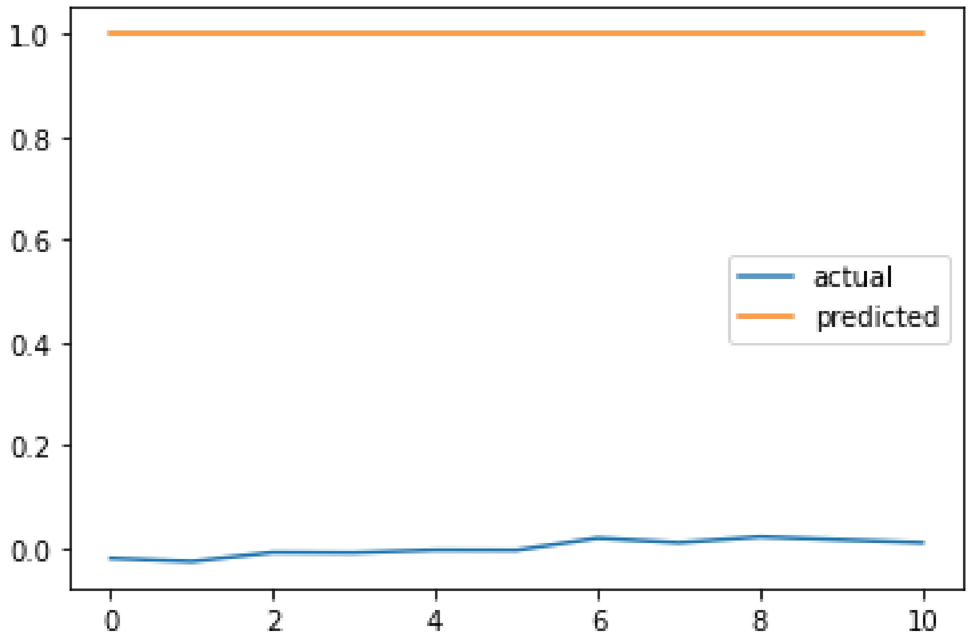
```
Epoch 2/5
3170/3170 - 4s - loss: 0.0364 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0364 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0364 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0364 - 4s/epoch - 1ms/step
```



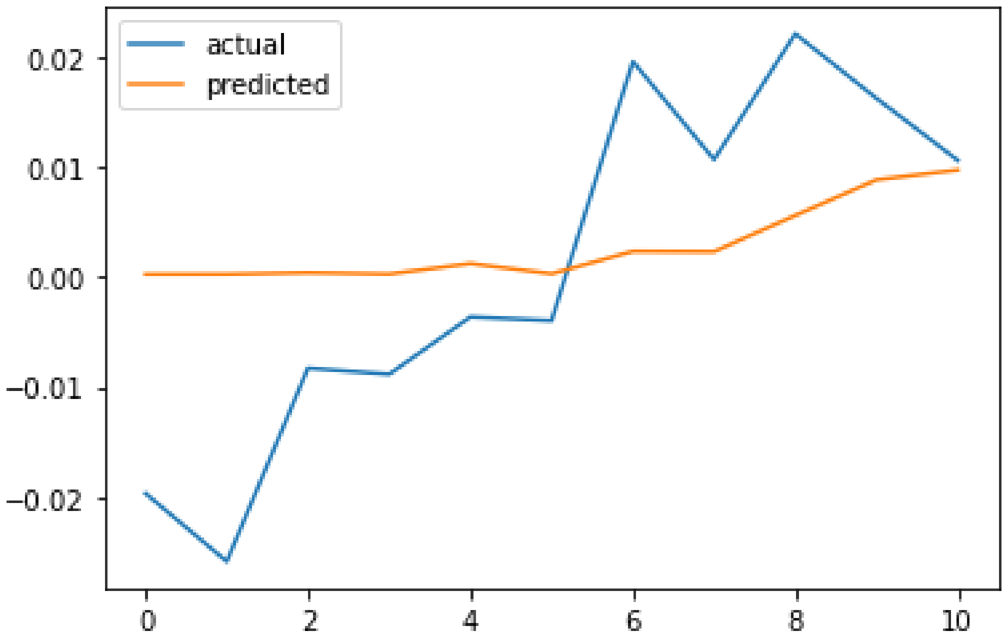

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0418 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0285 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0269 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0260 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0257 - 4s/epoch - 1ms/step
```



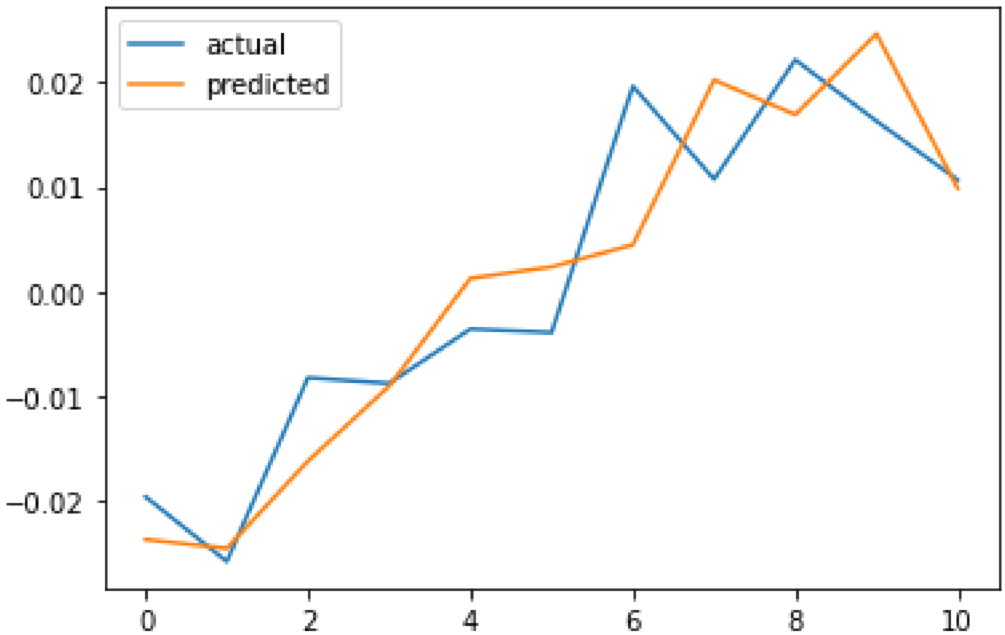

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.9962 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.9962 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.9962 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.9962 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.9962 - 4s/epoch - 1ms/step
```



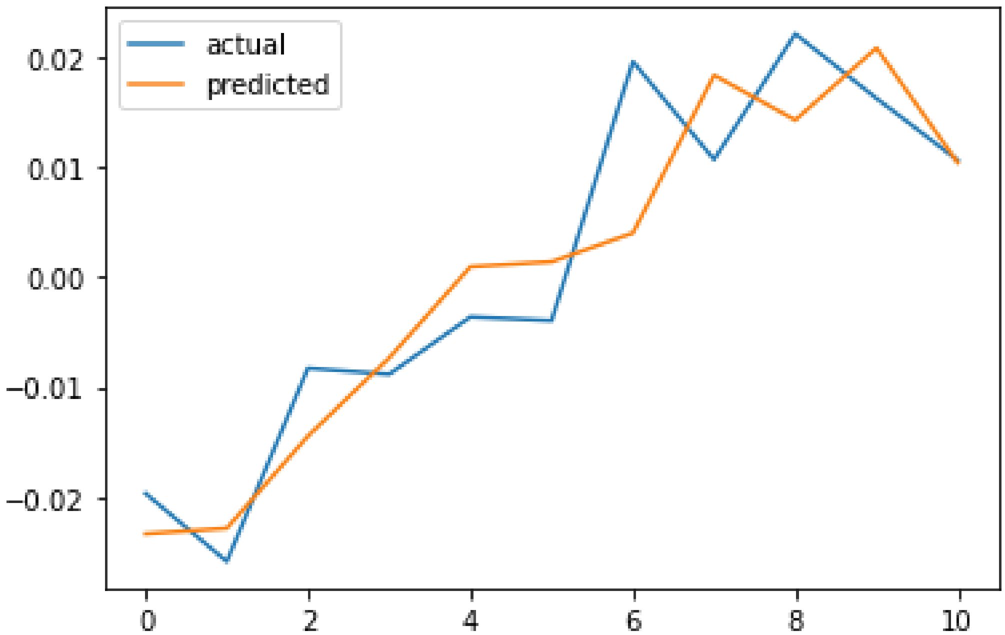

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0444 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0295 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0273 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0266 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0259 - 4s/epoch - 1ms/step
```



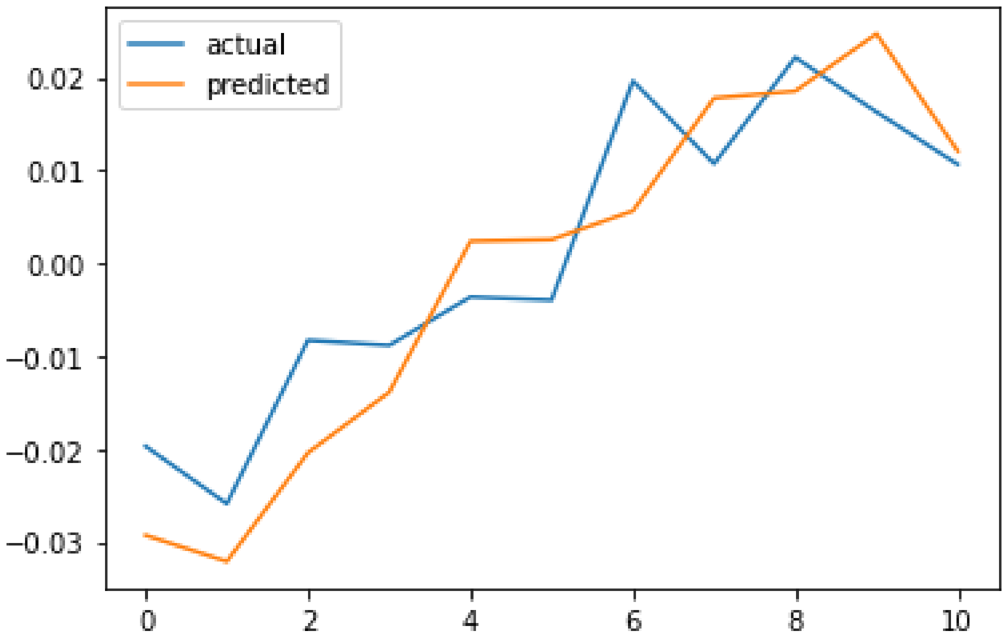

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0227 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0159 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0137 - 4s/epoch - 1ms/step
```



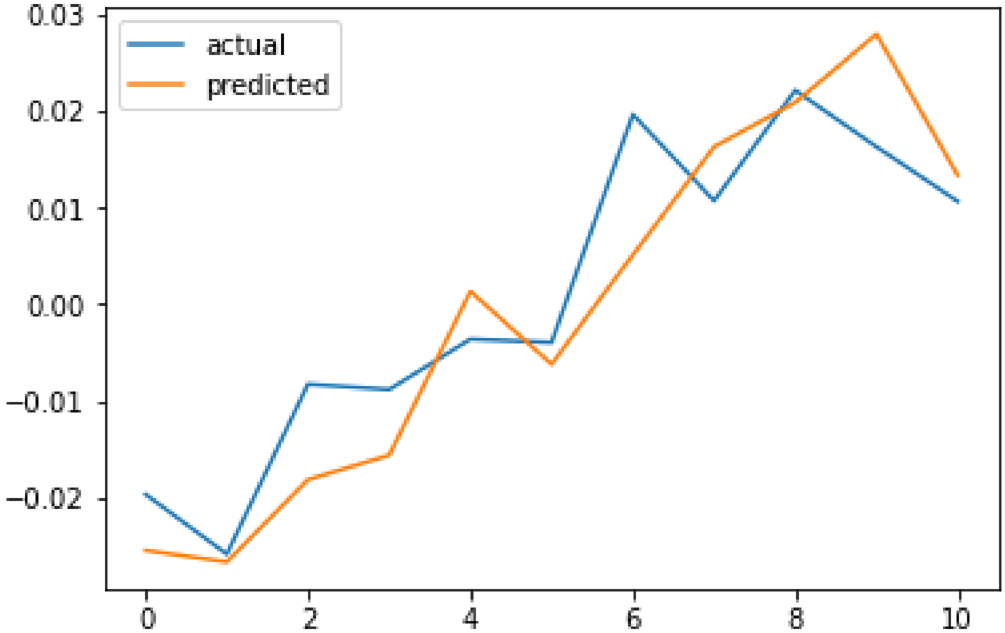

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0232 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0158 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



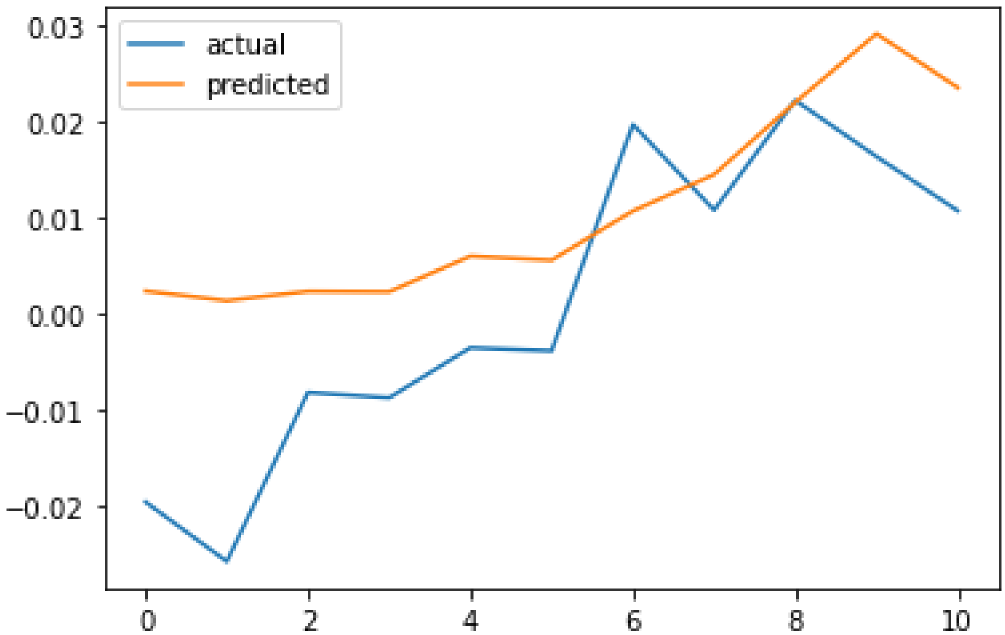

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0262 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0173 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0156 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0147 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
```



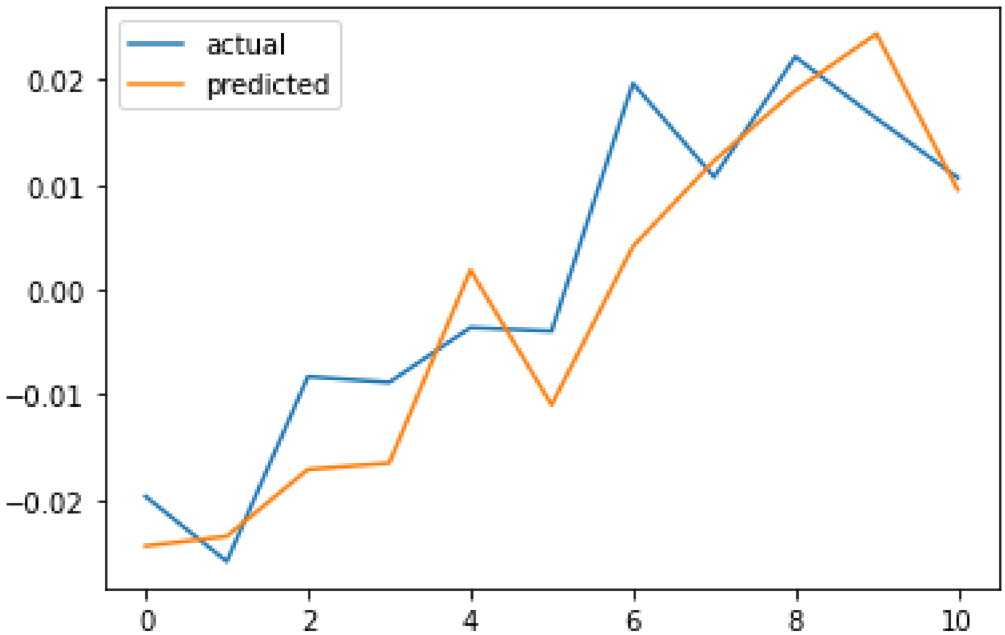

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0230 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0157 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0497 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0299 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0275 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0268 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0264 - 4s/epoch - 1ms/step
```



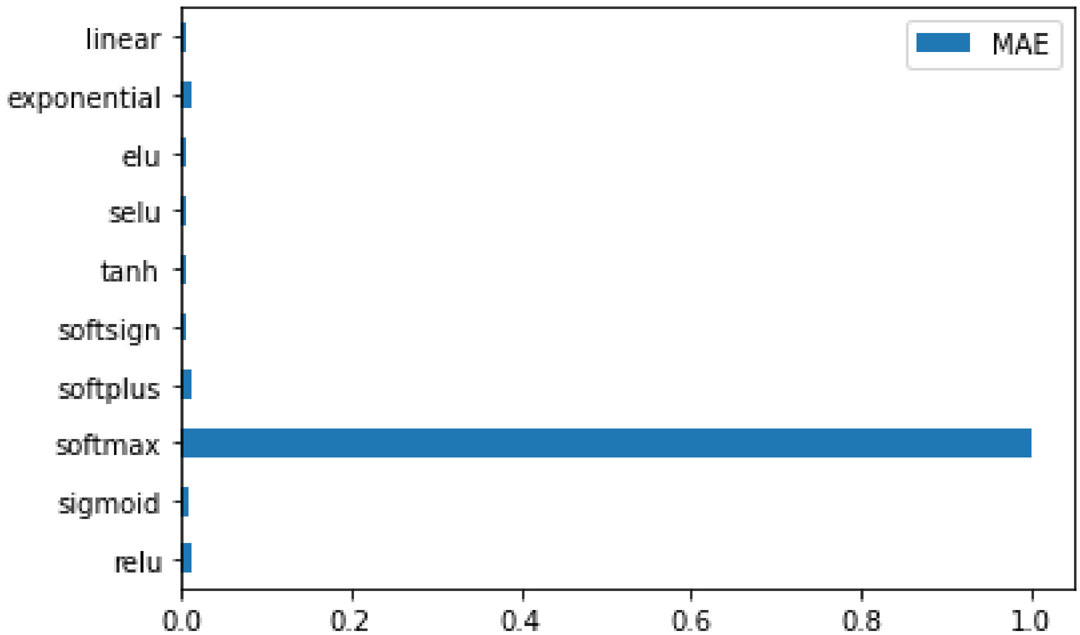

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0244 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0161 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



[15]:
```
print(activation_result.loc[activation_result["MAE"] ==␣
 ↪activation_result["MAE"].min()])
activation_result.plot(kind="barh")
```

```
           MAE
tanh  0.005479
```

[15]: <AxesSubplot:>



The LSTM parameter activation function is the function we use in every hidden layer node in the model. That is, for each node in our model, the value would be give by $\sigma(w \cdot x + b)$, where σ is the activation function. If we use no activation function, the defualt set would be linear. However, a specified activation function type except linear function would introduce non-lineartiy into the model.

We can observe that the `tanh` activation function has the lowest error, however, it doesn't vary a lot from other activation functions except for `softmax`, which has significantly higher error than other activation functions. This is becuase `softmax` function is a probabilistic function, and the vector returned by it would sum up to 1.

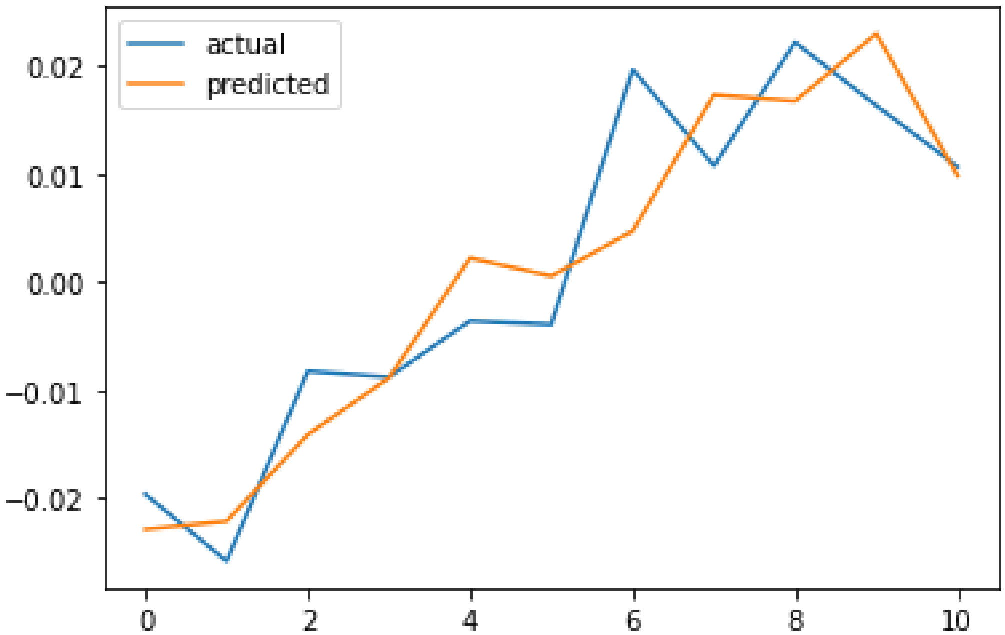
2.3 LSTM Parameter: loss function

[16]:
```
loss_lst = ["mse", "mae", "mape", "msle", "huber"]
loss_dic = {}

window_len = 10
for loss in loss_lst:
    def this_build_model(inputs, output_size, neurons, activ_func="linear",
                         dropout=0.10, loss=loss, optimizer="adam"):
        model = Sequential()

        model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))
```

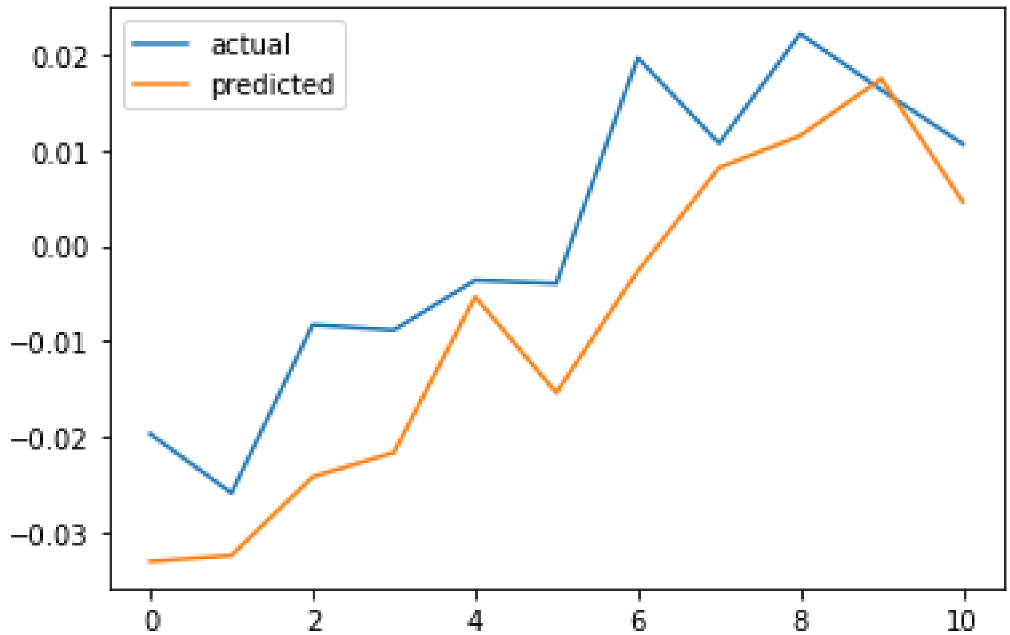
```
        model.compile(loss=loss, optimizer=optimizer)
        return model

    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)

    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >= split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
    LSTM_training_outputs = (training_set['Close'][window_len:].values/training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/test_set['Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)
```

```
    # initialise model architecture
    nn_model = this_build_model(LSTM_training_inputs, output_size=1, neurons =␣
↪32)
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                                epochs=5, batch_size=1, verbose=2, shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    Error = mean_absolute_error(LSTM_test_outputs, nn_model.
↪predict(LSTM_test_inputs))
    loss_dic[loss] = Error
loss_result = pd.DataFrame(loss_dic.values(), loss_dic.keys()).
↪rename(columns={0: "Error"})
```

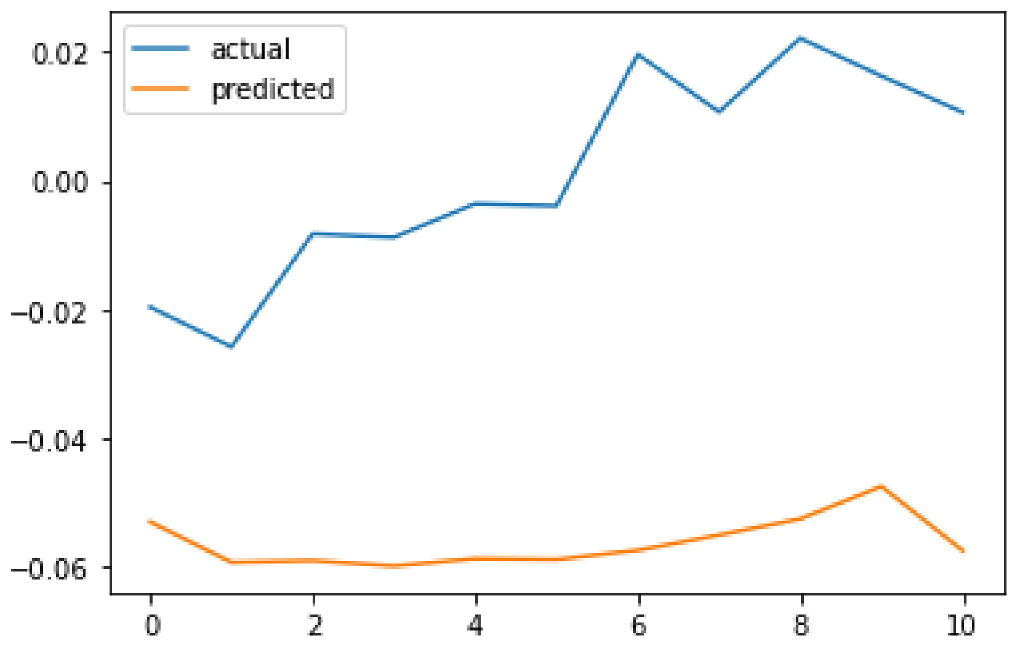
```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0013 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 5.4588e-04 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 4.5618e-04 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 4.3707e-04 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 4.2026e-04 - 4s/epoch - 1ms/step
```



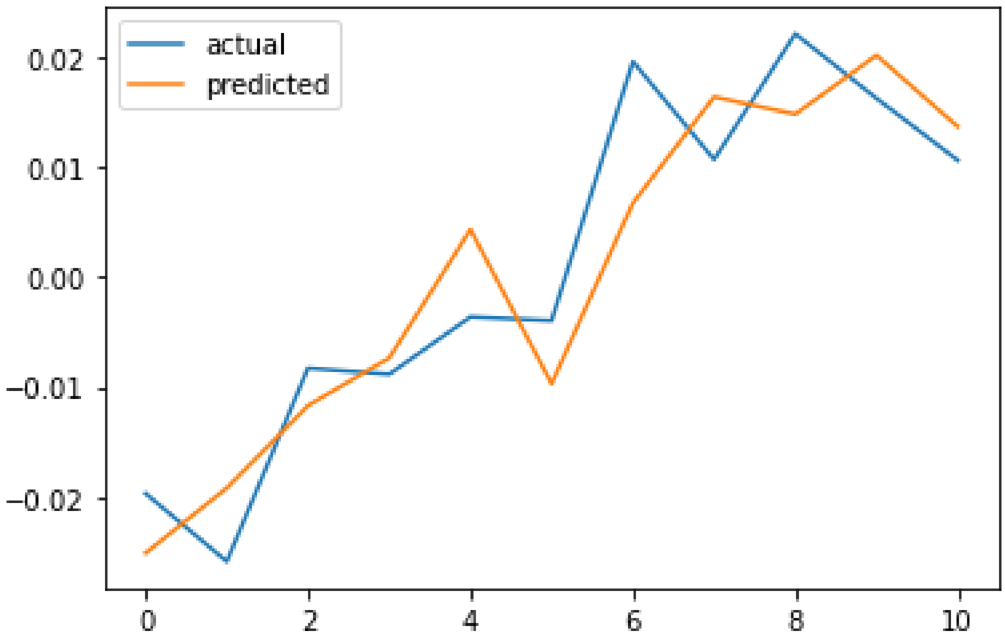

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0229 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0160 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



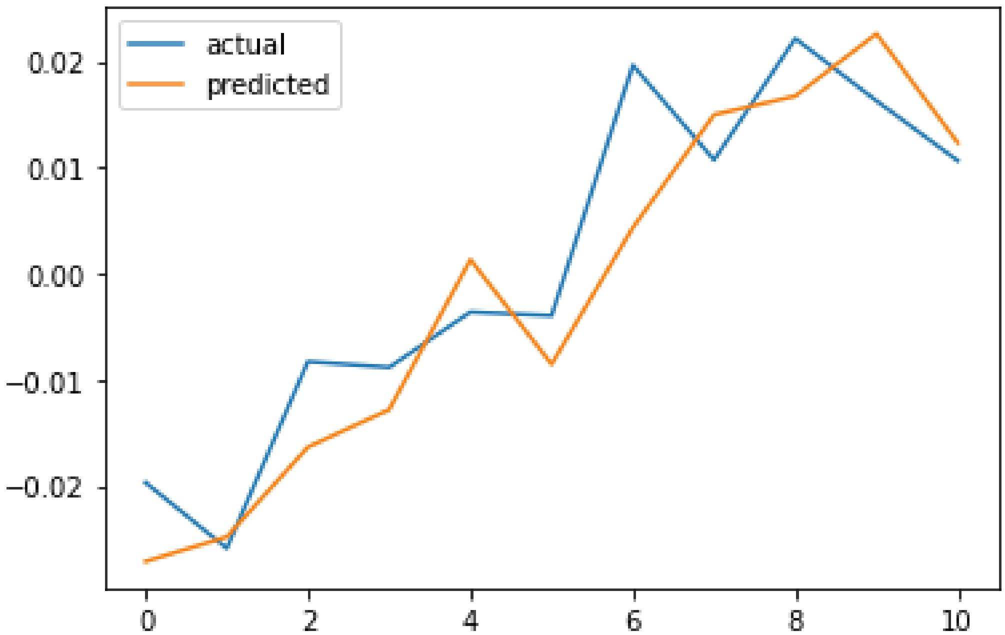

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 153750.9375 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 167885.2656 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 135851.4844 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 163576.8906 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 143367.5156 - 4s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 7.2380e-04 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 3.3209e-04 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 2.2228e-04 - 5s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 1.8553e-04 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 1.6499e-04 - 4s/epoch - 1ms/step
```



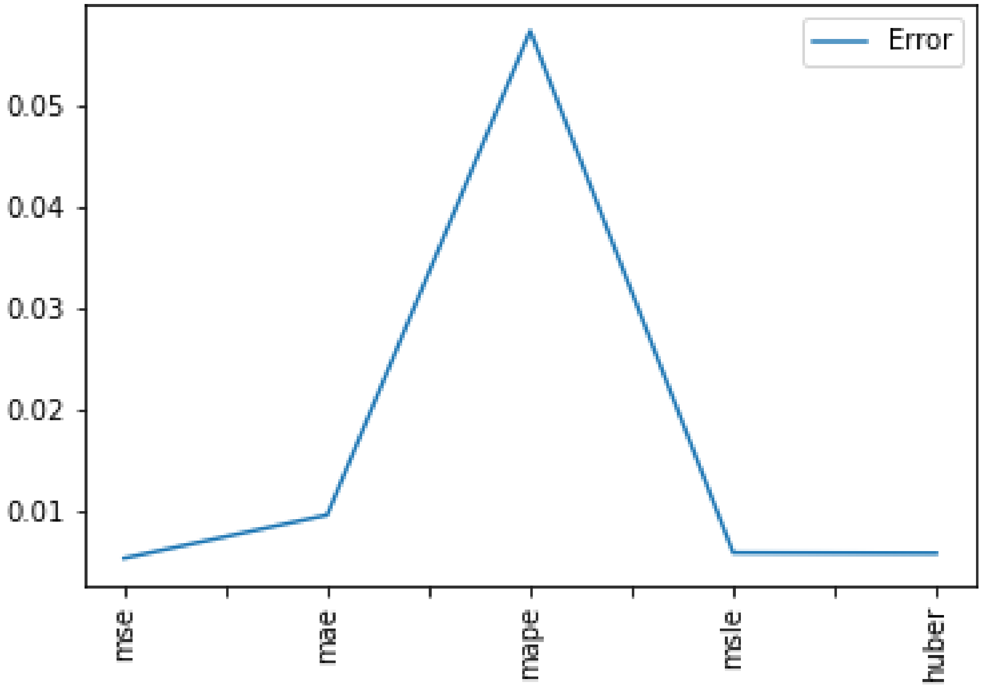

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0011 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 3.6827e-04 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 2.6161e-04 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 2.1965e-04 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 5s - loss: 2.0357e-04 - 5s/epoch - 1ms/step
```



[17]:
```
print(loss_result.loc[loss_result["Error"] == loss_result["Error"].min()])
loss_result.plot(rot=90)
```

```
       Error
mse  0.005243
```

[17]: <AxesSubplot:>



The loss function is the function that the model would seek to minimize during training. Here, the `mse` (Mean Squared Error) function would give us the lowest error value, however, this is becuase we used different measurement.

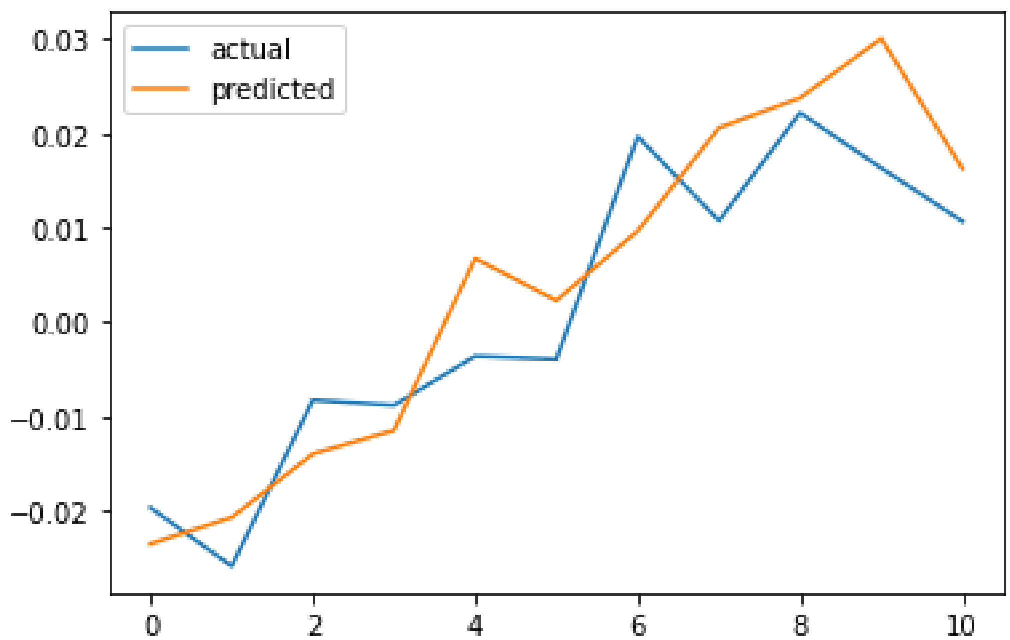
2.4 LSTM Parameter: dropout rate

[18]:
```
dropout_lst = np.arange(0, 1, 0.05)
dropout_dic = {}

for dropout in dropout_lst:
    def this_build_model(inputs, output_size, neurons, activ_func="linear",
                         dropout=dropout, loss="mae", optimizer="adam"):
        model = Sequential()

        model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))

        model.compile(loss="mae", optimizer=optimizer)
        return model

    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)
```

```
    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
↪split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
    LSTM_training_outputs = (training_set['Close'][window_len:].values/
↪training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for␣
↪LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/
↪test_set['Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
↪LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)

    # initialise model architecture
    nn_model = this_build_model(LSTM_training_inputs, output_size=1, neurons =␣
↪32)
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
```

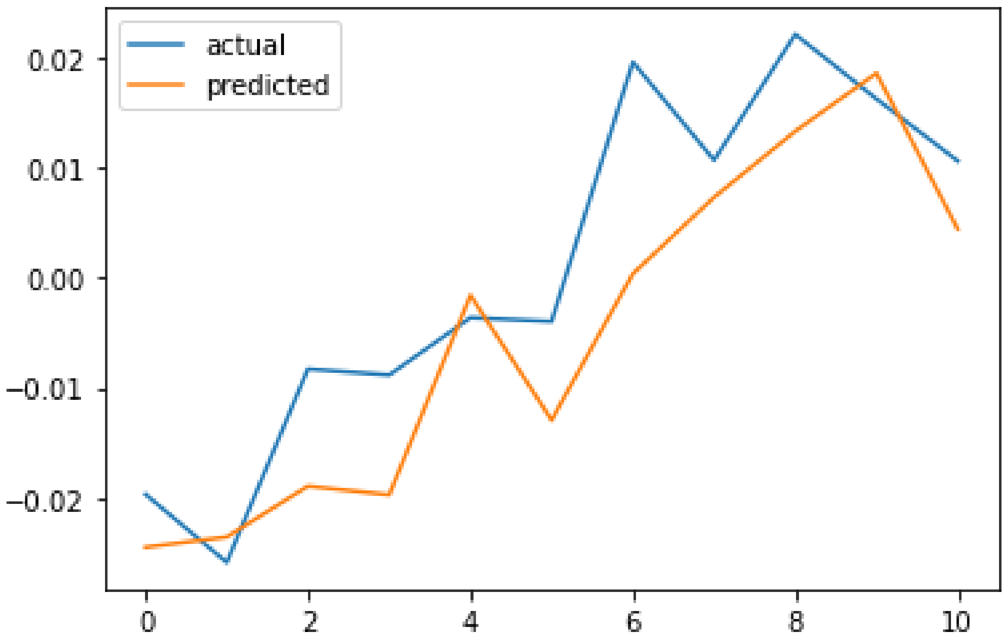
```
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                                epochs=5, batch_size=1, verbose=2, shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    MAE = mean_absolute_error(LSTM_test_outputs, nn_model.
 ↪predict(LSTM_test_inputs))
    dropout_dic[dropout] = MAE
dropout_result = pd.DataFrame(dropout_dic.values(), dropout_dic.keys()).
 ↪rename(columns={0: "MAE"})
```

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0200 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0157 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0146 - 5s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0141 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0137 - 4s/epoch - 1ms/step
```

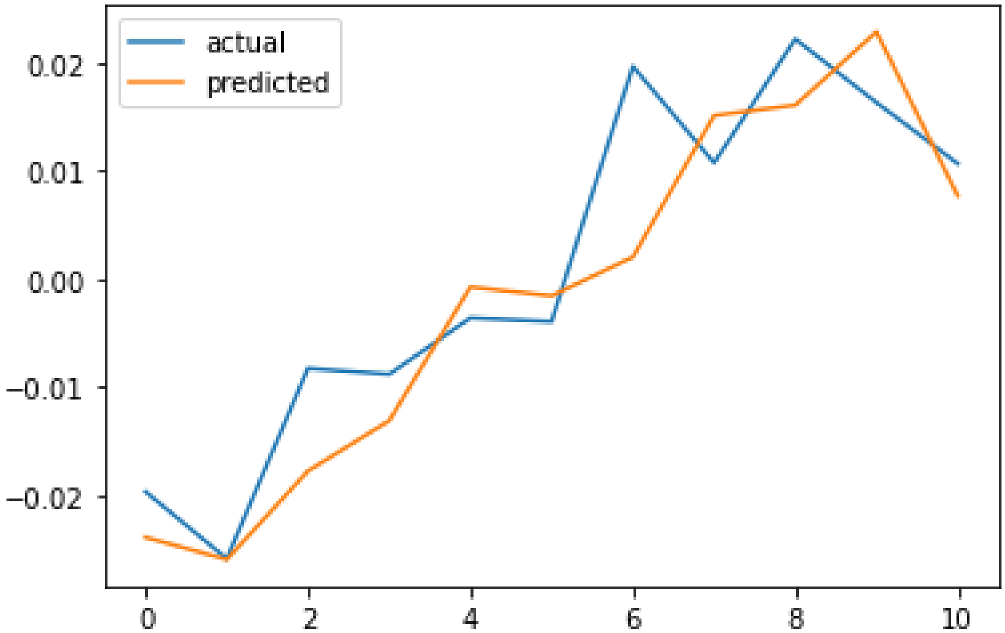



```
split_date: 2017-10-13 00:00:00
```

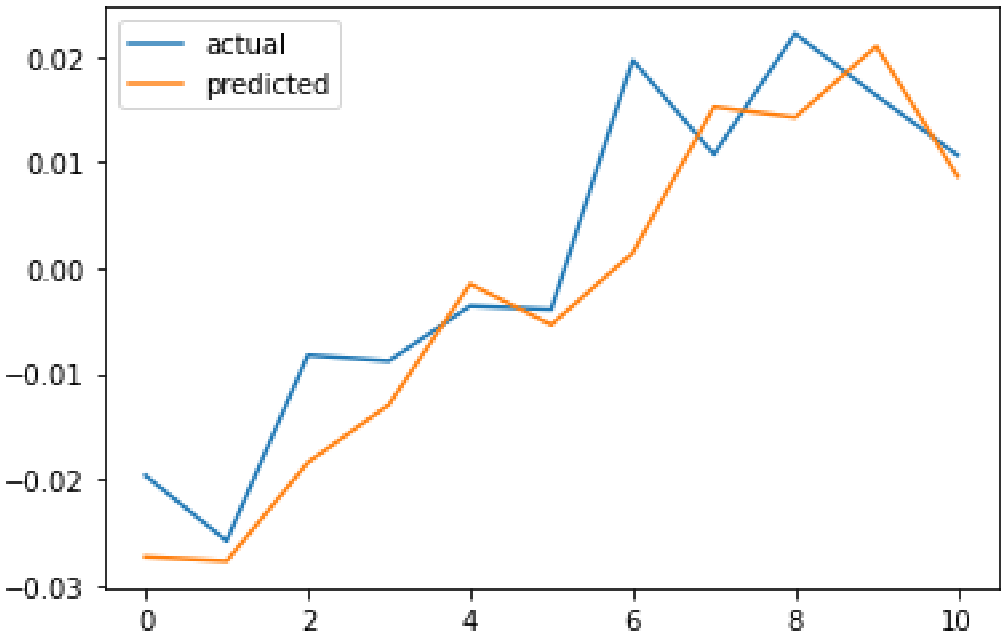
```
Epoch 1/5
3170/3170 - 5s - loss: 0.0216 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0159 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0137 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0135 - 4s/epoch - 1ms/step
```



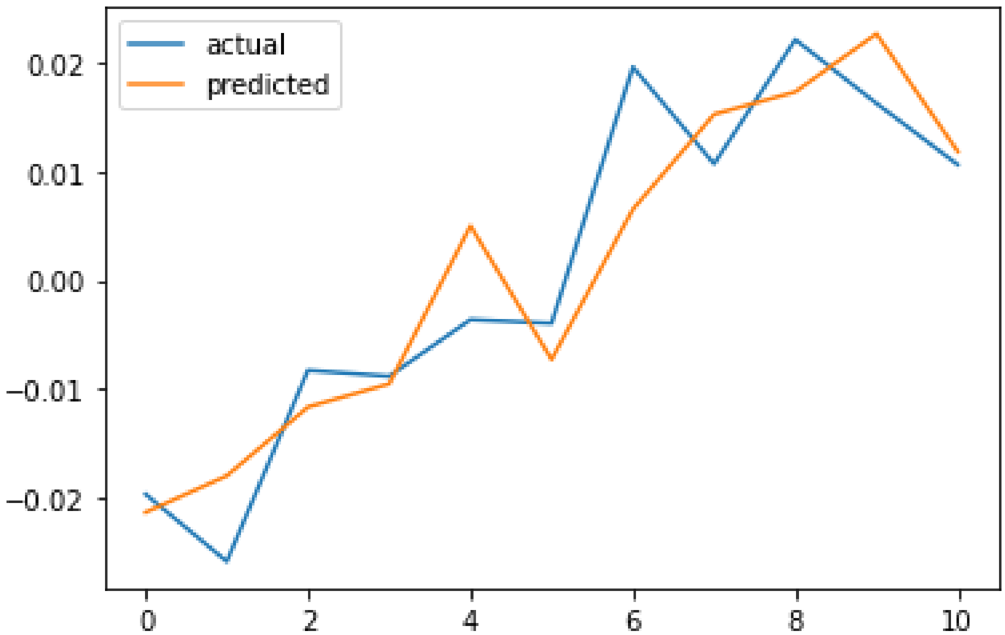

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0269 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0166 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0149 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0138 - 5s/epoch - 1ms/step
```



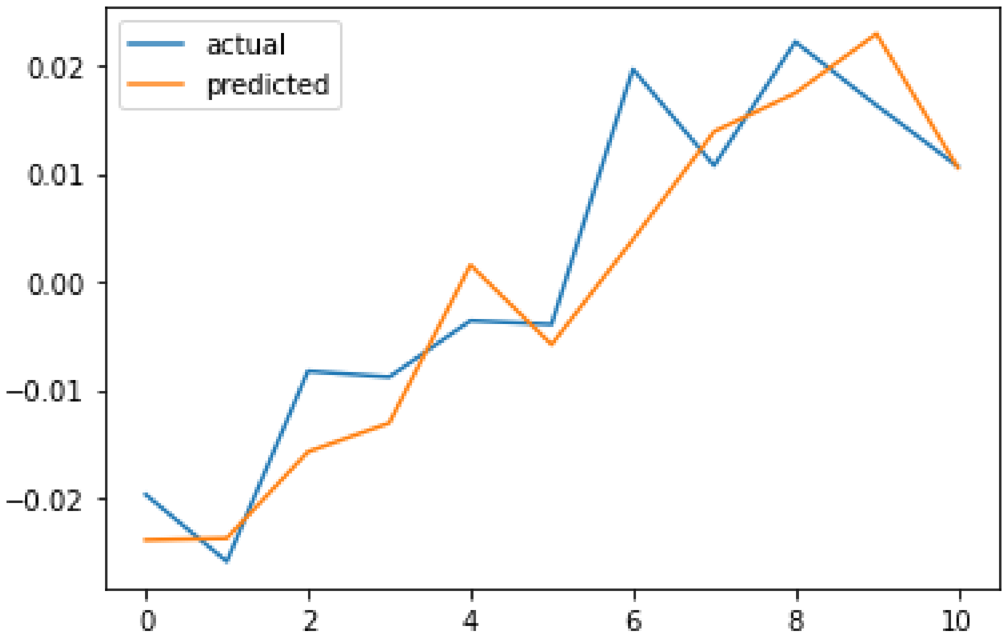

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0254 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0165 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0149 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0142 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
```



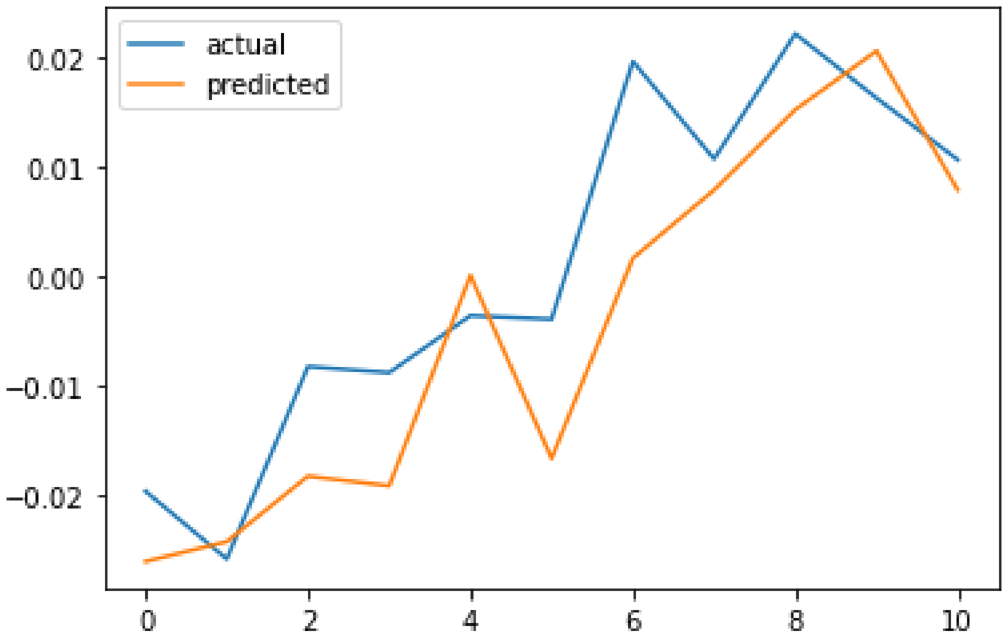

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0285 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0165 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0150 - 5s/epoch - 2ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0142 - 4s/epoch - 1ms/step
```



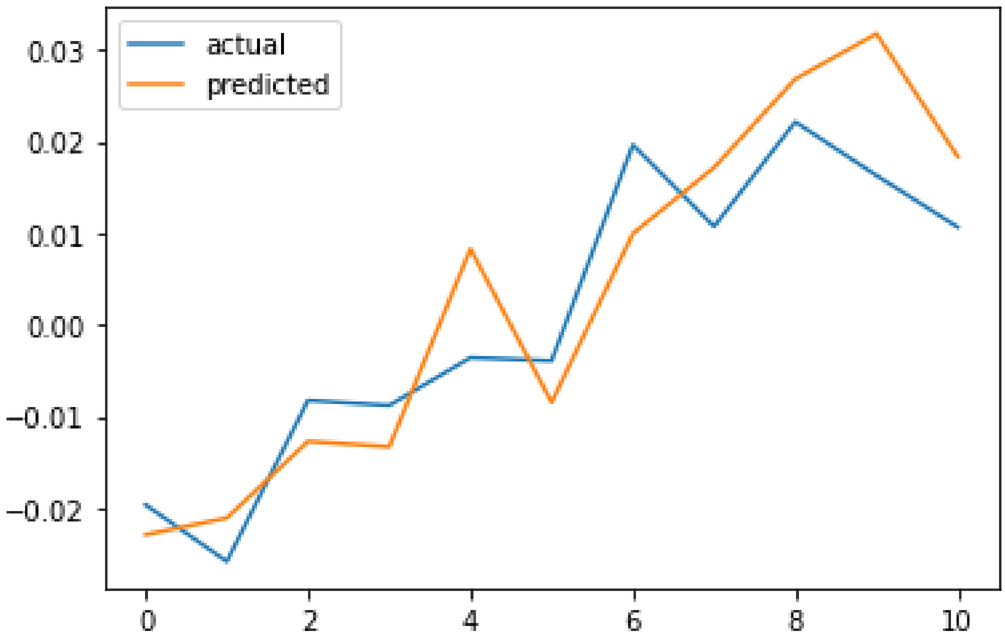

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0262 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0166 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0152 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0147 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
```



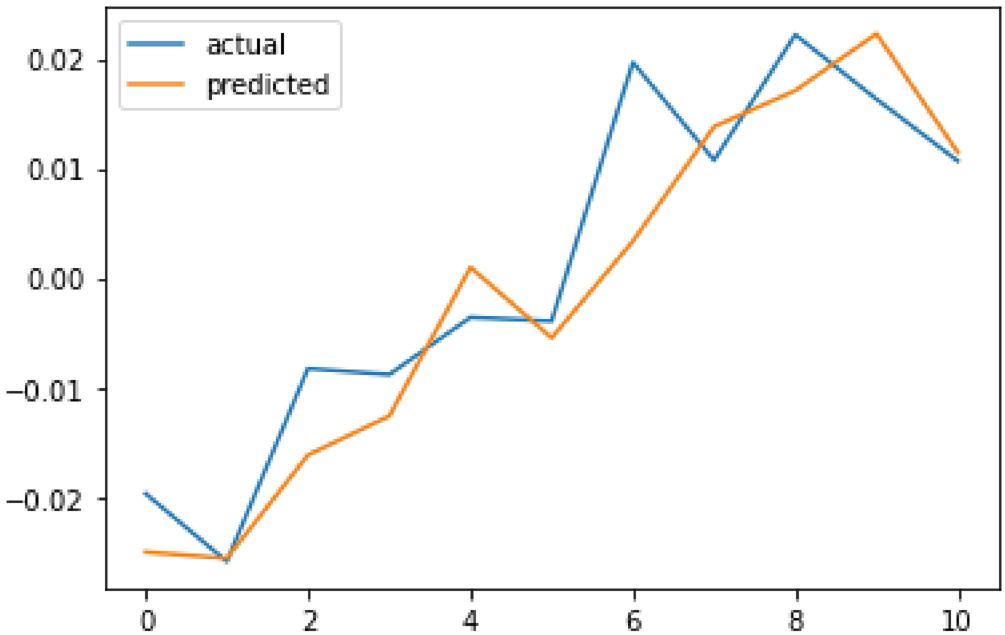

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0255 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0167 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0155 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0149 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
```



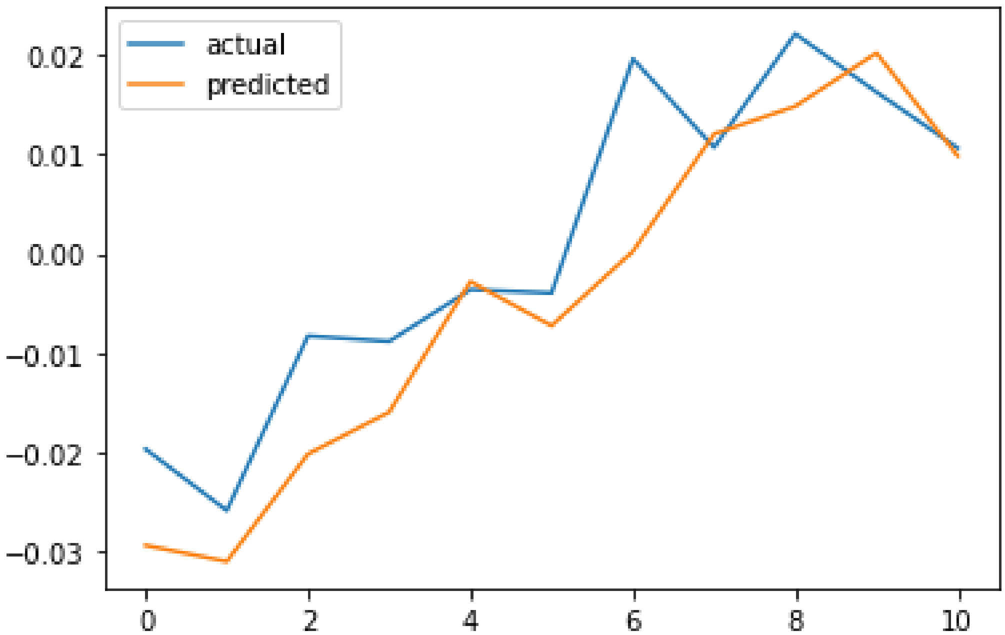

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0257 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0168 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0157 - 5s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0151 - 5s/epoch - 2ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0148 - 5s/epoch - 2ms/step
```



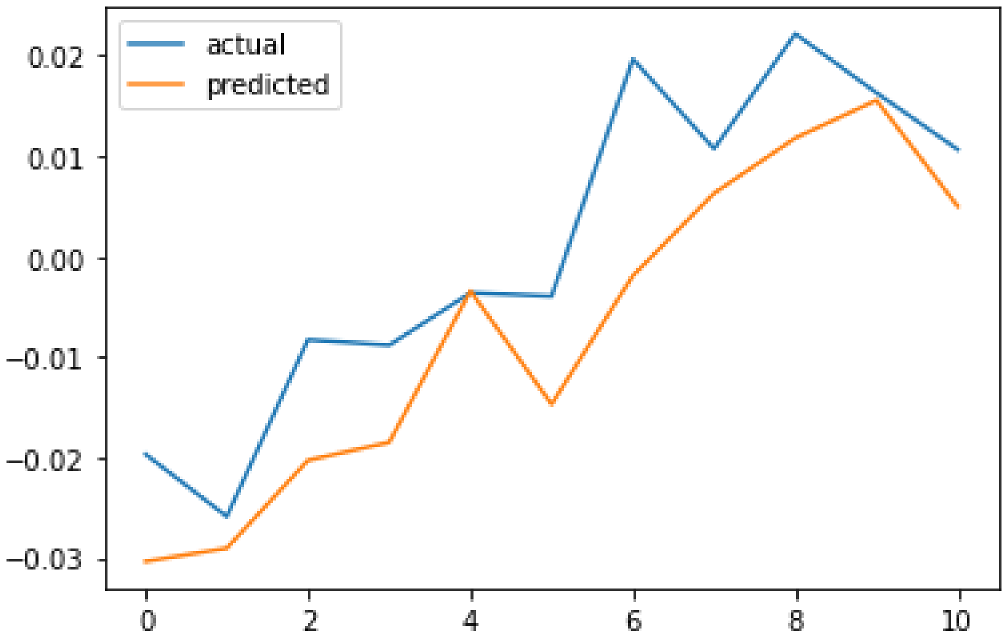

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0318 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0174 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0163 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0155 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0151 - 4s/epoch - 1ms/step
```



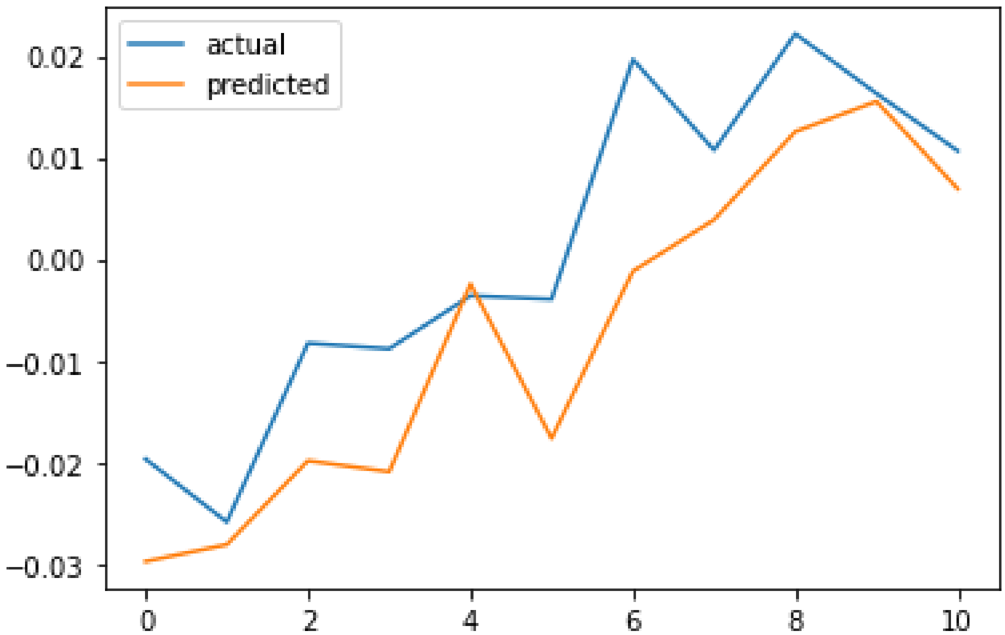

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0306 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0176 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0165 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0155 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0155 - 4s/epoch - 1ms/step
```



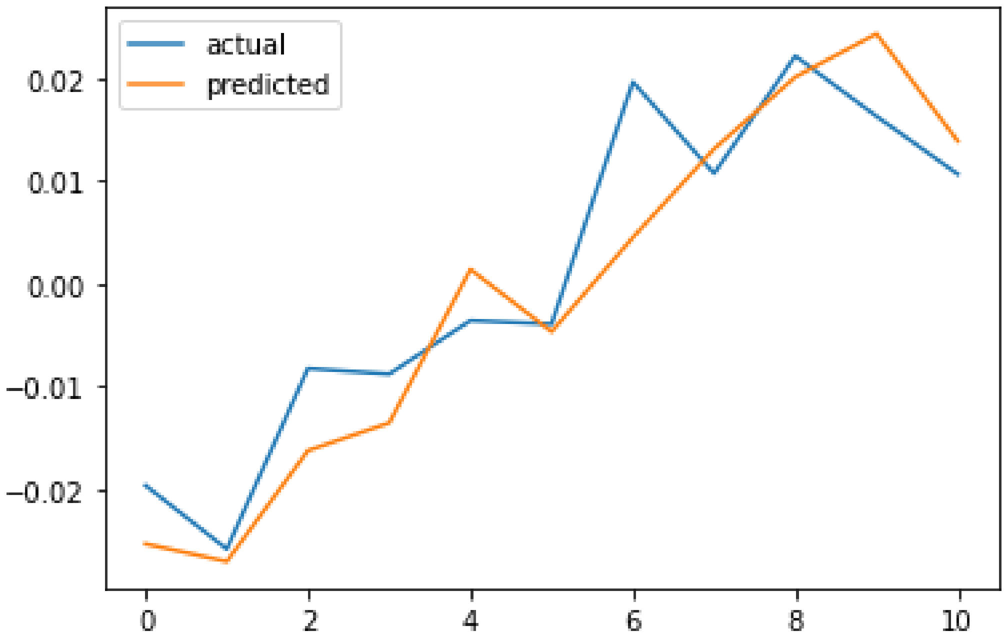

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0307 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0182 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0169 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0163 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0158 - 4s/epoch - 1ms/step
```



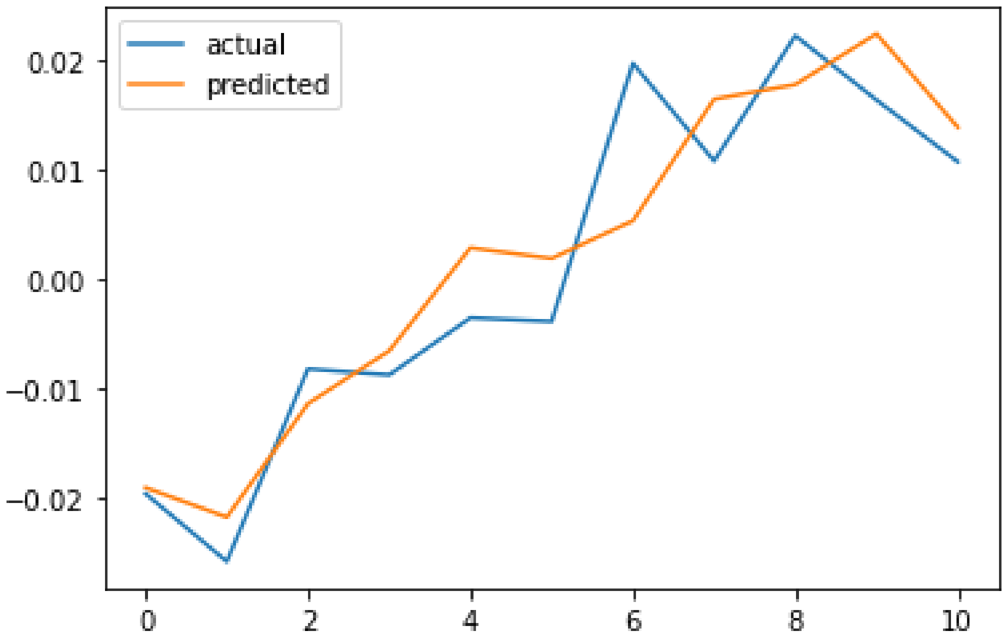

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0312 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0185 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0173 - 5s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0166 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0161 - 4s/epoch - 1ms/step
```



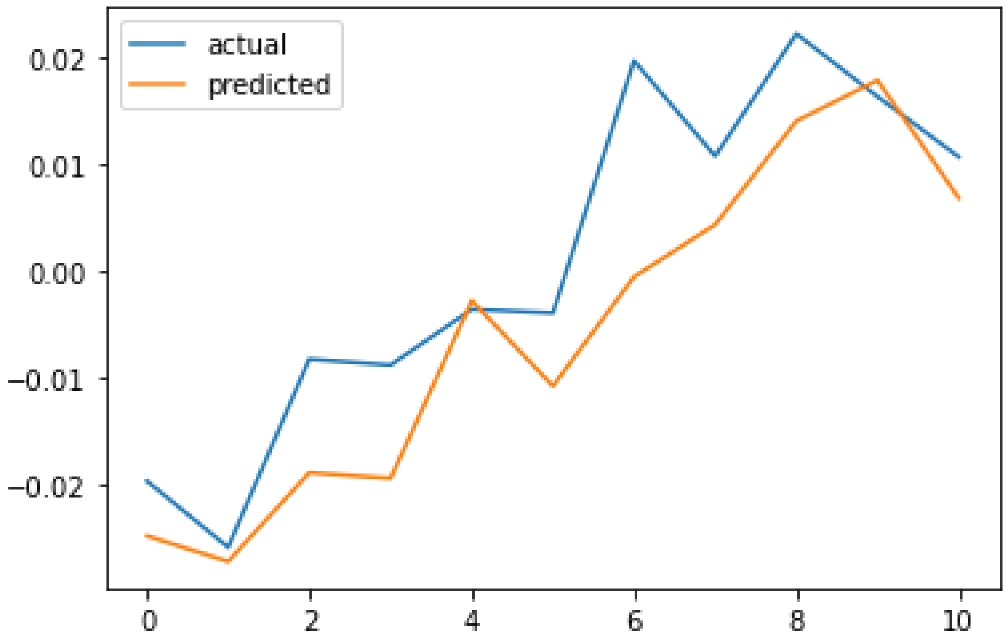

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0407 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0196 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0182 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0171 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0164 - 4s/epoch - 1ms/step
```



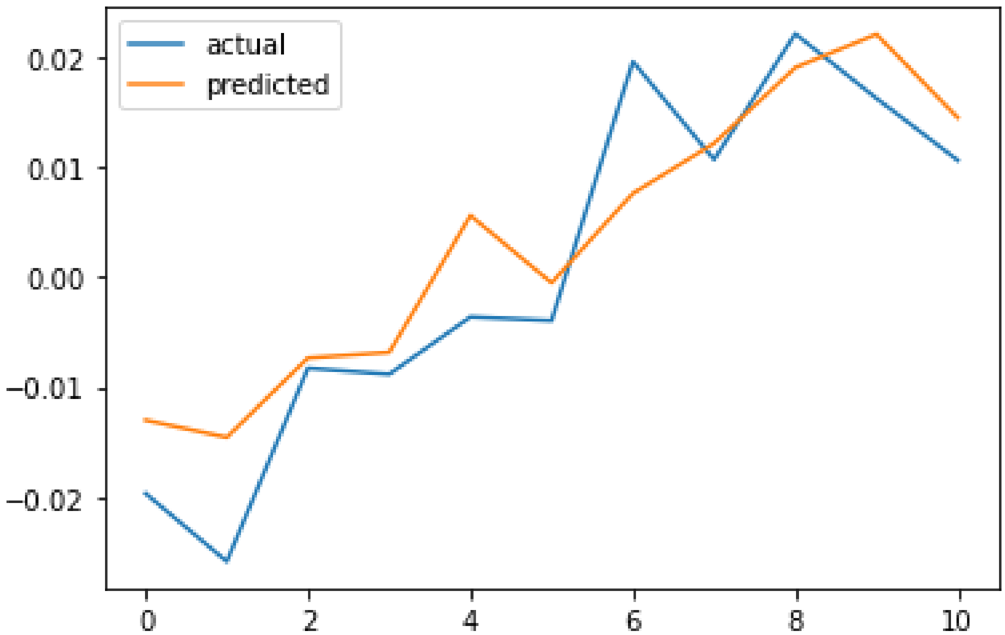

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0338 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0201 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0182 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0174 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0172 - 5s/epoch - 1ms/step
```



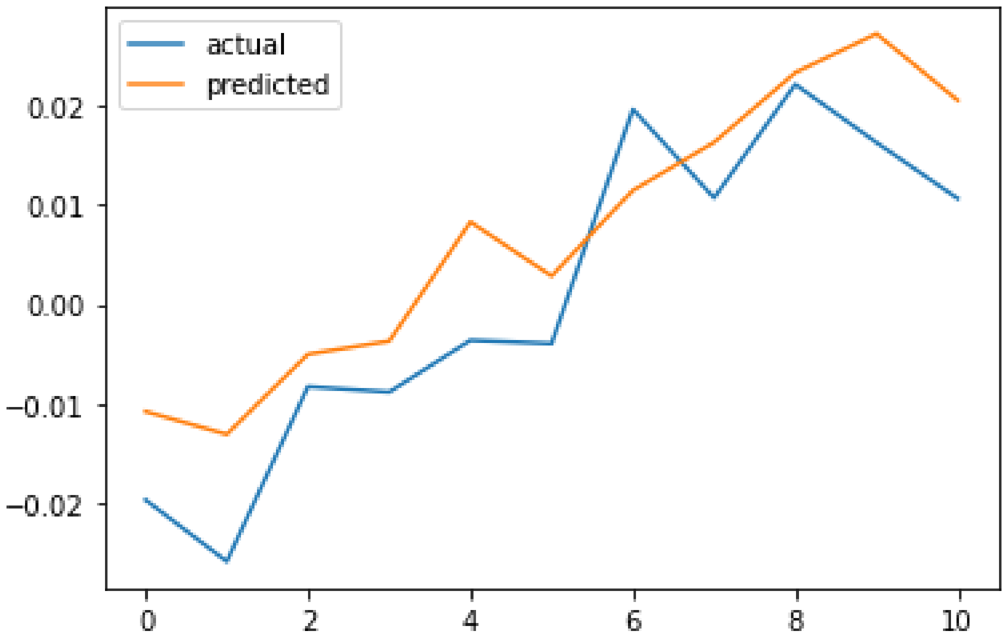

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0383 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0208 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0188 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0180 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0178 - 4s/epoch - 1ms/step
```



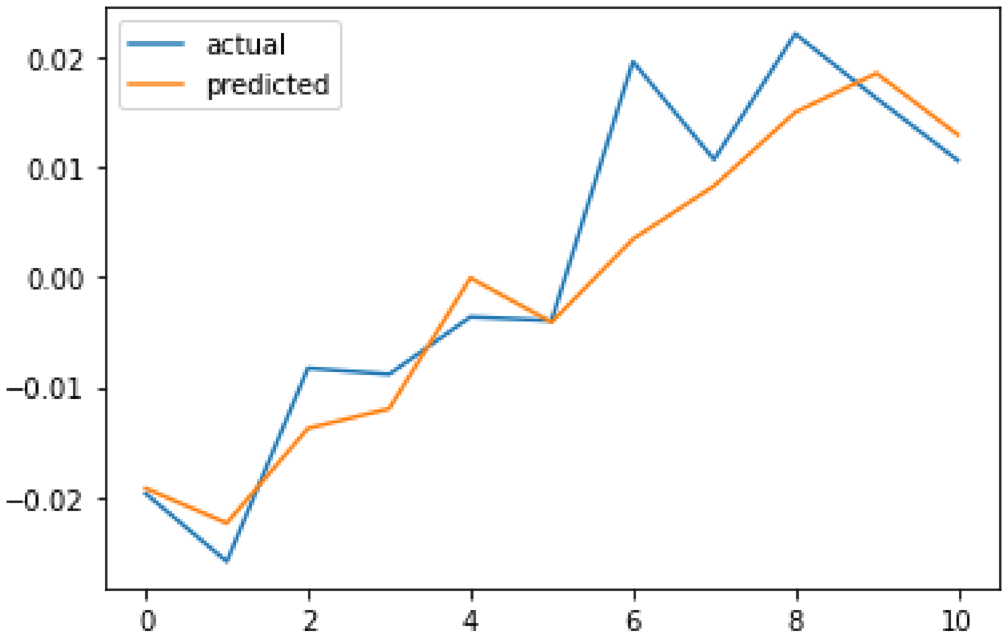

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0433 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0218 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0201 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0192 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0184 - 4s/epoch - 1ms/step
```



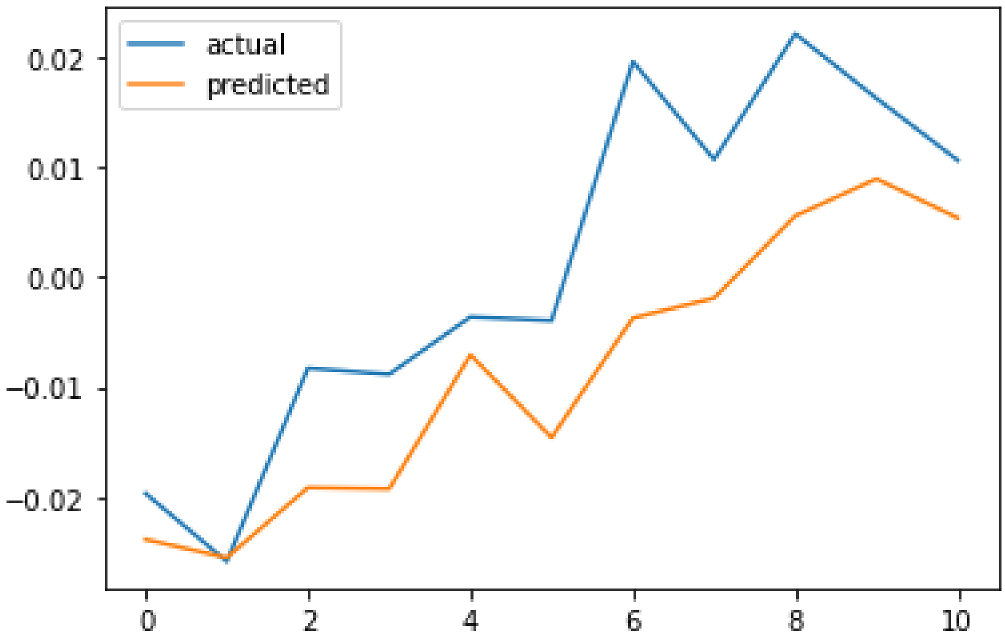

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0512 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0232 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0212 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0202 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0199 - 4s/epoch - 1ms/step
```



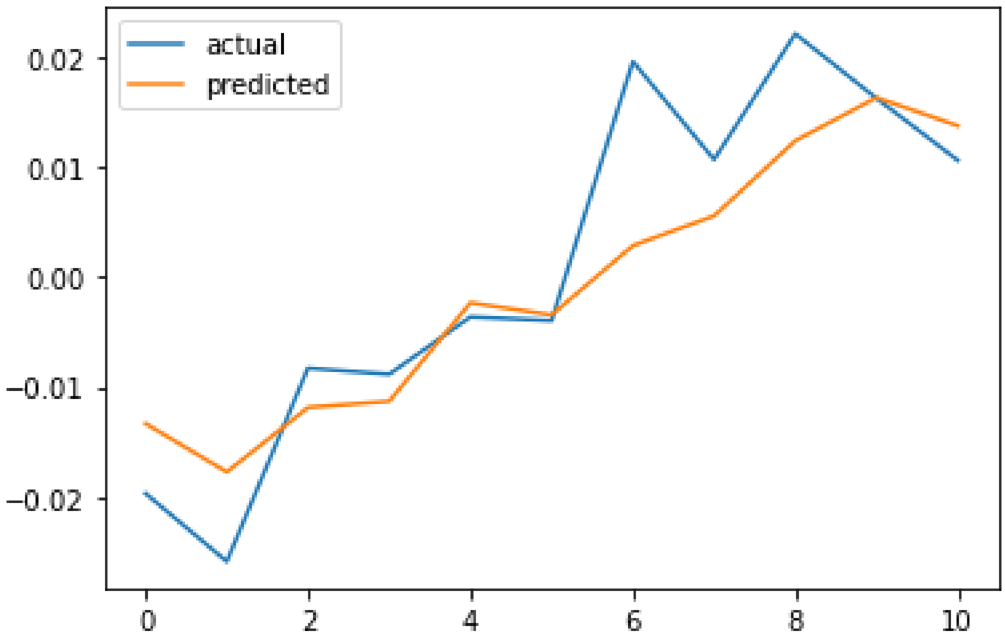

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0456 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0246 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0225 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0212 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0212 - 4s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0508 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0255 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0241 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0236 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0230 - 4s/epoch - 1ms/step
```



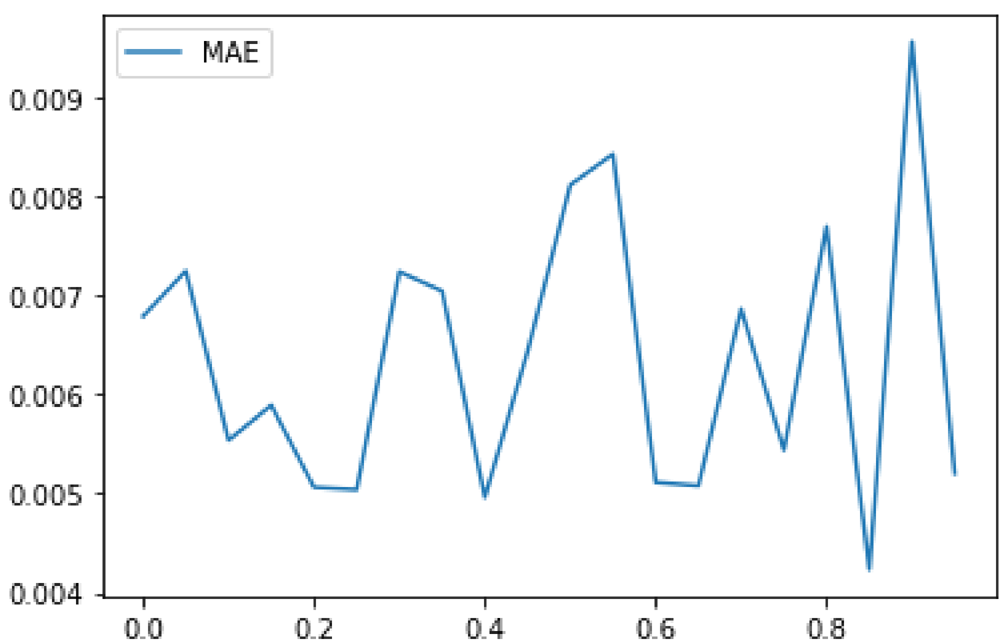

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0789 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0304 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0281 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0279 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0271 - 4s/epoch - 1ms/step
```



[19]:
```
print(dropout_result.loc[dropout_result["MAE"] == dropout_result["MAE"].min()])
dropout_result.plot()
```

```
           MAE
0.85  0.004235
```

[19]: <AxesSubplot:>



Dropout regularization is the fraction that we randomly "dropping out" unit activations in a single network for each single gradient step, the higher the dropout is, the stronger the regularization the it performs.

From the output, we can observe that with the increase of the dropout, the error firstly reduced and then increased, and the approximate minimum was reached when dropout ≈ 0.85.

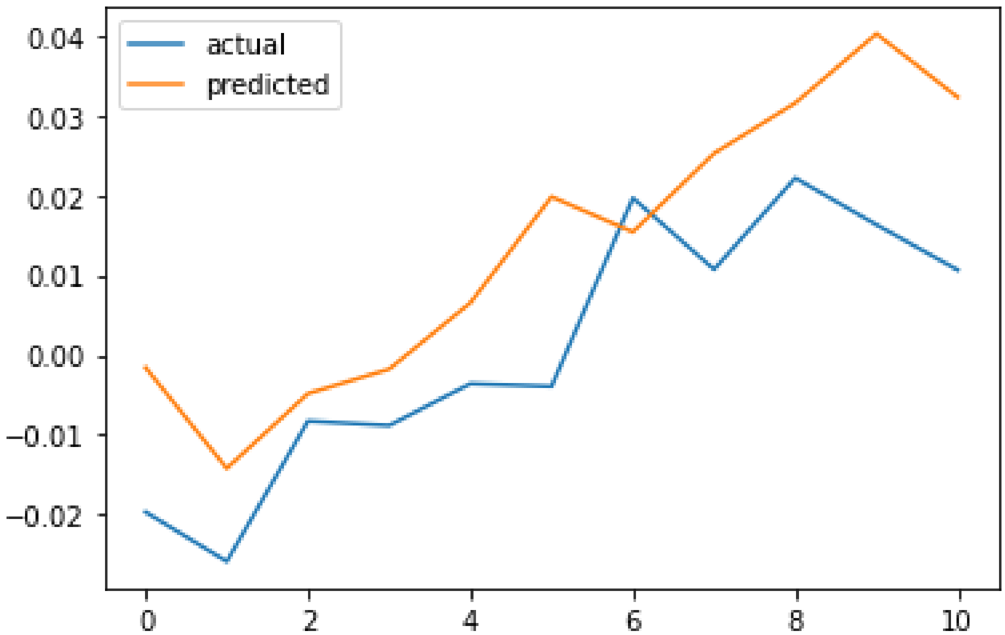
2.5 LSTM Parameter: optimizer

[20]:
```
optimizer_lst = ["sgd", "rmsprop", "adam", "adadelta", "adagrad", "adamax",␣
↪"nadam", "ftrl"]
optimizer_dic = {}

for optimizer in optimizer_lst:
    def this_build_model(inputs, output_size, neurons, activ_func="linear",
                         dropout=0.10, loss="mae", optimizer=optimizer):
        model = Sequential()

        model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
        model.add(Dropout(dropout))
        model. add(Dense(units=output_size))
        model.add(Activation(activ_func))

        model.compile(loss="mae", optimizer=optimizer)
        return model

    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)

    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
↪split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
```

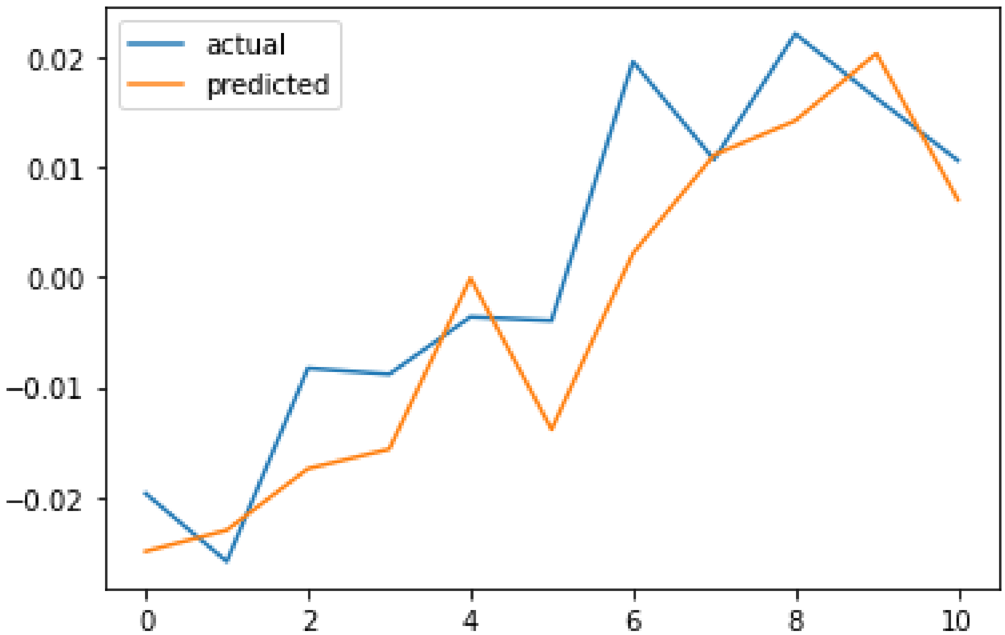
```
    LSTM_training_outputs = (training_set['Close'][window_len:].values/
↪training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for␣
↪LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/
↪test_set['Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
↪LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)

    # initialise model architecture
    nn_model = this_build_model(LSTM_training_inputs, output_size=1, neurons =␣
↪32)
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                            epochs=5, batch_size=1, verbose=2, shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    MAE = mean_absolute_error(LSTM_test_outputs, nn_model.
↪predict(LSTM_test_inputs))
    optimizer_dic[optimizer] = MAE
optimizer_result = pd.DataFrame(optimizer_dic.values(), optimizer_dic.keys()).
↪rename(columns={0: "MAE"})
```

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0371 - 5s/epoch - 1ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0255 - 4s/epoch - 1ms/step
```

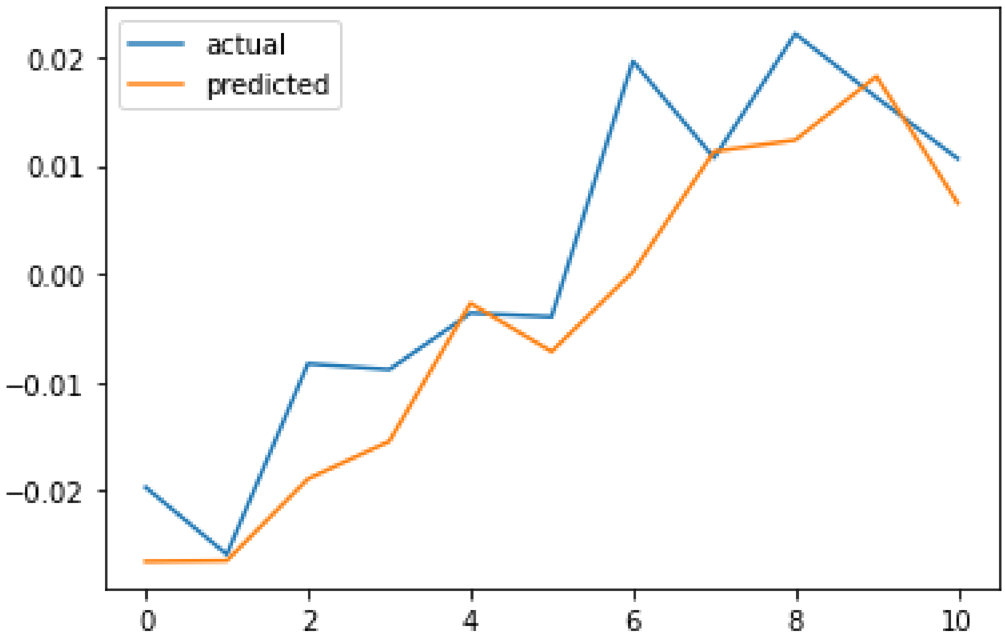
```
Epoch 3/5
3170/3170 - 4s - loss: 0.0234 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0224 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0214 - 4s/epoch - 1ms/step
```



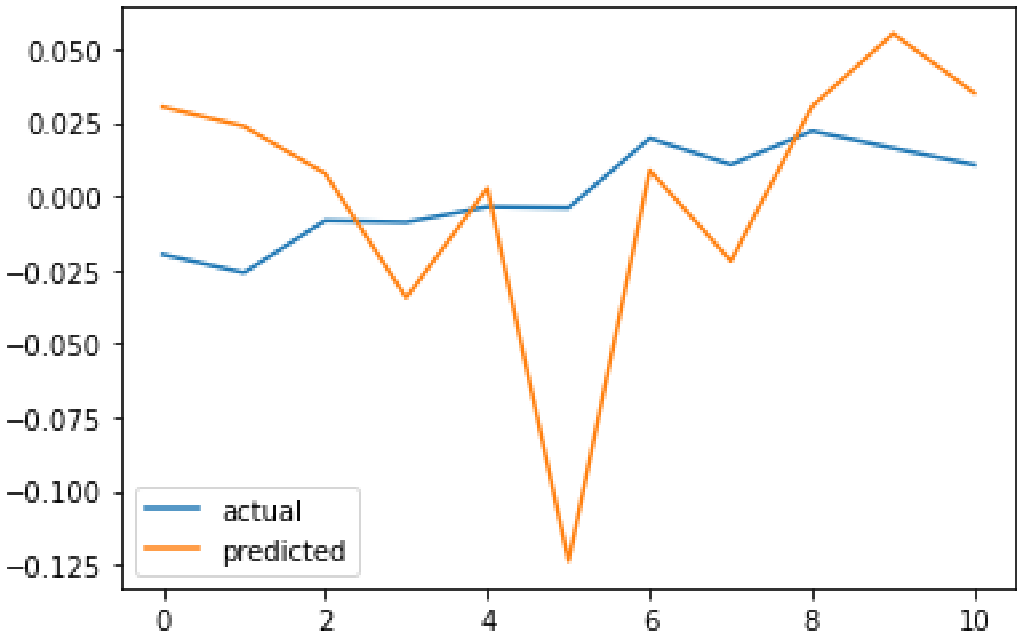

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0233 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0155 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



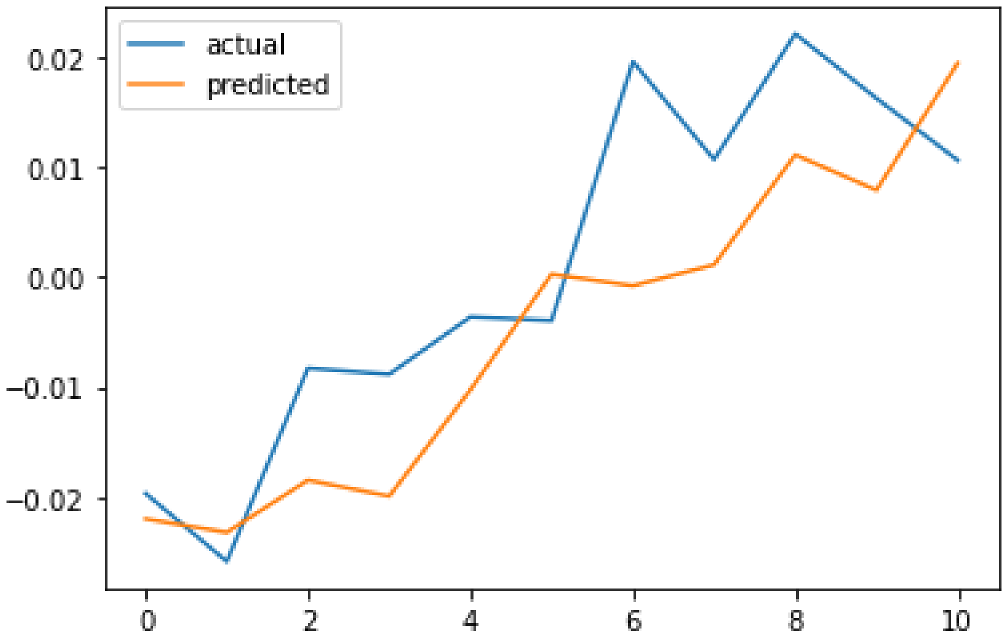

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0229 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0161 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0147 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



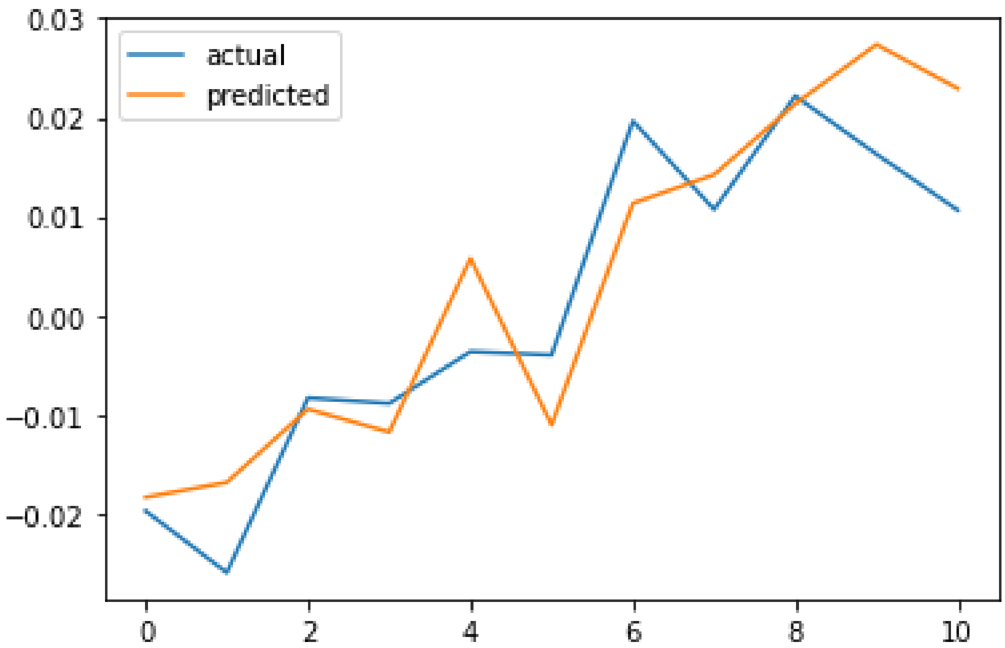

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 6s - loss: 0.1043 - 6s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0845 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0722 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0642 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0599 - 4s/epoch - 1ms/step
```



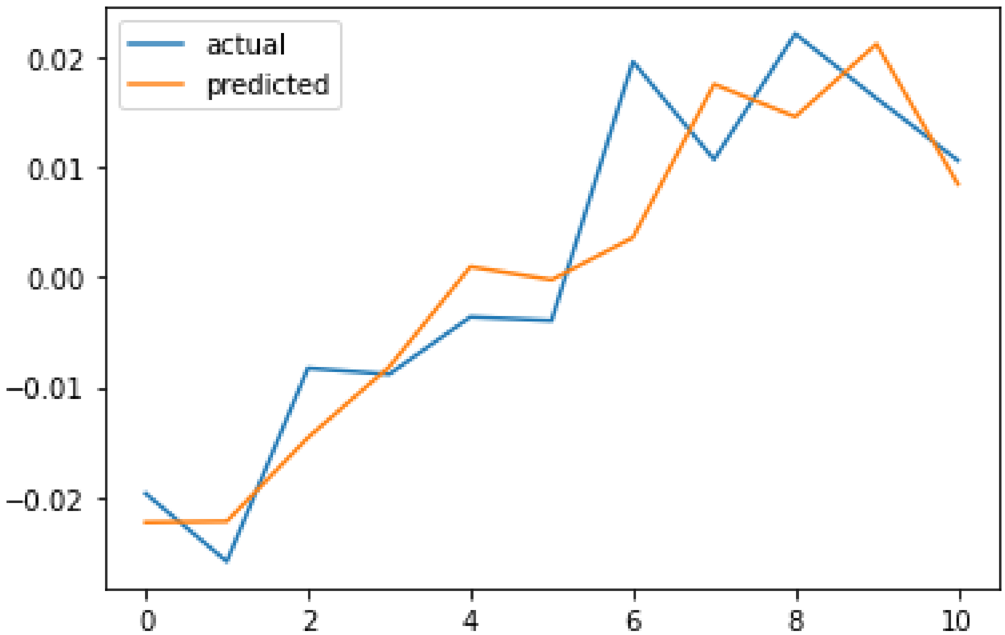

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0420 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0338 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0322 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0319 - 4s/epoch - 1ms/step
```



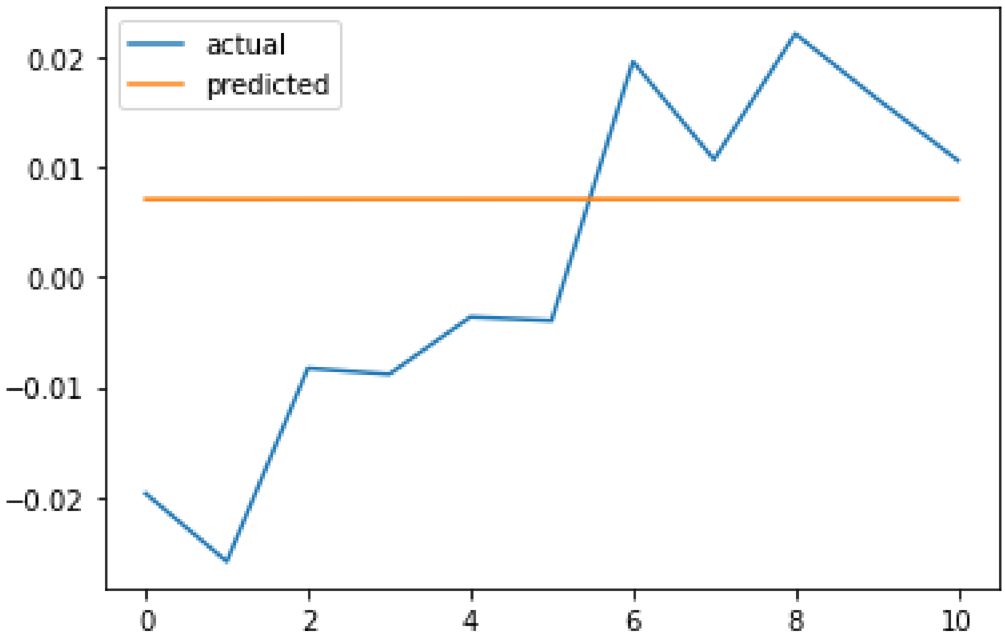

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0302 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0210 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0183 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0174 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0161 - 4s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0227 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0157 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0137 - 4s/epoch - 1ms/step
```



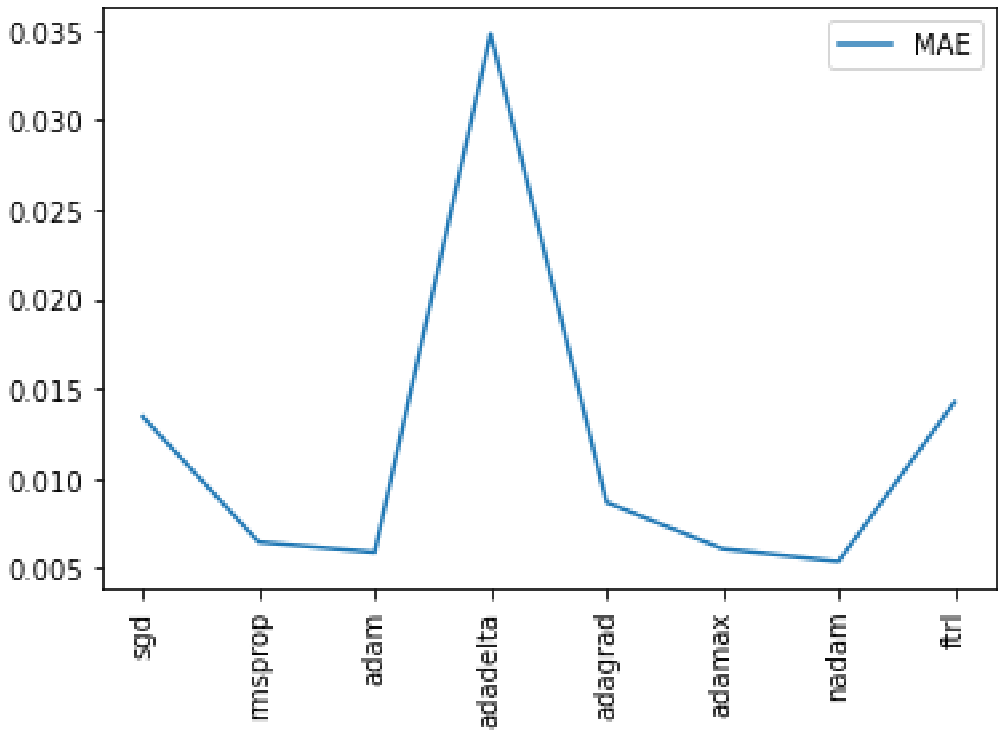

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 1ms/step
```



[21]:
```
print(optimizer_result.loc[optimizer_result["MAE"] == optimizer_result["MAE"].
 ↪min()])
optimizer_result.plot(rot=90)
```

```
              MAE
nadam  0.005371
```

[21]: <AxesSubplot:>



An optimizer is one of the two arguments required for compiling our model. Here, we would choose `nadam` as our optimizer since it generates lowest error.

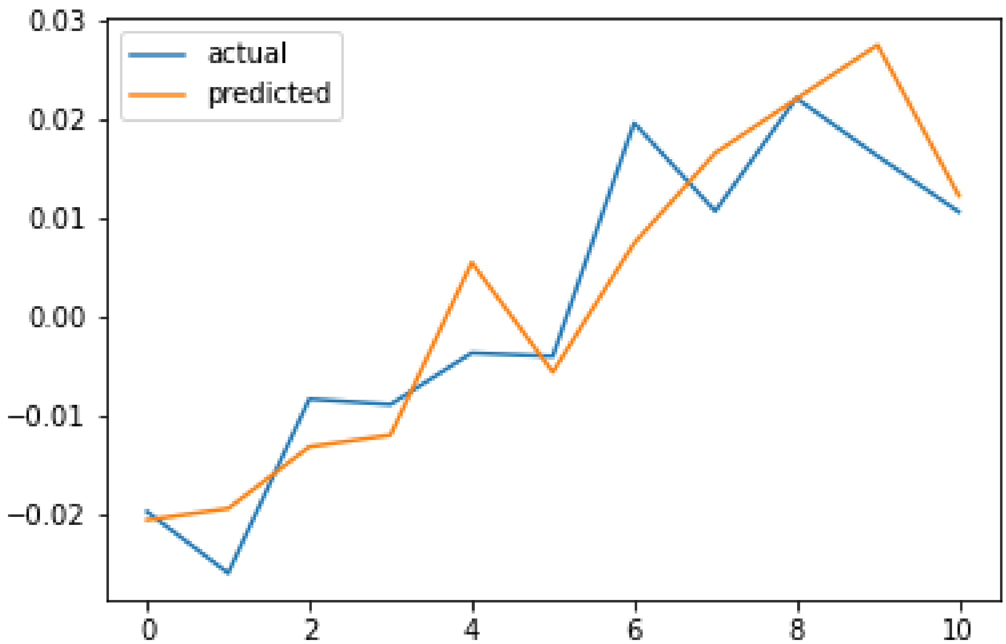
2.6 LSTM Parameter: nn layers/architecture

[22]:

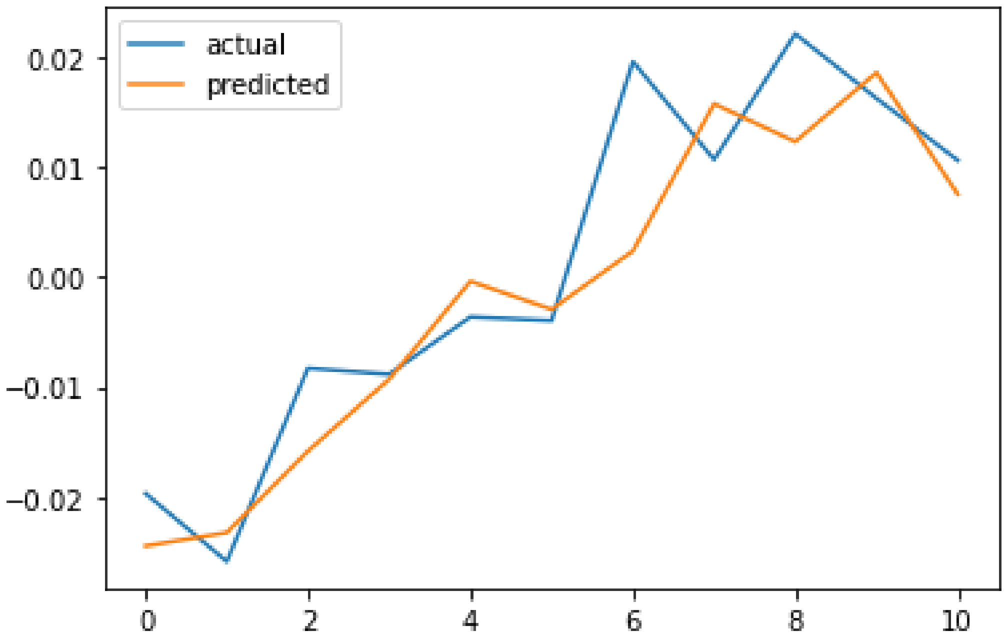
```
neurons_lst = np.arange(32, 168, 8)
neurons_dic = {}

for neurons in neurons_lst:
    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)

    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
 ↪split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()
```

```
        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
    LSTM_training_outputs = (training_set['Close'][window_len:].values/
↪training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for␣
↪LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/
↪test_set['Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
↪LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)

    # initialise model architecture
    nn_model = build_model(LSTM_training_inputs, output_size=1, neurons =␣
↪neurons)
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                            epochs=5, batch_size=1, verbose=2, shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    MAE = mean_absolute_error(LSTM_test_outputs, nn_model.
↪predict(LSTM_test_inputs))
    neurons_dic[neurons] = MAE
neurons_result = pd.DataFrame(neurons_dic.values(), neurons_dic.keys()).
↪rename(columns={0: "MAE"})
```

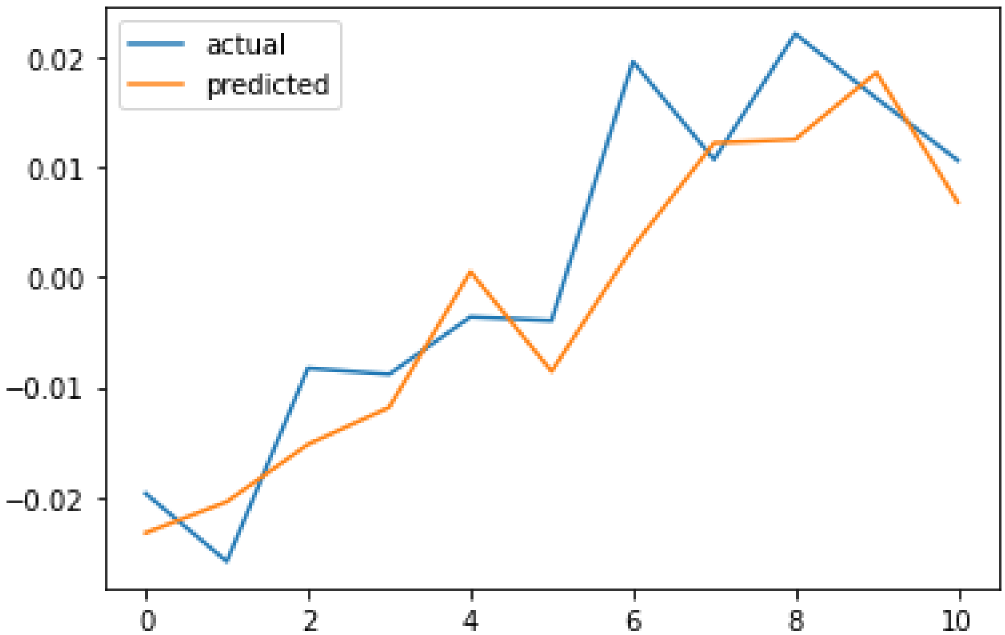
```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0246 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0163 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
```



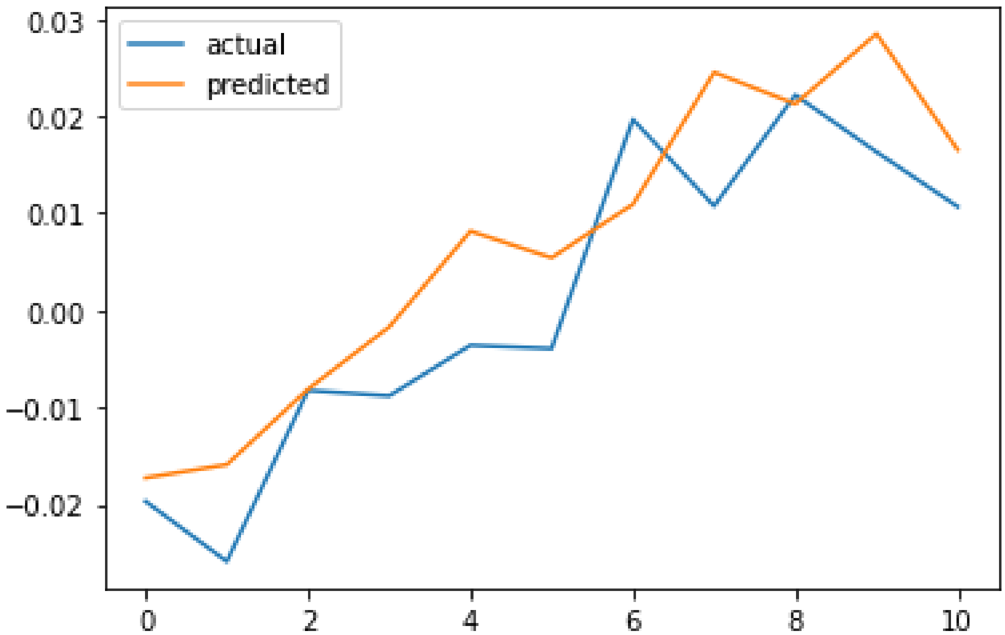

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0233 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0154 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0136 - 4s/epoch - 1ms/step
```



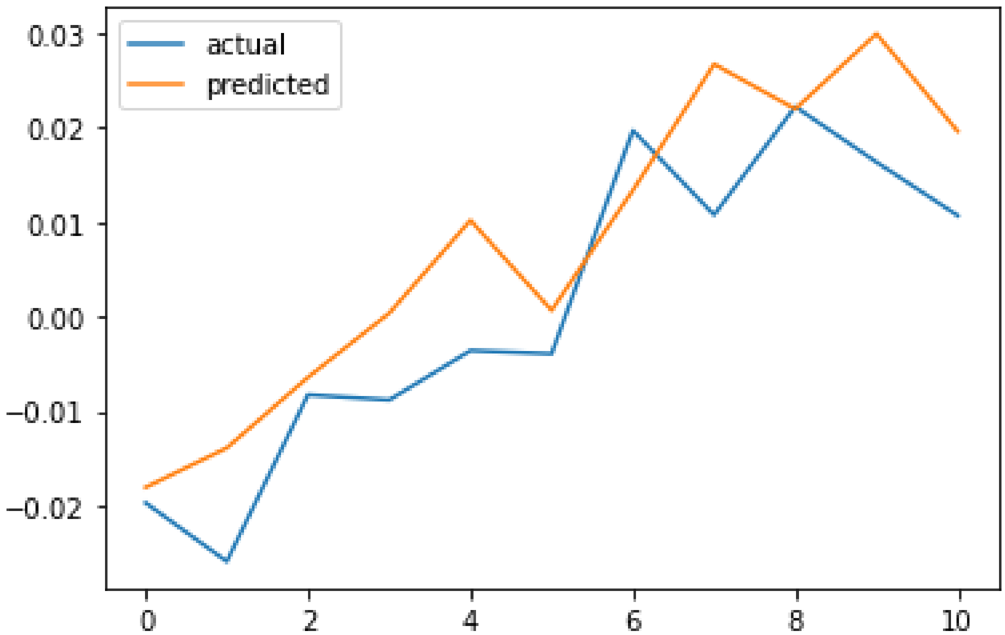

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0248 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0157 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



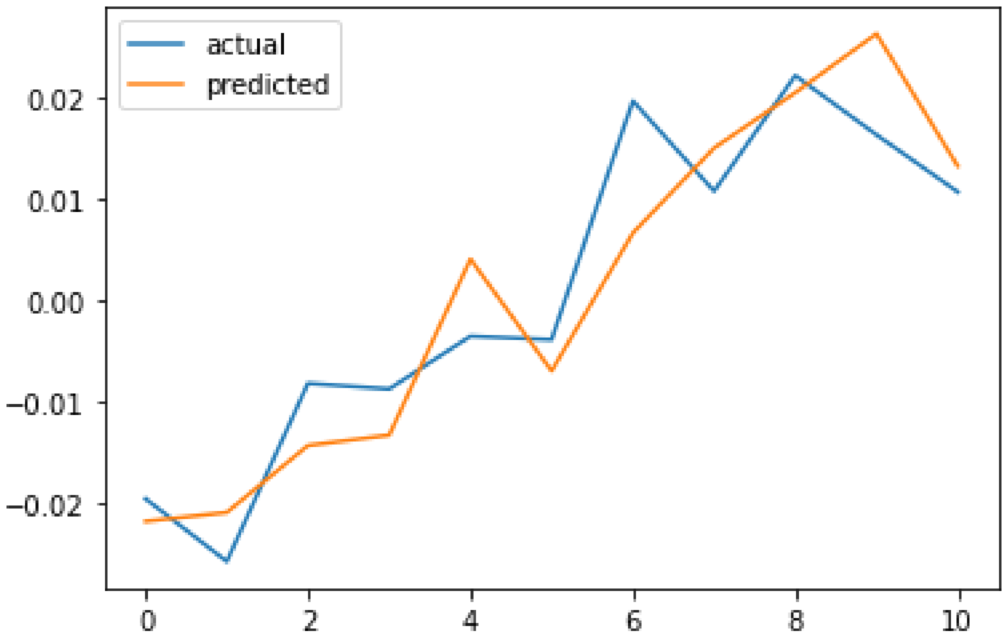

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0236 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0157 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0141 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



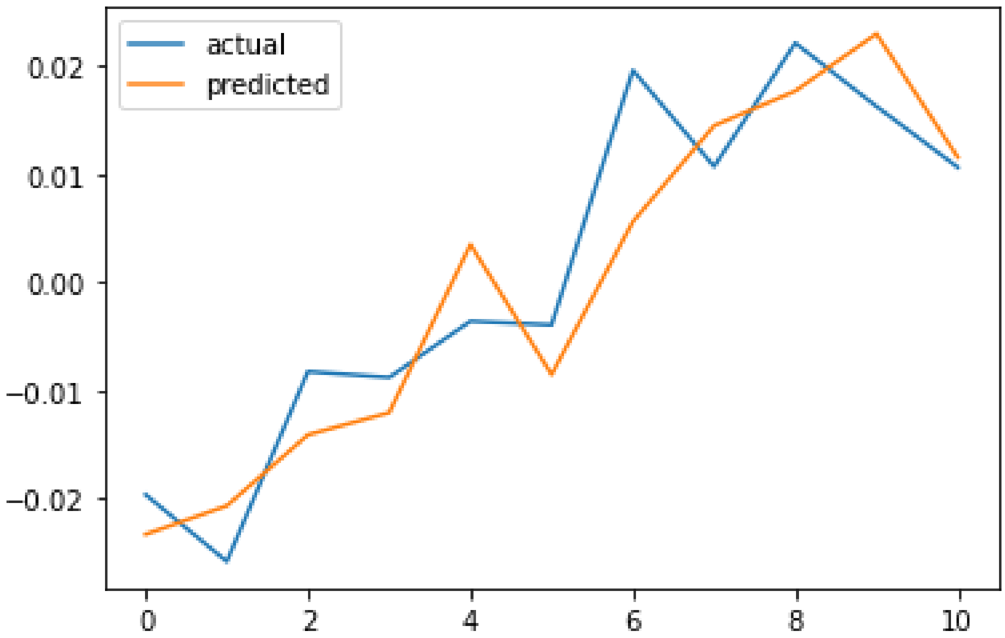

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0225 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0156 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



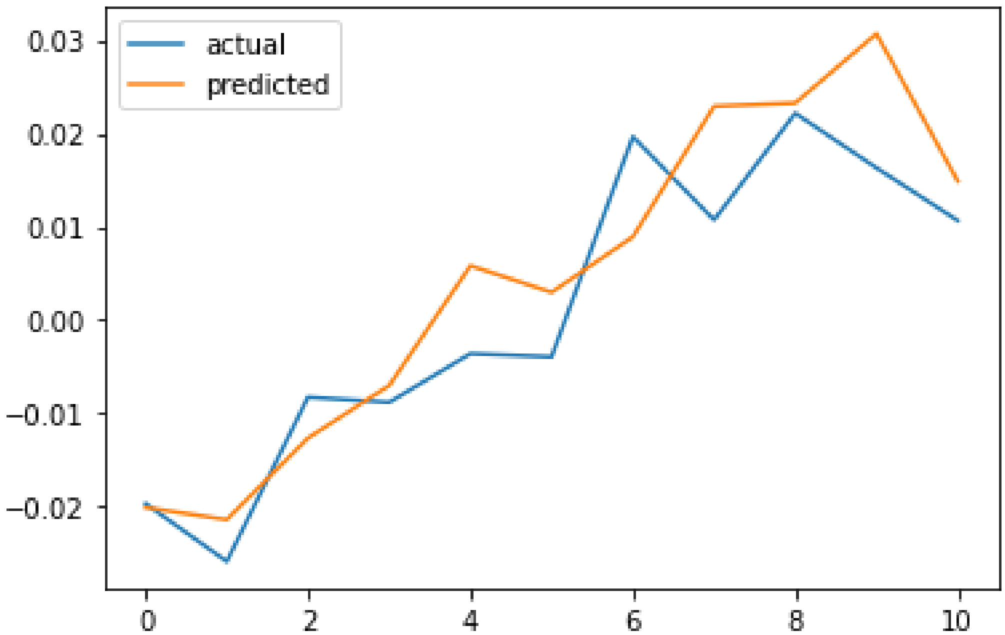

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0231 - 5s/epoch - 1ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0158 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0140 - 5s/epoch - 2ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



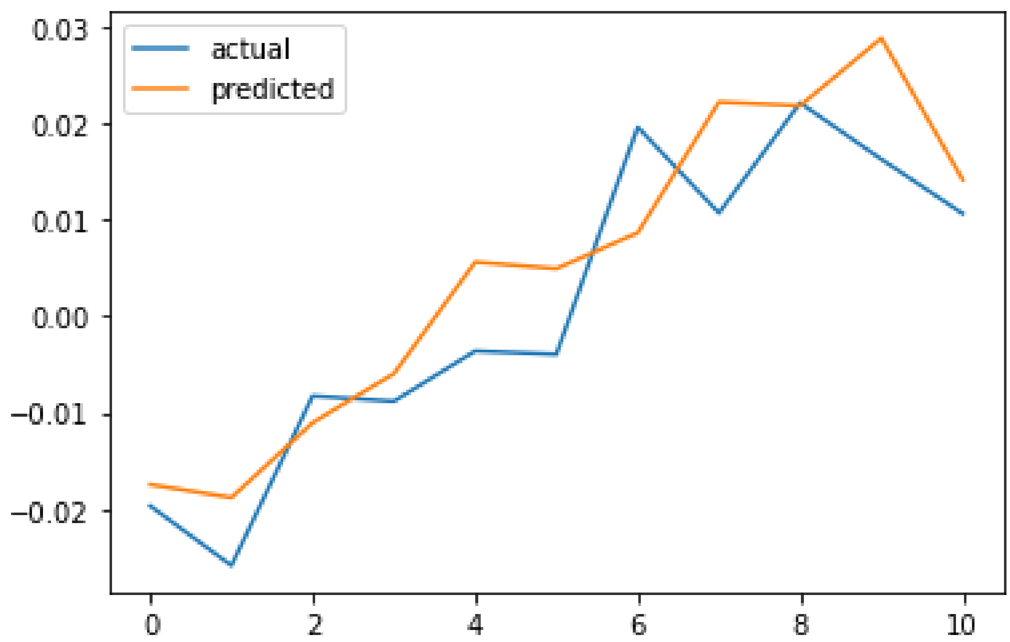

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0229 - 5s/epoch - 1ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0156 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0142 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
```



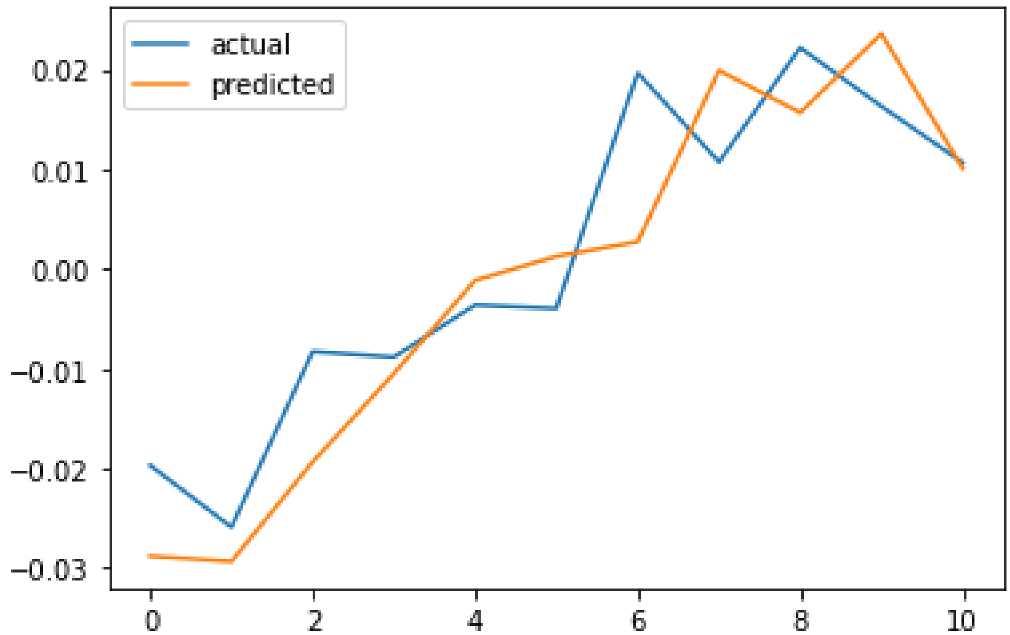

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0210 - 5s/epoch - 1ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0155 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



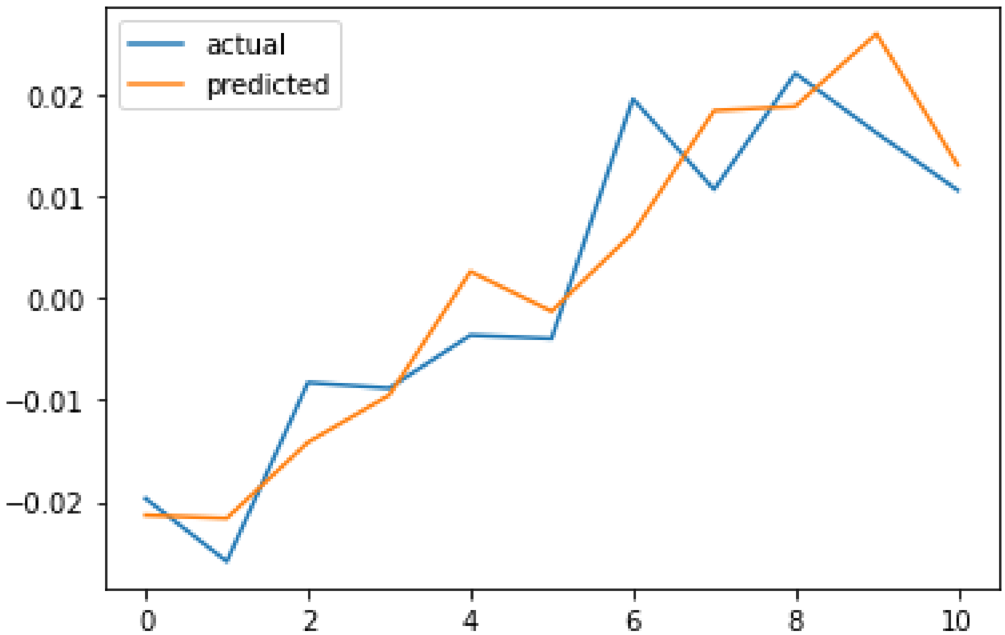

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 6s - loss: 0.0216 - 6s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0156 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0144 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0141 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0137 - 4s/epoch - 1ms/step
```



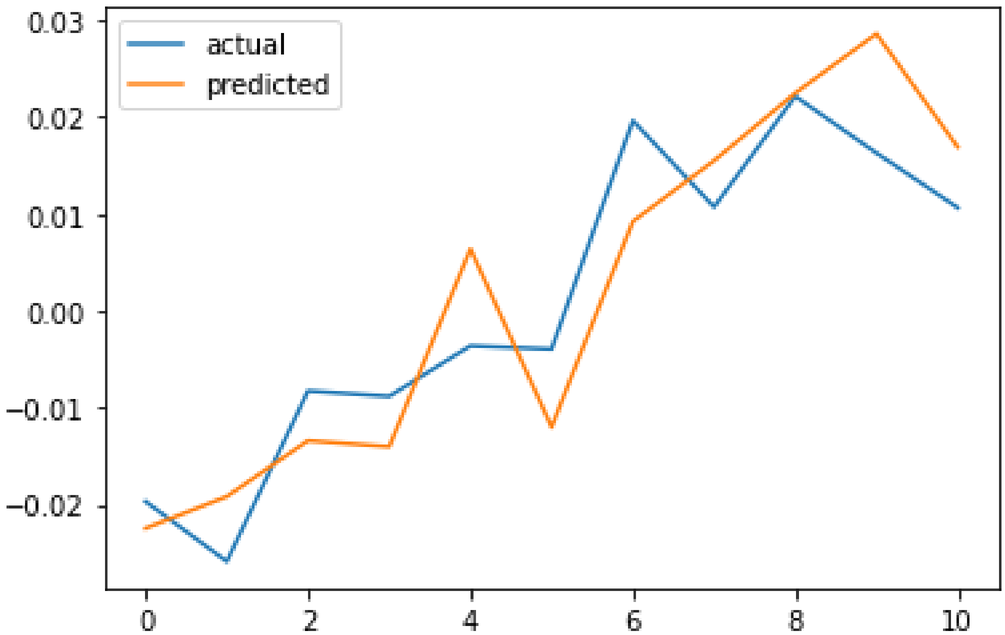

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0218 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0154 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0143 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0138 - 4s/epoch - 1ms/step
```



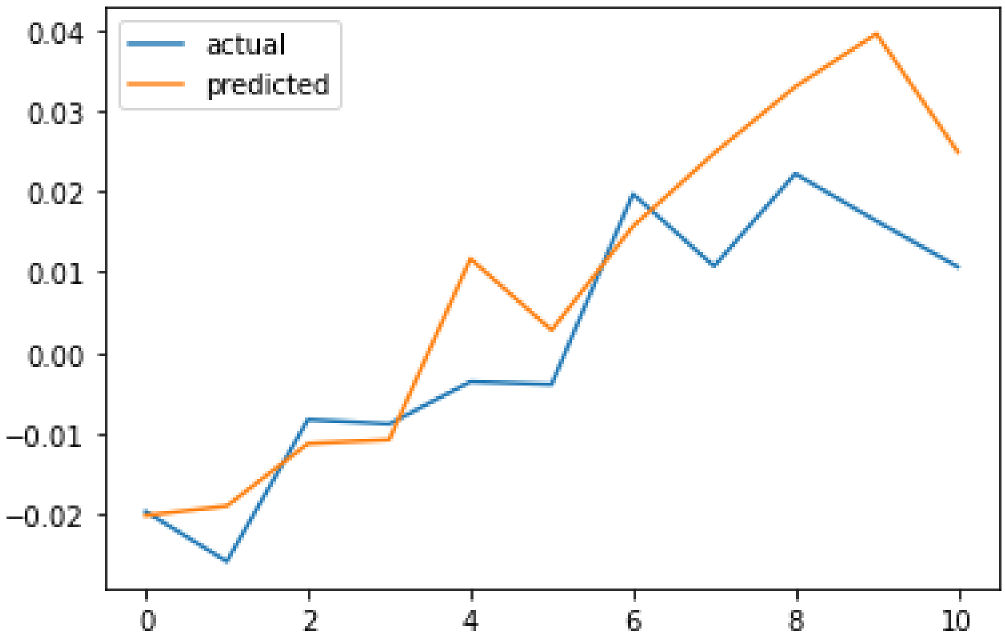

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0226 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0155 - 5s/epoch - 2ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0145 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0140 - 4s/epoch - 1ms/step
```



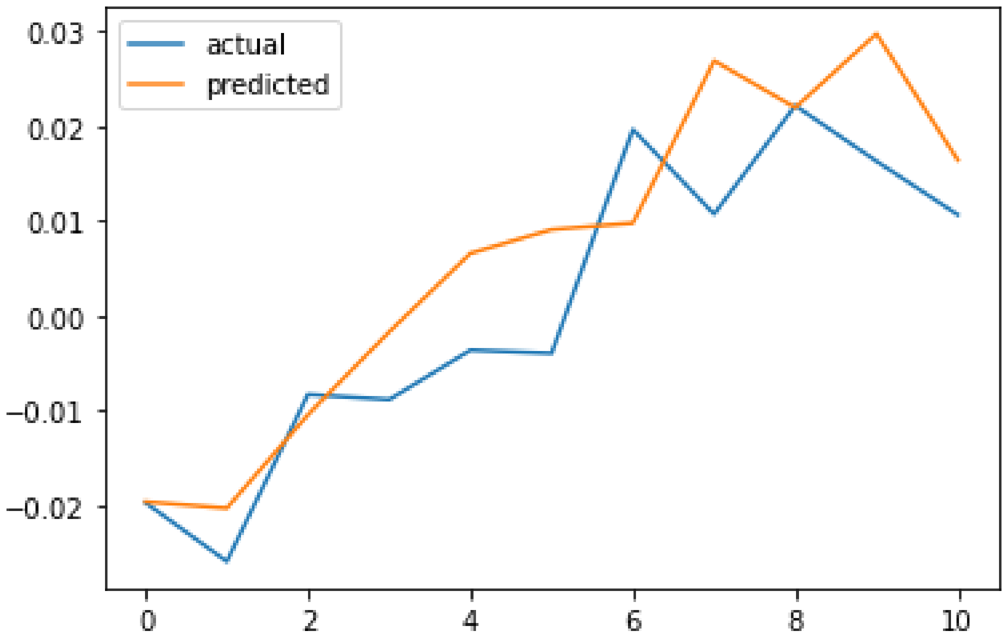

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0207 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0153 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



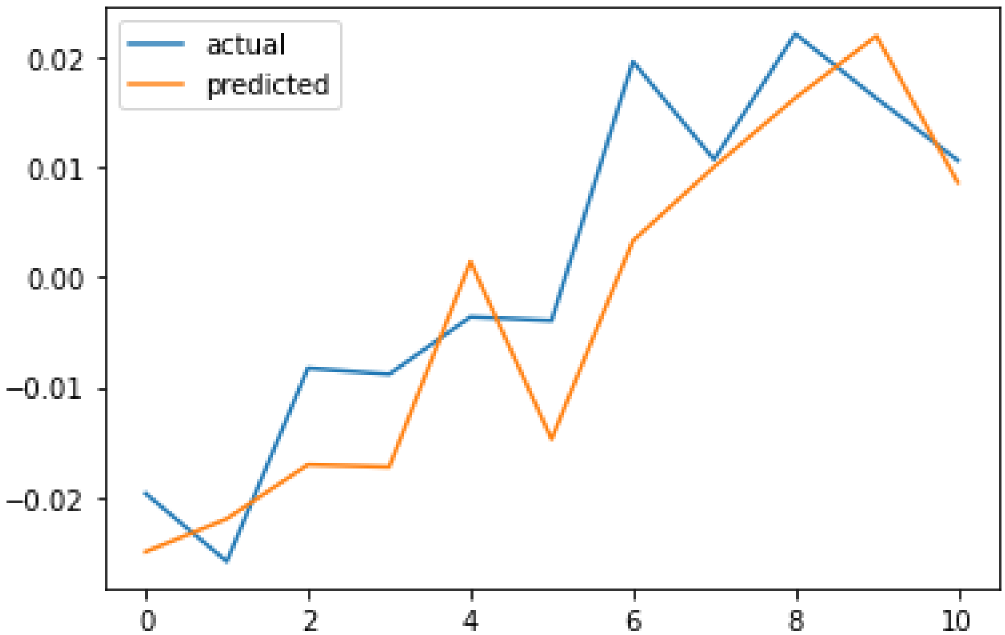

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0212 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0153 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0146 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0142 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0139 - 4s/epoch - 1ms/step
```



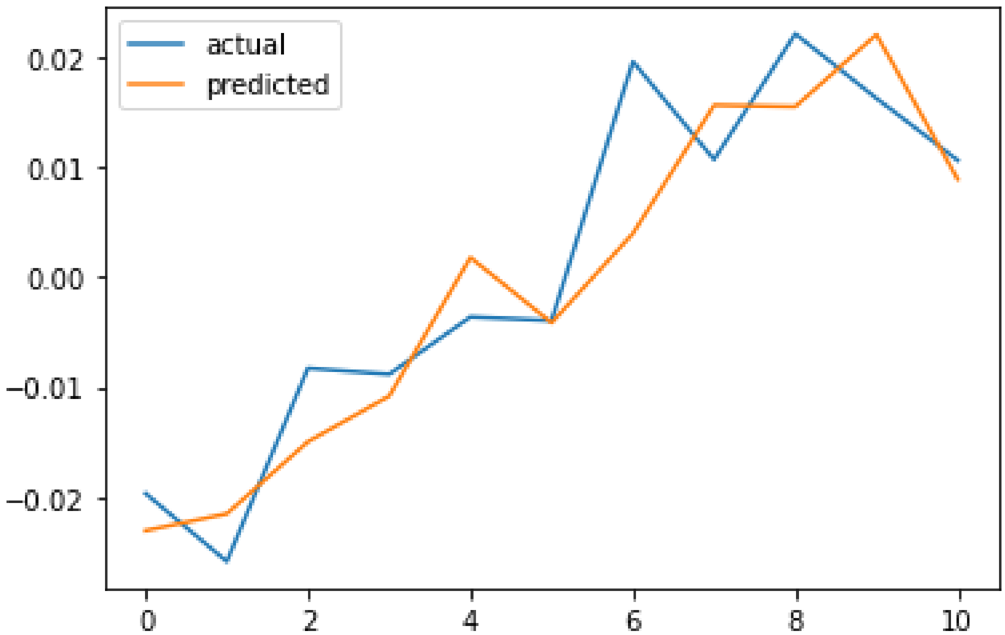

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 6s - loss: 0.0214 - 6s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0153 - 5s/epoch - 2ms/step
Epoch 3/5
3170/3170 - 6s - loss: 0.0146 - 6s/epoch - 2ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0141 - 5s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0141 - 4s/epoch - 1ms/step
```



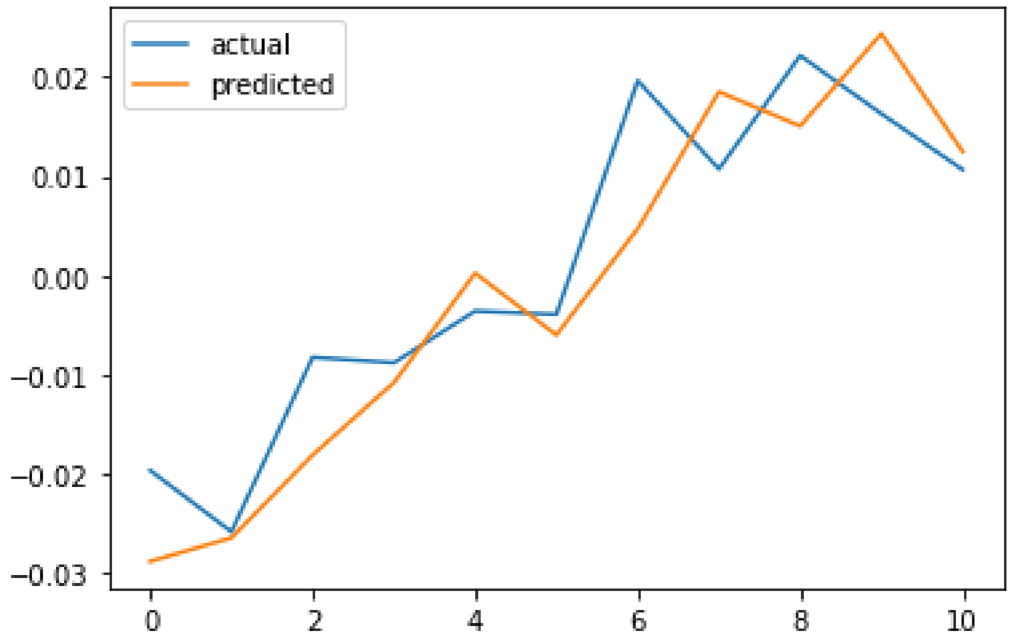

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0209 - 5s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 4s - loss: 0.0156 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0145 - 5s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0142 - 5s/epoch - 2ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0139 - 5s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 6s - loss: 0.0206 - 6s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0152 - 5s/epoch - 2ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0144 - 5s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 5s - loss: 0.0142 - 5s/epoch - 2ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0140 - 5s/epoch - 2ms/step
```



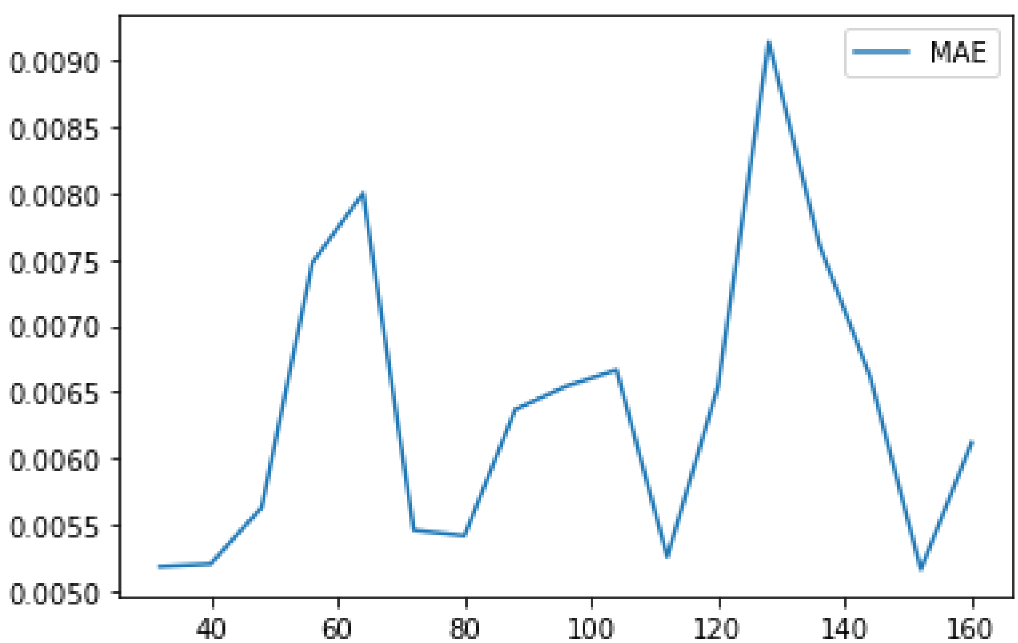

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 6s - loss: 0.0207 - 6s/epoch - 2ms/step
Epoch 2/5
3170/3170 - 5s - loss: 0.0152 - 5s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 5s - loss: 0.0147 - 5s/epoch - 2ms/step
Epoch 4/5
3170/3170 - 6s - loss: 0.0141 - 6s/epoch - 2ms/step
Epoch 5/5
3170/3170 - 5s - loss: 0.0141 - 5s/epoch - 2ms/step
```




```
[23]: print(neurons_result.loc[neurons_result["MAE"] == neurons_result["MAE"].min()])
neurons_result.plot()
```

```
          MAE
152  0.005162
```

[23]: <AxesSubplot:>



The NN architecture is made of neurons, which are simulations of human brain, which is a process that generates some inputs, then converts and transfers them to the node (activation function) and then returns an output NN architecture also contains the bias, transfer function and activation function.

Here, I tested different NN architectures with number of neurons from 32 to 168, and concluded that we would get the lowest error when the number is `152`.

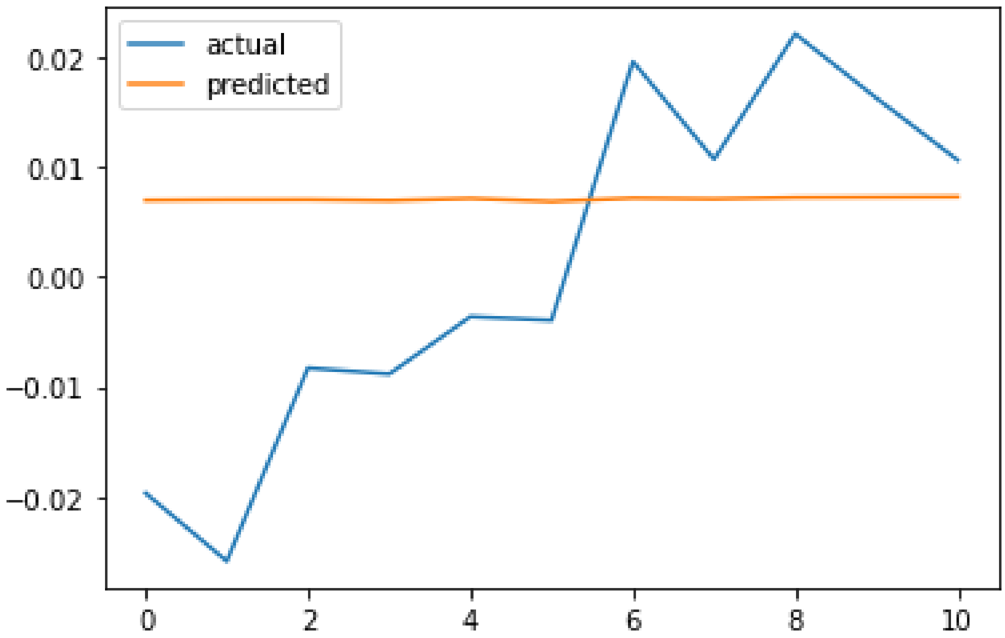
2.7 Epochs

[24]:
```
epochs_lst = range(5, 20)
epochs_dic = {}

for epochs in epochs_lst:
    def this_build_model(inputs, output_size, neurons, activ_func="linear",
                         dropout=0.10, loss="mae", optimizer=optimizer):
        model = Sequential()

        model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
        model.add(Dropout(dropout))
        model.add(Dense(units=output_size))
        model.add(Activation(activ_func))

        model.compile(loss="mae", optimizer=optimizer)
        return model

    split_date = list(df["Date"][-(2*window_len+1):])[0]
    print("split_date:",split_date)

    #Split the training and test set
    training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
 ↪split_date]
    training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
    test_set = test_set.drop(['Date','Label','OpenInt'], 1)

    #Create windows for training
    LSTM_training_inputs = []
    for i in range(len(training_set)-window_len):
        temp_set = training_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_training_inputs.append(temp_set)
    LSTM_training_inputs
```

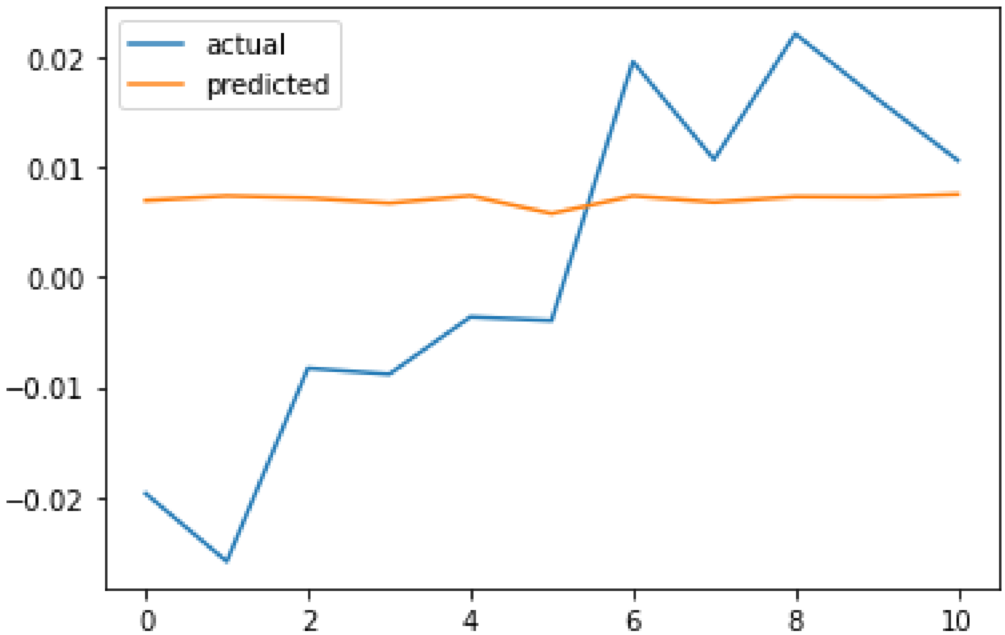
```
    LSTM_training_outputs = (training_set['Close'][window_len:].values/
↪training_set[
        'Close'][:-window_len].values)-1

    LSTM_training_inputs = [np.array(LSTM_training_input) for␣
↪LSTM_training_input in LSTM_training_inputs]
    LSTM_training_inputs = np.array(LSTM_training_inputs)

    # Create windows for testing
    LSTM_test_inputs = []
    for i in range(len(test_set)-window_len):
        temp_set = test_set[i:(i+window_len)].copy()

        for col in list(temp_set):
            temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

        LSTM_test_inputs.append(temp_set)
    LSTM_test_outputs = (test_set['Close'][window_len:].values/test_set[
        'Close'][:-window_len].values)-1

    LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
↪LSTM_test_inputs]
    LSTM_test_inputs = np.array(LSTM_test_inputs)

    # initialise model architecture
    nn_model = this_build_model(LSTM_training_inputs, output_size=1, neurons =␣
↪32)
    # model output is next price normalised to 10th previous closing price␣
↪train model on data
    # note: eth_history contains information on the training error per epoch
    nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                                epochs=epochs, batch_size=1, verbose=2,␣
↪shuffle=True)
    plt.plot(LSTM_test_outputs, label = "actual")
    plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
    plt.legend()
    plt.show()
    MAE = mean_absolute_error(LSTM_test_outputs, nn_model.
↪predict(LSTM_test_inputs))
    epochs_dic[epochs] = MAE
epochs_result = pd.DataFrame(epochs_dic.values(), epochs_dic.keys()).
↪rename(columns={0: "MAE"})
```

```
split_date: 2017-10-13 00:00:00
Epoch 1/5
3170/3170 - 5s - loss: 0.0361 - 5s/epoch - 2ms/step
Epoch 2/5
```

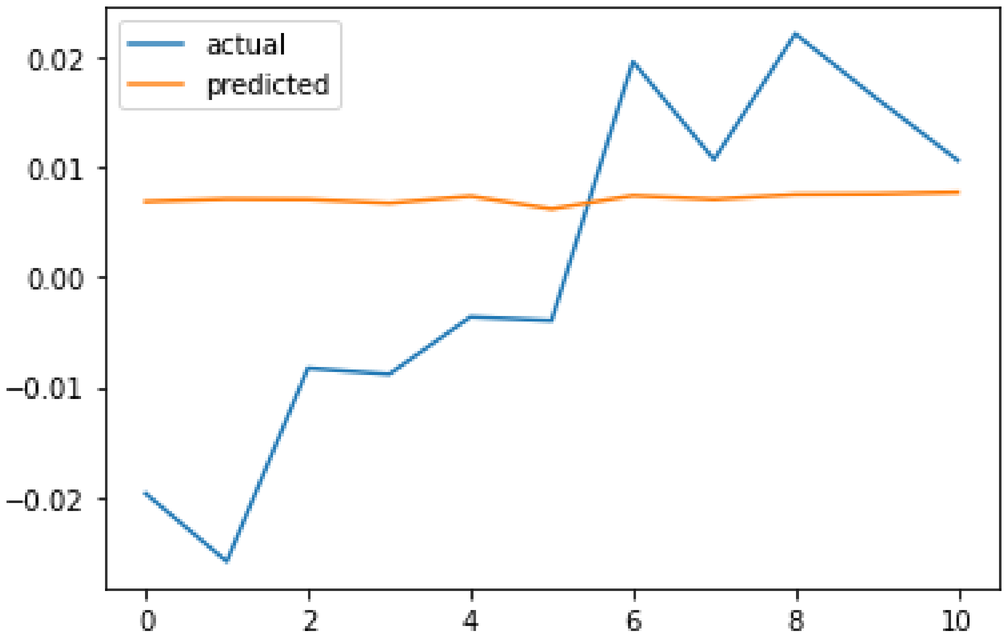
```
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/5
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 4/5
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 5/5
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
```



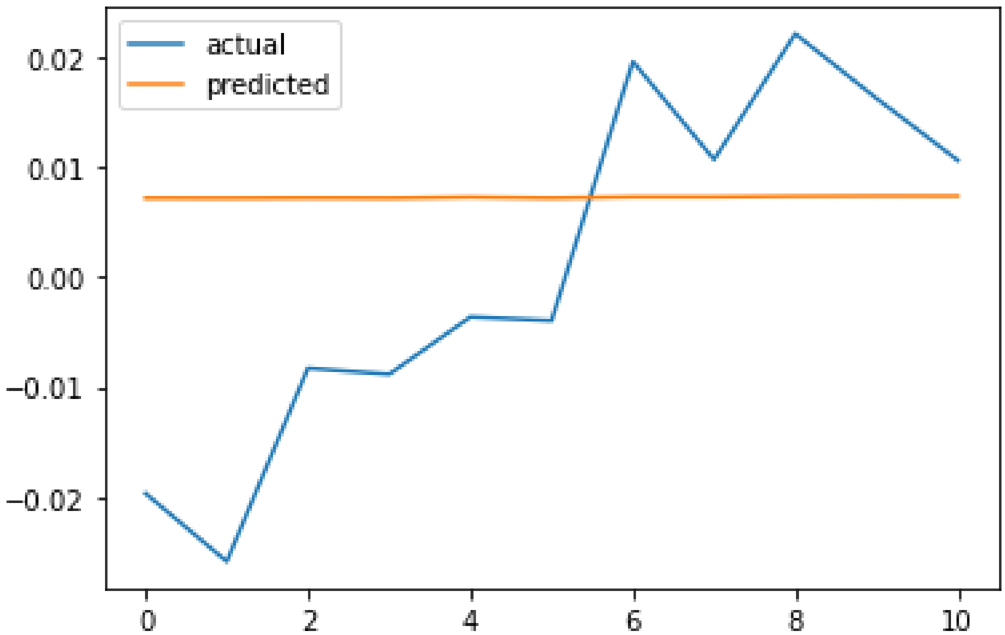

```
split_date: 2017-10-13 00:00:00
Epoch 1/6
3170/3170 - 5s - loss: 0.0361 - 5s/epoch - 2ms/step
Epoch 2/6
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 1ms/step
Epoch 3/6
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/6
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/6
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 6/6
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
```



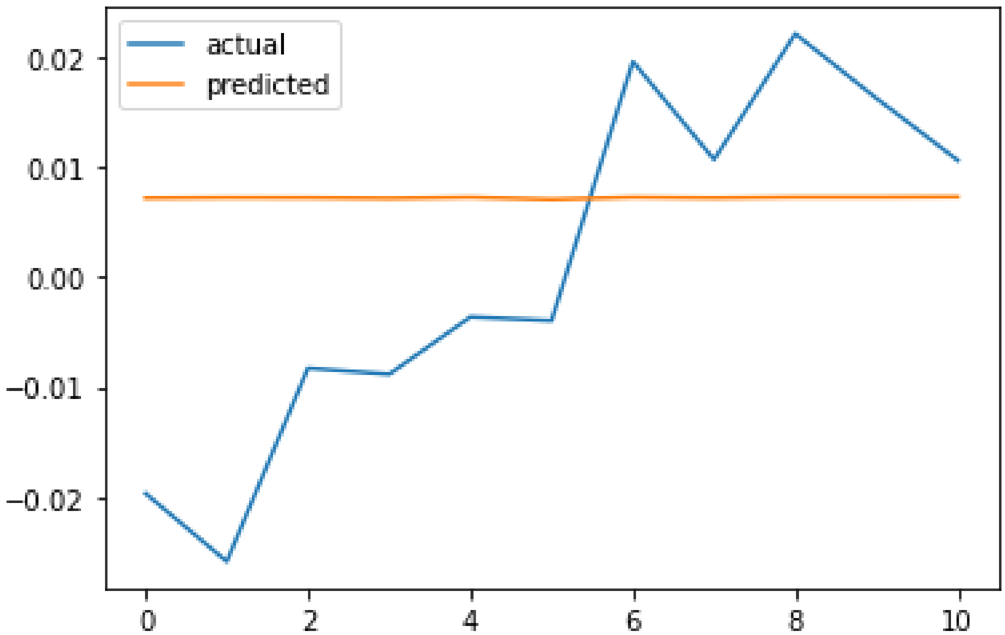

```
split_date: 2017-10-13 00:00:00
Epoch 1/7
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/7
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/7
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/7
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 5/7
3170/3170 - 5s - loss: 0.0358 - 5s/epoch - 1ms/step
Epoch 6/7
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 7/7
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
```



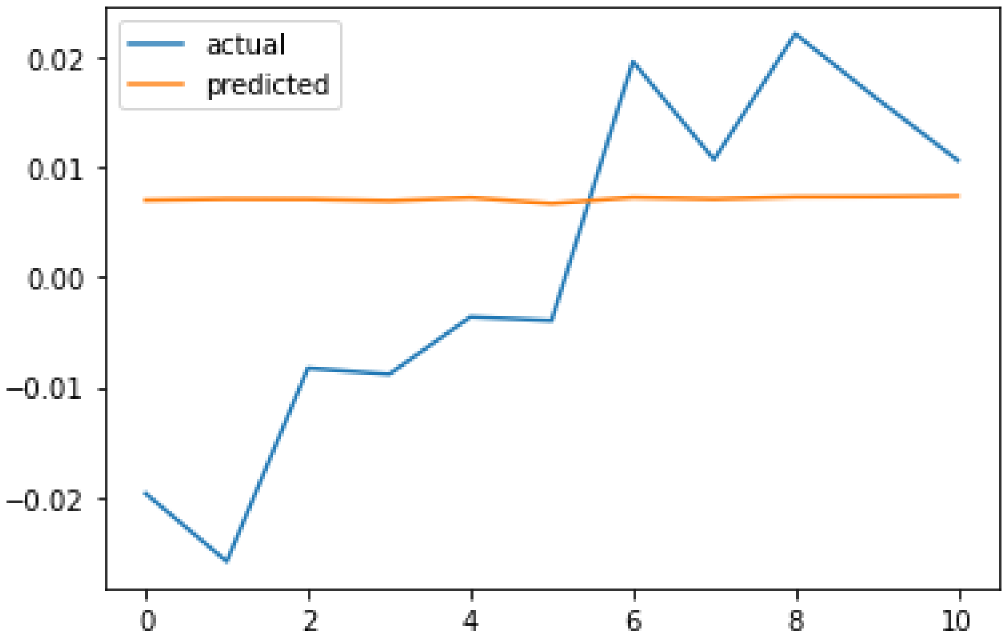

```
split_date: 2017-10-13 00:00:00
Epoch 1/8
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/8
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/8
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/8
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/8
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/8
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 1ms/step
Epoch 7/8
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 8/8
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
```



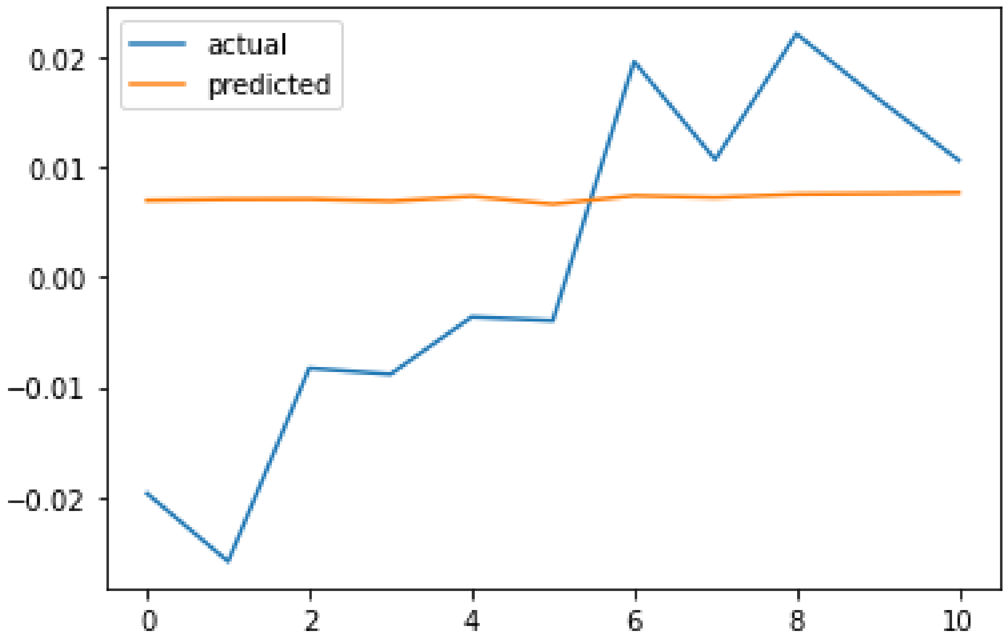

```
split_date: 2017-10-13 00:00:00
Epoch 1/9
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 1ms/step
Epoch 2/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 8/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 9/9
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
```



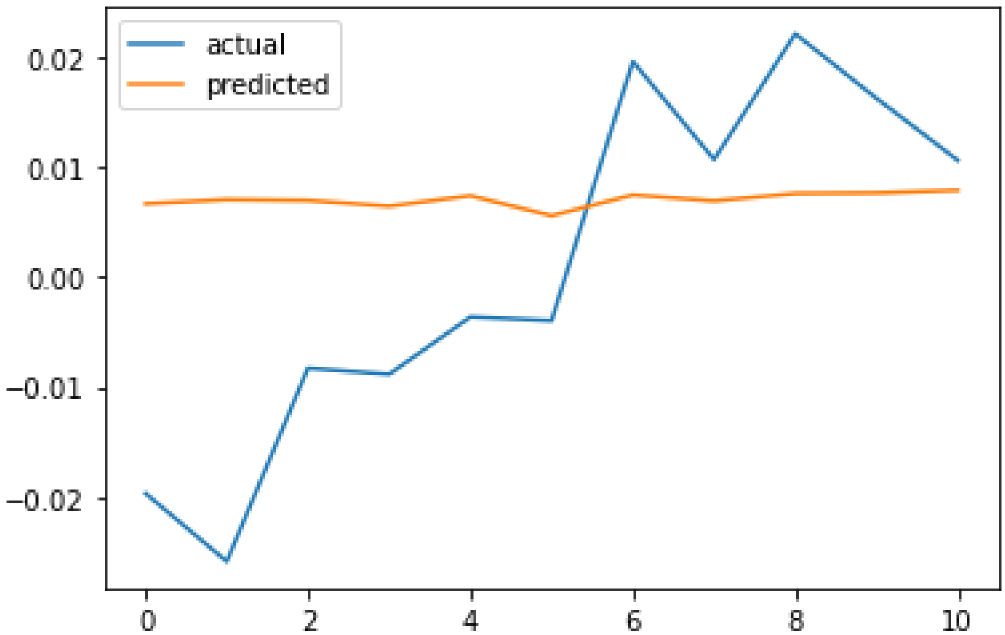

```
split_date: 2017-10-13 00:00:00
Epoch 1/10
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/10
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/10
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/10
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/10
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 1ms/step
Epoch 6/10
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/10
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 1ms/step
Epoch 8/10
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 9/10
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 10/10
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/11
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/11
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/11
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/11
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/11
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/11
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/11
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 8/11
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 9/11
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 10/11
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 11/11
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
```

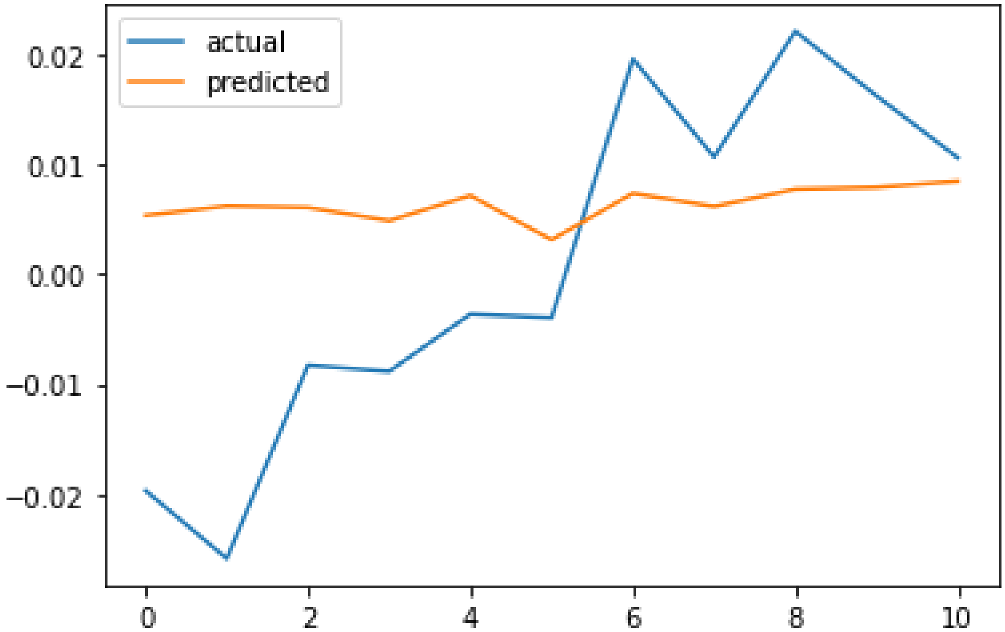



```
split_date: 2017-10-13 00:00:00
Epoch 1/12
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/12
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/12
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/12
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/12
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/12
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 7/12
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 8/12
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 9/12
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 10/12
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 11/12
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 12/12
3170/3170 - 4s - loss: 0.0354 - 4s/epoch - 1ms/step
```



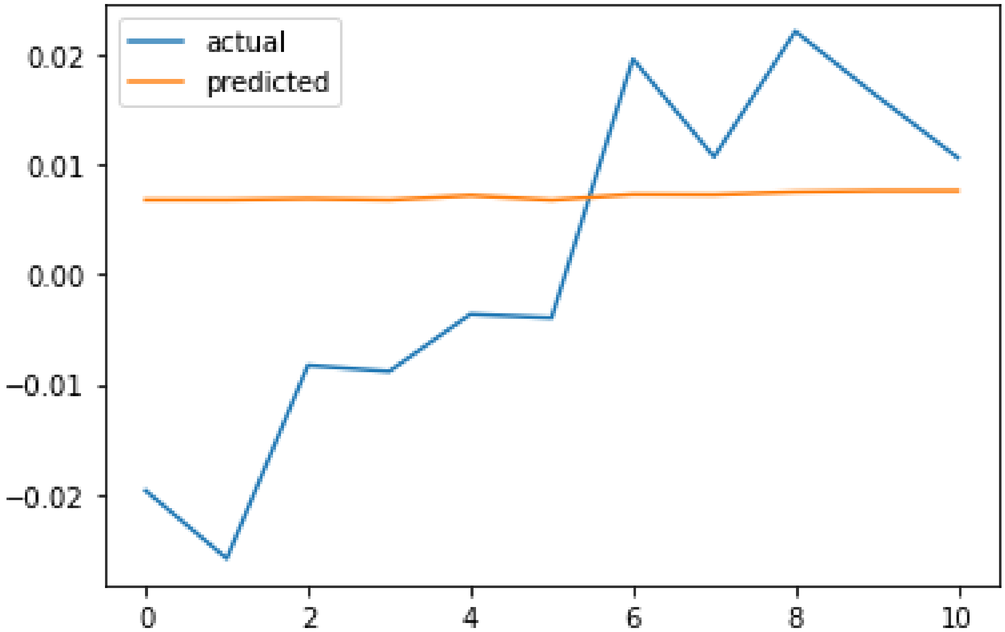

```
split_date: 2017-10-13 00:00:00
Epoch 1/13
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/13
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/13
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 4/13
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 5/13
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 6/13
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 7/13
3170/3170 - 4s - loss: 0.0355 - 4s/epoch - 1ms/step
Epoch 8/13
3170/3170 - 4s - loss: 0.0354 - 4s/epoch - 1ms/step
Epoch 9/13
3170/3170 - 4s - loss: 0.0353 - 4s/epoch - 1ms/step
Epoch 10/13
3170/3170 - 4s - loss: 0.0352 - 4s/epoch - 1ms/step
Epoch 11/13
3170/3170 - 4s - loss: 0.0350 - 4s/epoch - 1ms/step
Epoch 12/13
3170/3170 - 4s - loss: 0.0348 - 4s/epoch - 1ms/step
Epoch 13/13
```

```
3170/3170 - 4s - loss: 0.0345 - 4s/epoch - 1ms/step
```



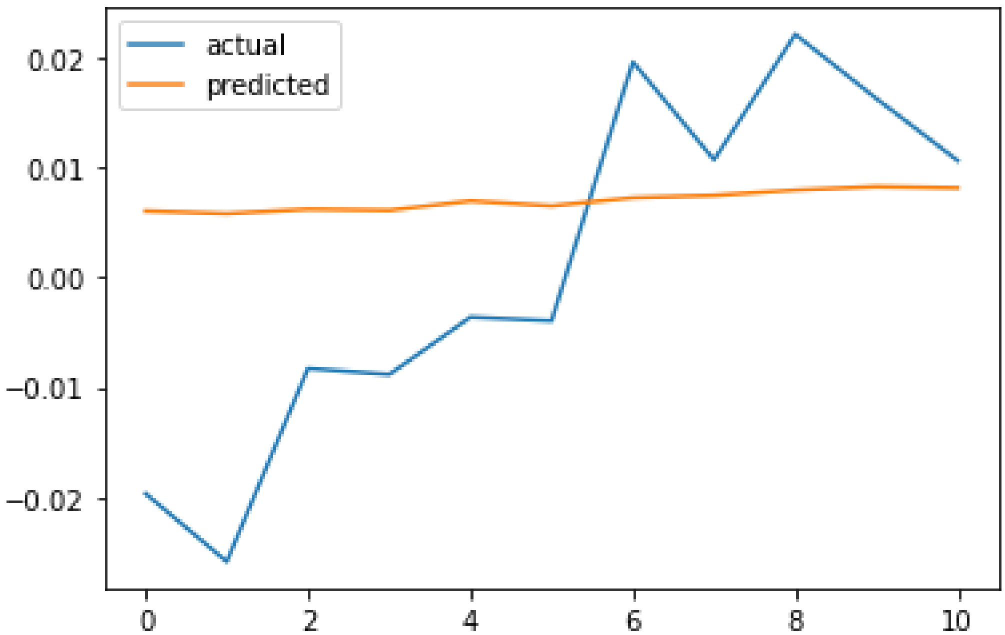

```
split_date: 2017-10-13 00:00:00
Epoch 1/14
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 1ms/step
Epoch 2/14
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/14
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/14
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/14
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/14
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/14
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 8/14
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 9/14
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 10/14
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 11/14
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 12/14
```

```
3170/3170 - 5s - loss: 0.0357 - 5s/epoch - 1ms/step
Epoch 13/14
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 14/14
3170/3170 - 4s - loss: 0.0355 - 4s/epoch - 1ms/step
```



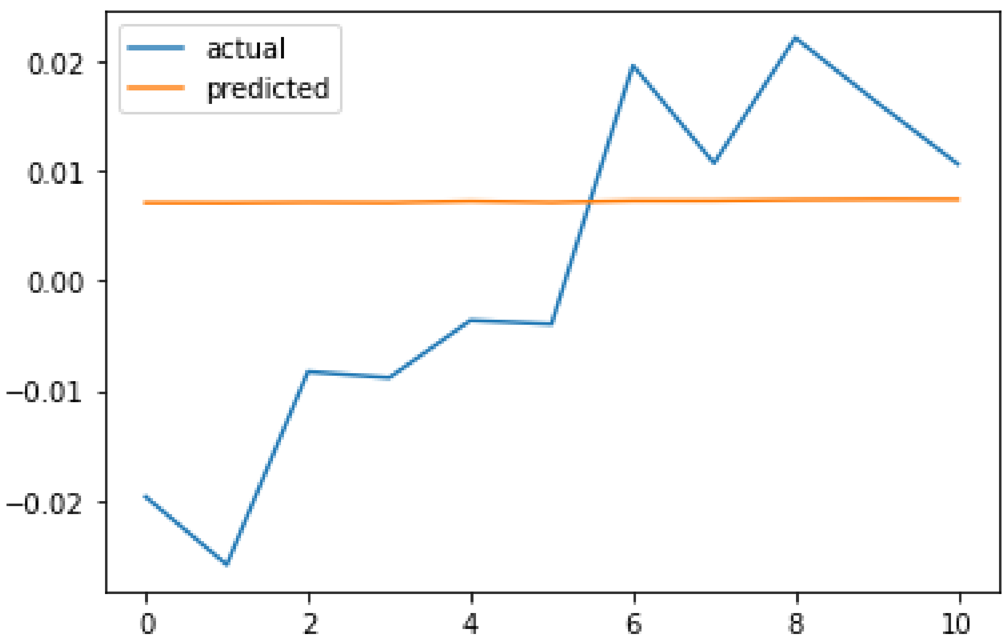

```
split_date: 2017-10-13 00:00:00
Epoch 1/15
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/15
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/15
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/15
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/15
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/15
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/15
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 8/15
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 1ms/step
Epoch 9/15
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 10/15
```

```
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 11/15
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 12/15
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 13/15
3170/3170 - 4s - loss: 0.0354 - 4s/epoch - 1ms/step
Epoch 14/15
3170/3170 - 4s - loss: 0.0351 - 4s/epoch - 1ms/step
Epoch 15/15
3170/3170 - 4s - loss: 0.0347 - 4s/epoch - 1ms/step
```



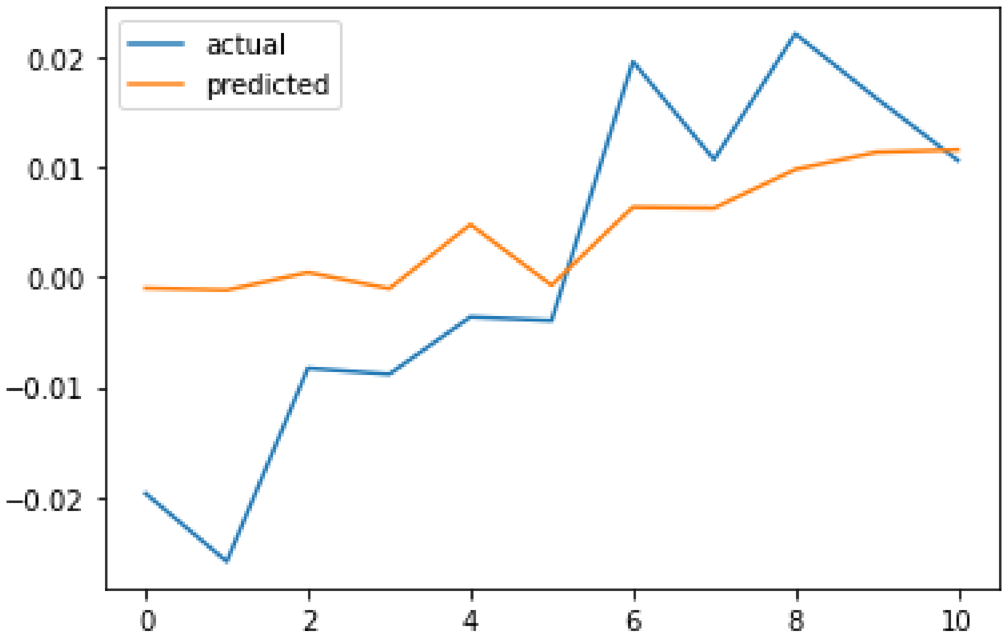

```
split_date: 2017-10-13 00:00:00
Epoch 1/16
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 1ms/step
Epoch 2/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/16
```

```
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 8/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 9/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 10/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 11/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 12/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 13/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 14/16
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 15/16
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 16/16
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
```



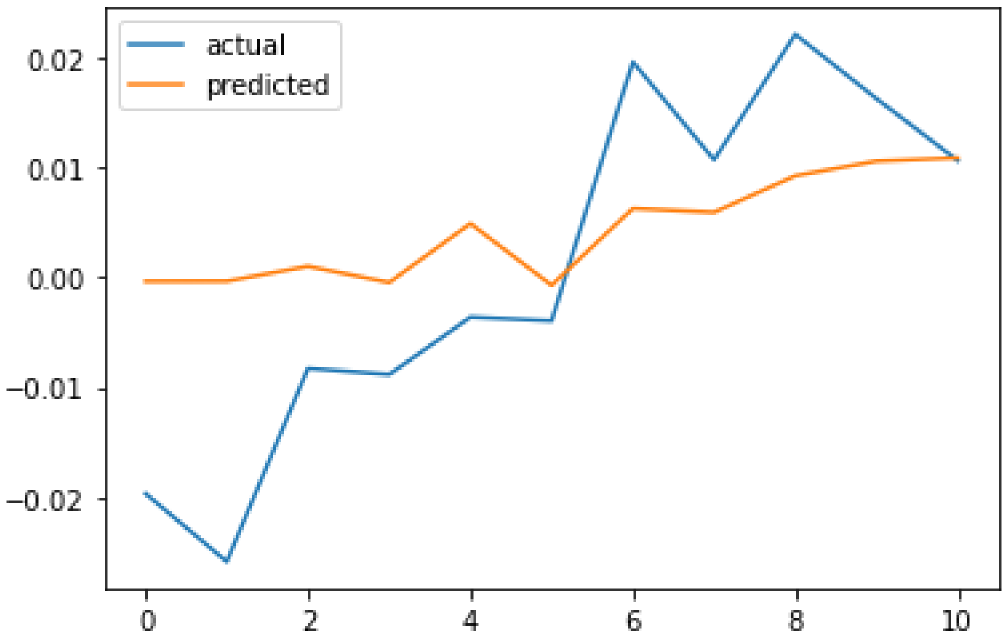

```
split_date: 2017-10-13 00:00:00
Epoch 1/17
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/17
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 3/17
```

```
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 4/17
3170/3170 - 4s - loss: 0.0355 - 4s/epoch - 1ms/step
Epoch 5/17
3170/3170 - 4s - loss: 0.0353 - 4s/epoch - 1ms/step
Epoch 6/17
3170/3170 - 4s - loss: 0.0351 - 4s/epoch - 1ms/step
Epoch 7/17
3170/3170 - 4s - loss: 0.0349 - 4s/epoch - 1ms/step
Epoch 8/17
3170/3170 - 4s - loss: 0.0347 - 4s/epoch - 1ms/step
Epoch 9/17
3170/3170 - 4s - loss: 0.0343 - 4s/epoch - 1ms/step
Epoch 10/17
3170/3170 - 4s - loss: 0.0341 - 4s/epoch - 1ms/step
Epoch 11/17
3170/3170 - 4s - loss: 0.0336 - 4s/epoch - 1ms/step
Epoch 12/17
3170/3170 - 4s - loss: 0.0331 - 4s/epoch - 1ms/step
Epoch 13/17
3170/3170 - 4s - loss: 0.0325 - 4s/epoch - 1ms/step
Epoch 14/17
3170/3170 - 4s - loss: 0.0317 - 4s/epoch - 1ms/step
Epoch 15/17
3170/3170 - 4s - loss: 0.0309 - 4s/epoch - 1ms/step
Epoch 16/17
3170/3170 - 4s - loss: 0.0299 - 4s/epoch - 1ms/step
Epoch 17/17
3170/3170 - 4s - loss: 0.0288 - 4s/epoch - 1ms/step
```



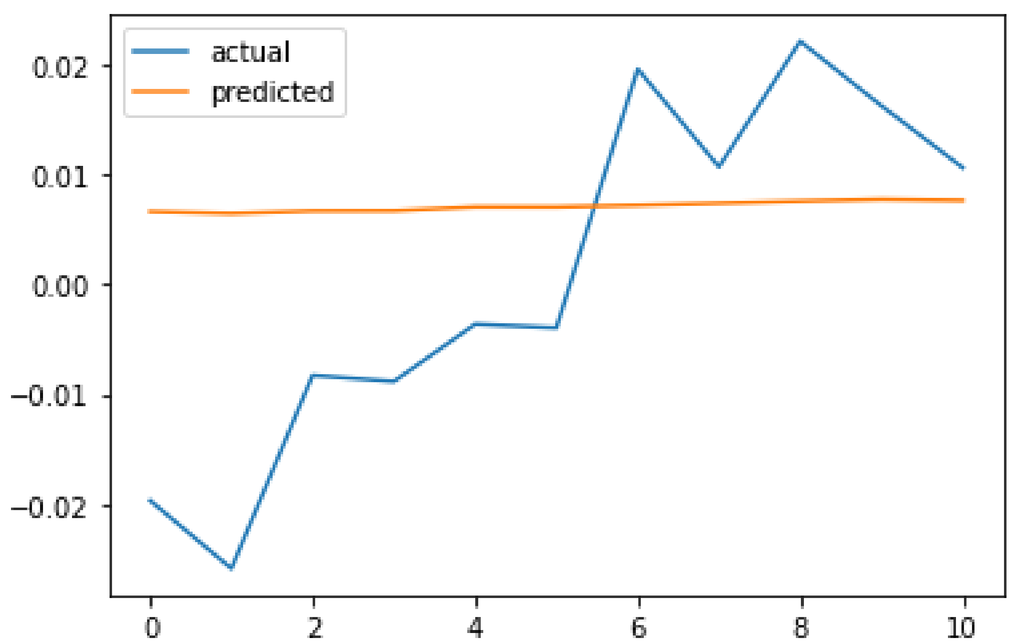

```
split_date: 2017-10-13 00:00:00
Epoch 1/18
3170/3170 - 5s - loss: 0.0359 - 5s/epoch - 2ms/step
Epoch 2/18
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 3/18
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 4/18
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 5/18
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 6/18
3170/3170 - 4s - loss: 0.0355 - 4s/epoch - 1ms/step
Epoch 7/18
3170/3170 - 4s - loss: 0.0354 - 4s/epoch - 1ms/step
Epoch 8/18
3170/3170 - 4s - loss: 0.0352 - 4s/epoch - 1ms/step
Epoch 9/18
3170/3170 - 4s - loss: 0.0351 - 4s/epoch - 1ms/step
Epoch 10/18
3170/3170 - 4s - loss: 0.0348 - 4s/epoch - 1ms/step
Epoch 11/18
3170/3170 - 4s - loss: 0.0346 - 4s/epoch - 1ms/step
Epoch 12/18
3170/3170 - 4s - loss: 0.0342 - 4s/epoch - 1ms/step
Epoch 13/18
```

```
3170/3170 - 4s - loss: 0.0338 - 4s/epoch - 1ms/step
Epoch 14/18
3170/3170 - 4s - loss: 0.0333 - 4s/epoch - 1ms/step
Epoch 15/18
3170/3170 - 4s - loss: 0.0327 - 4s/epoch - 1ms/step
Epoch 16/18
3170/3170 - 4s - loss: 0.0318 - 4s/epoch - 1ms/step
Epoch 17/18
3170/3170 - 4s - loss: 0.0309 - 4s/epoch - 1ms/step
Epoch 18/18
3170/3170 - 4s - loss: 0.0297 - 4s/epoch - 1ms/step
```




```
split_date: 2017-10-13 00:00:00
Epoch 1/19
3170/3170 - 5s - loss: 0.0360 - 5s/epoch - 2ms/step
Epoch 2/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 3/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 4/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 5/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 6/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 7/19
```

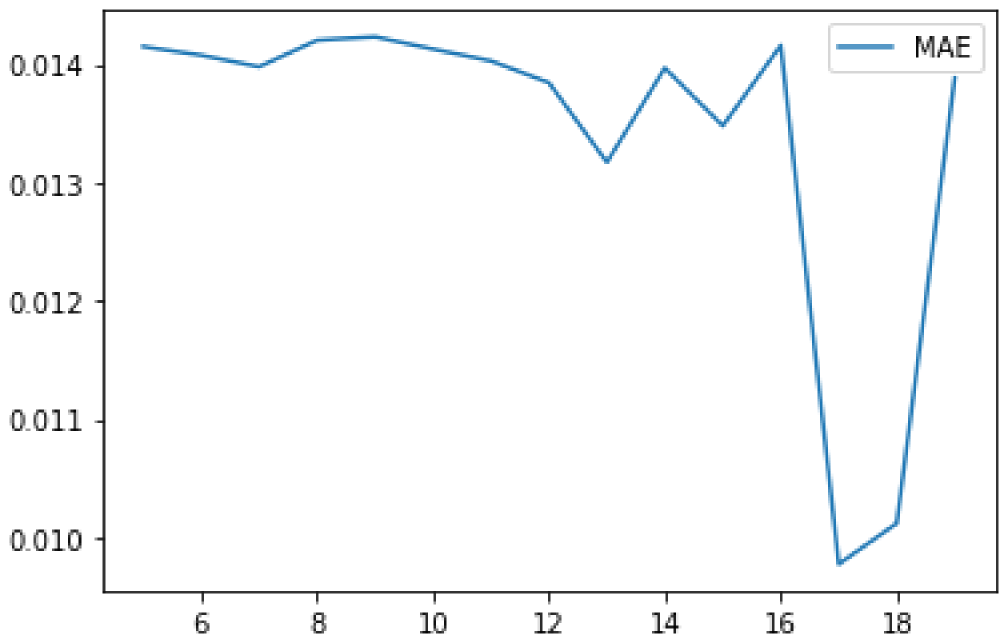
```
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 8/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 9/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 10/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 11/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 12/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 13/19
3170/3170 - 4s - loss: 0.0359 - 4s/epoch - 1ms/step
Epoch 14/19
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 15/19
3170/3170 - 4s - loss: 0.0358 - 4s/epoch - 1ms/step
Epoch 16/19
3170/3170 - 4s - loss: 0.0357 - 4s/epoch - 1ms/step
Epoch 17/19
3170/3170 - 4s - loss: 0.0356 - 4s/epoch - 1ms/step
Epoch 18/19
3170/3170 - 4s - loss: 0.0355 - 4s/epoch - 1ms/step
Epoch 19/19
3170/3170 - 4s - loss: 0.0353 - 4s/epoch - 1ms/step
```



[25]:
```
print(epochs_result.loc[epochs_result["MAE"] == epochs_result["MAE"].min()])
epochs_result.plot()
```

```
         MAE
17  0.009772
```

[25]: <AxesSubplot:>



In our model, one epoch means that we completed one forward algorithm and one backpropagation. With the increase of the number of epochs, we get better outcomes (lower errors), and then the errors would get higher at some points due to overfitting.

Here, I tested the number of epochs from 5 to 20, and concluded that when epochs = 17, we would get a the lowest error significantly, since it is really obvious as we can see in the plot above.

2.8 Conclusion

The parameters I choose hence are:

window length = 1

activation function = tanh

loss function = mse

dropout rate = 0.85

optimizer = nadam

neurons = 152

epochs = 17

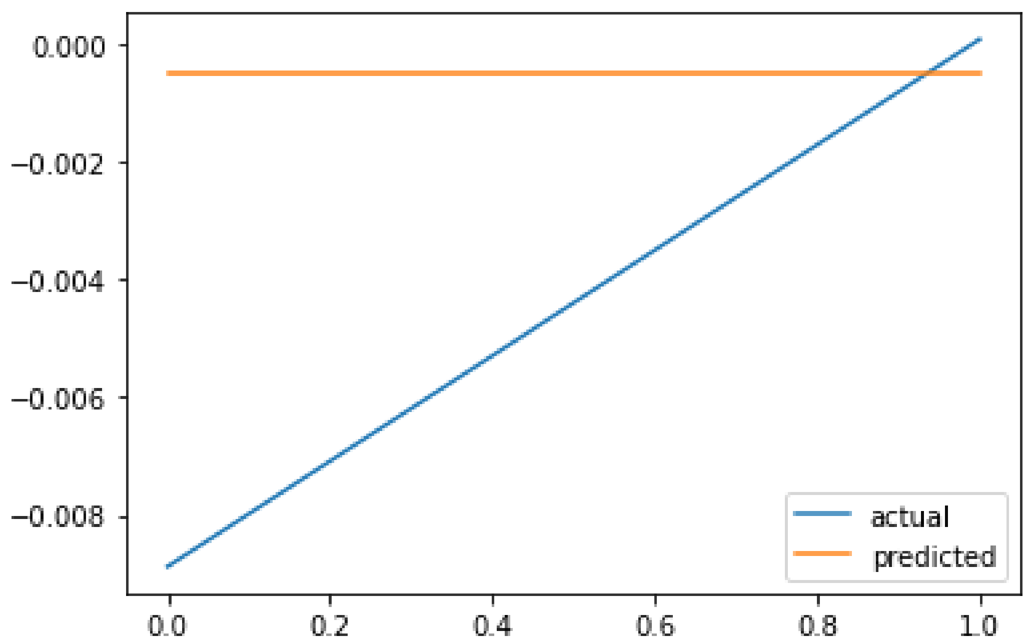
[26]:
```
window_len = window_result.loc[window_result["MAE"] == window_result["MAE"].
 ↪min()].index[0]
loss = loss_result.loc[loss_result["Error"] == loss_result["Error"].min()].
 ↪index[0]
activ_func = activation_result.loc[activation_result["MAE"] ==␣
 ↪activation_result["MAE"].min()].index[0]
dropout = dropout_result.loc[dropout_result["MAE"] == dropout_result["MAE"].
 ↪min()].index[0]
optimizer = optimizer_result.loc[optimizer_result["MAE"] ==␣
 ↪optimizer_result["MAE"].min()].index[0]
neurons = neurons_result.loc[neurons_result["MAE"] == neurons_result["MAE"].
 ↪min()].index[0]
epochs = epochs_result.loc[epochs_result["MAE"] == epochs_result["MAE"].min()].
 ↪index[0]
```

[27]:
```
def test_build_model(inputs, output_size, neurons, activ_func=activ_func,
                     dropout=dropout, loss=loss, optimizer=optimizer):
    model = Sequential()
    model.add(LSTM(neurons, input_shape=(inputs.shape[1], inputs.shape[2])))
    model.add(Dropout(dropout))
    model.add(Dense(units=output_size))
    model.add(Activation(activ_func))
    model.compile(loss=loss, optimizer=optimizer)
    return model

split_date = list(df["Date"][-(2*window_len+1):])[0]
print("split_date:",split_date)

# Split the training and test set
training_set, test_set = df[df['Date'] < split_date], df[df['Date'] >=␣
 ↪split_date]
training_set = training_set.drop(['Date','Label', 'OpenInt'], 1)
test_set = test_set.drop(['Date','Label','OpenInt'], 1)

# Create windows for training
LSTM_training_inputs = []
for i in range(len(training_set)-window_len):
    temp_set = training_set[i:(i+window_len)].copy()

    for col in list(temp_set):
        temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1
    LSTM_training_inputs.append(temp_set)

LSTM_training_inputs
LSTM_training_outputs = (training_set['Close'][window_len:].values/training_set[
```

```
    'Close'][:-window_len].values)-1

LSTM_training_inputs = [np.array(LSTM_training_input) for LSTM_training_input␣
 ↪in LSTM_training_inputs]
LSTM_training_inputs = np.array(LSTM_training_inputs)

# Create windows for testing
LSTM_test_inputs = []
for i in range(len(test_set)-window_len):
    temp_set = test_set[i:(i+window_len)].copy()

    for col in list(temp_set):
        temp_set[col] = temp_set[col]/temp_set[col].iloc[0] - 1

    LSTM_test_inputs.append(temp_set)
LSTM_test_outputs = (test_set['Close'][window_len:].values/test_set['Close'][:
 ↪-window_len].values)-1

LSTM_test_inputs = [np.array(LSTM_test_inputs) for LSTM_test_inputs in␣
 ↪LSTM_test_inputs]
LSTM_test_inputs = np.array(LSTM_test_inputs)

# initialise model architecture
nn_model = test_build_model(LSTM_training_inputs, output_size=1, neurons =␣
 ↪neurons)
# model output is next price normalised to 10th previous closing price train␣
 ↪model on data
# note: eth_history contains information on the training error per epoch
nn_history = nn_model.fit(LSTM_training_inputs, LSTM_training_outputs,
                            epochs=epochs, batch_size=1, verbose=2,␣
 ↪shuffle=True)
plt.plot(LSTM_test_outputs, label = "actual")
plt.plot(nn_model.predict(LSTM_test_inputs), label = "predicted")
plt.legend()
plt.show()
MAE = mean_absolute_error(LSTM_test_outputs, nn_model.predict(LSTM_test_inputs))
print("MAE: ", MAE)
```

```
split_date: 2017-11-08 00:00:00
Epoch 1/17
3197/3197 - 4s - loss: 3.8853e-04 - 4s/epoch - 1ms/step
Epoch 2/17
3197/3197 - 3s - loss: 3.6437e-04 - 3s/epoch - 871us/step
Epoch 3/17
3197/3197 - 3s - loss: 3.5731e-04 - 3s/epoch - 869us/step
Epoch 4/17
3197/3197 - 3s - loss: 3.6031e-04 - 3s/epoch - 847us/step
```

```
Epoch 5/17
3197/3197 - 3s - loss: 3.5913e-04 - 3s/epoch - 818us/step
Epoch 6/17
3197/3197 - 3s - loss: 3.5808e-04 - 3s/epoch - 858us/step
Epoch 7/17
3197/3197 - 3s - loss: 3.5748e-04 - 3s/epoch - 887us/step
Epoch 8/17
3197/3197 - 3s - loss: 3.6086e-04 - 3s/epoch - 864us/step
Epoch 9/17
3197/3197 - 3s - loss: 3.5506e-04 - 3s/epoch - 872us/step
Epoch 10/17
3197/3197 - 3s - loss: 3.5845e-04 - 3s/epoch - 883us/step
Epoch 11/17
3197/3197 - 3s - loss: 3.5793e-04 - 3s/epoch - 852us/step
Epoch 12/17
3197/3197 - 3s - loss: 3.5977e-04 - 3s/epoch - 835us/step
Epoch 13/17
3197/3197 - 3s - loss: 3.5791e-04 - 3s/epoch - 835us/step
Epoch 14/17
3197/3197 - 3s - loss: 3.5756e-04 - 3s/epoch - 839us/step
Epoch 15/17
3197/3197 - 3s - loss: 3.5714e-04 - 3s/epoch - 881us/step
Epoch 16/17
3197/3197 - 3s - loss: 3.5969e-04 - 3s/epoch - 1ms/step
Epoch 17/17
3197/3197 - 3s - loss: 3.5872e-04 - 3s/epoch - 1ms/step
```



MAE: 0.004486353821837541

3 Part 2

[28]:
```
def sigmoid(p):
    return 1/(1+math.exp(-p))

def backpropagation(x=1, y=4, z=5, target=np.array([0.1, 0.05]), lr=0.01):
    w1, w2, w3, w4, w5, w6, w7, w8, w9, w10 = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.
↪7, 0.8, 0.9, 0.1
    h_1, h_2 = sigmoid(w1*x + w3*y + w5*z + 0.5), sigmoid(w2*x + w4*y + w6*z +␣
↪0.5)
    o_1, o_2 = sigmoid(w7*h_1 + w9*h_2 + 0.5), sigmoid(w8*h_1 + w10*h_2 + 0.5)
    sse = 1/2 * sum((target-np.array([o_1, o_2]))**2)
    sse_o1, sse_o2 = o_1 - target[0], o_2 - target[1]
    o1_n1, o2_n2 = o_1 * (1 - o_1), o_2 * (1 - o_2)
    n1_w7, n1_w9, n2_w8, n2_w10 = h_1, h_2, h_1, h_2
    sse_w7, sse_w8 = sse_o1 * o1_n1 * n1_w7, sse_o2 * o2_n2 * n2_w8
    sse_w9, sse_w10 = sse_o1 * o1_n1 * n1_w9, sse_o2 * o2_n2 * n2_w10
    new_w7, new_w8, new_w9, new_w10 = w7 + lr * n1_w7, w8 + lr * n2_w8, w9+ lr␣
↪* n1_w9, w10 + lr * n2_w10
    h1_m1, h2_m2 = h_1 * (1 - h_1), h_2 * (1 - h_2)
    m1_w1, m1_w3, m1_w5, m2_w2, m2_w4, m2_w6 = w1, w3, w5, w2, w4, w6
    h1_w1, h1_w3, h1_w5 = h1_m1 * m1_w1, h1_m1 * m1_w3, h1_m1 * m1_w5
    h2_w2, h2_w4, h2_w6 = h2_m2 * m2_w2, h2_m2 * m2_w4, h2_m2 * m2_w6
    n1_h1, n1_h2, n2_h1, n2_h2 = w7, w9, w8, w10
    sse_h1, sse_h2 = o1_n1 * n1_h1 + o2_n2 * n2_h1, o1_n1 * n1_h2 + o2_n2 *␣
↪n2_h2
    sse_m1, sse_m2 = sse_h1 * h1_m1, sse_h2 * h2_m2
    sse_w1, sse_w3, sse_w5 = sse_h1 * h1_w1, sse_h1 * h1_w3, sse_h1 * h1_w5
    sse_w2, sse_w4, sse_w6 = sse_h2 * h2_w2, sse_h2 * h2_w4, sse_h2 * h2_w6
    new_w1, new_w3, new_w5 = w1 + lr * sse_w1, w3 + lr * sse_w3, w5 + lr *␣
↪sse_w5
    new_w2, new_w4, new_w6 = w2 + lr * sse_w2, w4 + lr * sse_w4, w6 + lr *␣
↪sse_w6
    return {"new w1": new_w1, "new w2": new_w2, "new w3": new_w3, "new w4":␣
↪new_w4, "new w5": new_w5,
            "new w6": new_w6, "new w7": new_w7, "new w8": new_w8, "new w9":␣
↪new_w9, "new w10": new_w10}

backpropagation()
```

[28]:
```
{'new w1': 0.10000259641466082,
 'new w2': 0.2000010319300706,
 'new w3': 0.3000077892439824,
 'new w4': 0.4000020638601412,
```

```
 'new w5': 0.500012982073304,
 'new w6': 0.6000030957902117,
 'new w7': 0.7098661308217233,
 'new w8': 0.8098661308217234,
 'new w9': 0.9099503319834995,
 'new w10': 0.10995033198349943}
```

[ ]: