LSTNet-For-Cryptocurrency-Market-Prediction

June 11, 2020

0.1 1. Setup

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objs as go
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
from tqdm.notebook import tqdm
from datetime import datetime, timedelta

import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
```

[3]:
```
device = "cuda:0" if torch.cuda.is_available() else "cpu"
device = torch.device(device)
```

0.2 2. Load data

[4]:
```
data_path = "Data/BNB_BTC_1h.csv"
data = pd.read_csv(data_path)
print(data.shape)
data.head()
```

```
(21461, 12)
```

[4]:

	Open-Time	Open	High	Low	Close	Volume	\
0	2018-01-01 00:00:00	0.000623	0.000625	0.000614	0.000618	130791.0	
1	2018-01-01 01:00:00	0.000618	0.000625	0.000611	0.000622	89072.0	
2	2018-01-01 02:00:00	0.000622	0.000625	0.000620	0.000622	101296.0	
3	2018-01-01 03:00:00	0.000622	0.000624	0.000619	0.000622	83701.0	
4	2018-01-01 04:00:00	0.000622	0.000628	0.000622	0.000625	103000.0	

	Close-Time	Quote-Asset-Value	Number-of-Trades	\
0	2018-01-01 00:59:59	80.955613	3328.0	

```
1  2018-01-01 01:59:59          55.013318            2653.0
2  2018-01-01 02:59:59          63.078538            2663.0
3  2018-01-01 03:59:59          52.015837            2726.0
4  2018-01-01 04:59:59          64.458631            2921.0

   Taker-Buy-Base-Asset-Volume  Taker-Buy-Quote-Asset-Volume  Ignore
0                      54856.0                     33.995884     0.0
1                      39926.0                     24.693081     0.0
2                      59607.0                     37.145069     0.0
3                      53257.0                     33.115652     0.0
4                      55663.0                     34.841113     0.0
```

[5]:
```
columns = data.columns[:7]
data = data[columns]
data.head()
```

[5]:
```
             Open-Time      Open      High       Low     Close     Volume  \
0  2018-01-01 00:00:00  0.000623  0.000625  0.000614  0.000618  130791.0
1  2018-01-01 01:00:00  0.000618  0.000625  0.000611  0.000622   89072.0
2  2018-01-01 02:00:00  0.000622  0.000625  0.000620  0.000622  101296.0
3  2018-01-01 03:00:00  0.000622  0.000624  0.000619  0.000622   83701.0
4  2018-01-01 04:00:00  0.000622  0.000628  0.000622  0.000625  103000.0

            Close-Time
0  2018-01-01 00:59:59
1  2018-01-01 01:59:59
2  2018-01-01 02:59:59
3  2018-01-01 03:59:59
4  2018-01-01 04:59:59
```

[6]:
```
data.describe()
```

[6]:
```
                Open          High           Low         Close        Volume
count  21320.000000  21320.000000  21320.000000  21320.000000  2.132000e+04
mean       0.002113      0.002125      0.002100      0.002113  7.988280e+04
std        0.000804      0.000809      0.000799      0.000804  9.706676e+04
min        0.000567      0.000570      0.000529      0.000568  0.000000e+00
25%        0.001540      0.001549      0.001530      0.001540  3.322559e+04
50%        0.001935      0.001944      0.001926      0.001935  5.791882e+04
75%        0.002371      0.002385      0.002356      0.002371  9.633458e+04
max        0.004738      0.004813      0.004680      0.004736  5.178274e+06
```

0.3 3. Clean data

0.3.1 3.1 Deal NaN

[7]:
```
na_mask = data.isna().any(axis=1)
print(na_mask.sum())
data[na_mask].head()
```

```
141
```

[7]:

	Open-Time	Open	High	Low	Close	Volume	Close-Time
76	2018-01-04 04:00:00	NaN	NaN	NaN	NaN	NaN	2018-01-04 04:59:59
913	2018-02-08 01:00:00	NaN	NaN	NaN	NaN	NaN	2018-02-08 01:59:59
914	2018-02-08 02:00:00	NaN	NaN	NaN	NaN	NaN	2018-02-08 02:59:59
915	2018-02-08 03:00:00	NaN	NaN	NaN	NaN	NaN	2018-02-08 03:59:59
916	2018-02-08 04:00:00	NaN	NaN	NaN	NaN	NaN	2018-02-08 04:59:59

As 141 is very small compared to 21461 and DL being robust to such noise, I am just filling Nan with 'ffill'

[8]:
```
data = data.ffill()
print(data.isna().any(axis=1).sum())
data[na_mask].head()
```

```
0
```

[8]:

	Open-Time	Open	High	Low	Close	Volume	\
76	2018-01-04 04:00:00	0.000612	0.000612	0.000611	0.000612	130.0	
913	2018-02-08 01:00:00	0.001064	0.001073	0.001059	0.001063	42985.9	
914	2018-02-08 02:00:00	0.001064	0.001073	0.001059	0.001063	42985.9	
915	2018-02-08 03:00:00	0.001064	0.001073	0.001059	0.001063	42985.9	
916	2018-02-08 04:00:00	0.001064	0.001073	0.001059	0.001063	42985.9	

	Close-Time
76	2018-01-04 04:59:59
913	2018-02-08 01:59:59
914	2018-02-08 02:59:59
915	2018-02-08 03:59:59
916	2018-02-08 04:59:59

0.3.2 3.2 Clean sequence

[9]:
```
data['Open-Time'] = pd.to_datetime(data['Open-Time'])
data = data.sort_values(by=['Open-Time'])
start_dt = data.loc[0, 'Open-Time']
end_dt = data.loc[len(data)-1, 'Open-Time']
expected_end_dt = start_dt + timedelta(hours=len(data) - 1)
print(start_dt)
print(expected_end_dt)
```

```
print(end_dt)
```

```
2018-01-01 00:00:00
2020-06-13 04:00:00
2020-06-11 09:00:00
```

[10]:
```
print(len(data))
print(data['Open-Time'].nunique())
```

```
21461
21461
```

There are no duplicates. But end date expectation is not matching. Lets dig deeper

[11]:
```
data['Open-Time-Shifted'] = data['Open-Time'].shift(periods=1)
data.loc[1:, 'Seconds-Lag'] = (data['Open-Time'] - data['Open-Time-Shifted'])[1:
 ↪].apply(timedelta.total_seconds)
data.head()
```

[11]:

	Open-Time	Open	High	Low	Close	Volume	\
0	2018-01-01 00:00:00	0.000623	0.000625	0.000614	0.000618	130791.0	
1	2018-01-01 01:00:00	0.000618	0.000625	0.000611	0.000622	89072.0	
2	2018-01-01 02:00:00	0.000622	0.000625	0.000620	0.000622	101296.0	
3	2018-01-01 03:00:00	0.000622	0.000624	0.000619	0.000622	83701.0	
4	2018-01-01 04:00:00	0.000622	0.000628	0.000622	0.000625	103000.0	

	Close-Time	Open-Time-Shifted	Seconds-Lag
0	2018-01-01 00:59:59	NaT	NaN
1	2018-01-01 01:59:59	2018-01-01 00:00:00	3600.0
2	2018-01-01 02:59:59	2018-01-01 01:00:00	3600.0
3	2018-01-01 03:59:59	2018-01-01 02:00:00	3600.0
4	2018-01-01 04:59:59	2018-01-01 03:00:00	3600.0

[12]:
```
data['Seconds-Lag'].describe()
```

[12]:
```
count    21460.000000
mean      3592.786580
std        113.919953
min       1694.000000
25%       3600.000000
50%       3600.000000
75%       3600.000000
max       3600.000000
Name: Seconds-Lag, dtype: float64
```

[13]:
```
mask = data['Seconds-Lag'] < 3600
print(mask.sum())
data[mask].head(n=10)
```

86

[13]:
```
                Open-Time      Open      High       Low     Close     Volume  \
946 2018-02-09 09:28:14  0.001063  0.001110  0.001000  0.001080  264762.75
947 2018-02-09 10:00:00  0.001063  0.001110  0.001000  0.001080  264762.75
948 2018-02-09 10:28:14  0.001080  0.001080  0.001050  0.001066  247583.54
949 2018-02-09 11:00:00  0.001080  0.001080  0.001050  0.001066  247583.54
950 2018-02-09 11:28:14  0.001066  0.001078  0.001050  0.001077  249736.63
951 2018-02-09 12:00:00  0.001066  0.001078  0.001050  0.001077  249736.63
952 2018-02-09 12:28:14  0.001076  0.001082  0.001072  0.001078  266598.78
953 2018-02-09 13:00:00  0.001076  0.001082  0.001072  0.001078  266598.78
954 2018-02-09 13:28:14  0.001079  0.001136  0.001078  0.001107  475272.41
955 2018-02-09 14:00:00  0.001079  0.001136  0.001078  0.001107  475272.41

                Close-Time   Open-Time-Shifted  Seconds-Lag
946  2018-02-09 10:28:14 2018-02-09 09:00:00       1694.0
947  2018-02-09 10:59:59 2018-02-09 09:28:14       1906.0
948  2018-02-09 11:28:14 2018-02-09 10:00:00       1694.0
949  2018-02-09 11:59:59 2018-02-09 10:28:14       1906.0
950  2018-02-09 12:28:14 2018-02-09 11:00:00       1694.0
951  2018-02-09 12:59:59 2018-02-09 11:28:14       1906.0
952  2018-02-09 13:28:14 2018-02-09 12:00:00       1694.0
953  2018-02-09 13:59:59 2018-02-09 12:28:14       1906.0
954  2018-02-09 14:28:14 2018-02-09 13:00:00       1694.0
955  2018-02-09 14:59:59 2018-02-09 13:28:14       1906.0
```

[14]:
```
irregular_time_mask =  data['Open-Time'].dt.minute != 0
print(irregular_time_mask.sum())
print(data.shape)
data = data[np.logical_not(irregular_time_mask)]
data = data.reset_index(drop=True)
print(data.shape)
```

```
43
(21461, 9)
(21418, 9)
```

[15]:
```
data['Open-Time'] = pd.to_datetime(data['Open-Time'])
data = data.sort_values(by=['Open-Time'])
start_dt = data.loc[0, 'Open-Time']
end_dt = data.loc[len(data)-1, 'Open-Time']
expected_end_dt = start_dt + timedelta(hours=len(data) - 1)
print(start_dt)
print(expected_end_dt)
print(end_dt)
```

```
2018-01-01 00:00:00
2020-06-11 09:00:00
```

```
2020-06-11 09:00:00
```

[16]:
```
columns = data.columns[:6]
data = data[columns]
print(data.shape)
data.head()
```

```
(21418, 6)
```

[16]:

	Open-Time	Open	High	Low	Close	Volume
0	2018-01-01 00:00:00	0.000623	0.000625	0.000614	0.000618	130791.0
1	2018-01-01 01:00:00	0.000618	0.000625	0.000611	0.000622	89072.0
2	2018-01-01 02:00:00	0.000622	0.000625	0.000620	0.000622	101296.0
3	2018-01-01 03:00:00	0.000622	0.000624	0.000619	0.000622	83701.0
4	2018-01-01 04:00:00	0.000622	0.000628	0.000622	0.000625	103000.0

0.4 4. Define train dataset

[17]:
```
train_max_open_time = datetime.strptime('2020-05-31 23:00:00', '%Y-%m-%d %H:%M:
↪%S')
train_data = data[data['Open-Time'] <= train_max_open_time].copy()
print(len(data), len(train_data))
train_data.tail()
```

```
21418 21168
```

[17]:

	Open-Time	Open	High	Low	Close	Volume
21163	2020-05-31 19:00:00	0.001827	0.001830	0.001825	0.001830	15025.47
21164	2020-05-31 20:00:00	0.001830	0.001830	0.001822	0.001829	21273.65
21165	2020-05-31 21:00:00	0.001829	0.001829	0.001801	0.001817	48774.92
21166	2020-05-31 22:00:00	0.001817	0.001826	0.001811	0.001812	42385.93
21167	2020-05-31 23:00:00	0.001813	0.001816	0.001793	0.001808	81678.70

[18]:
```
min_price = train_data['Low'].min()
max_price = train_data['High'].max()
min_volume = train_data['Volume'].min()
max_volume = train_data['Volume'].max()
print(min_price, max_price)
print(min_volume, max_volume)
```

```
0.000529 0.004813399999999999
0.0 5178273.91
```

[19]:
```
train_data.loc[:, 'Open-N'] = (train_data['Open'] - min_price) / (max_price-␣
↪min_price)
train_data.loc[:, 'High-N'] = (train_data['High'] - min_price) / (max_price-␣
↪min_price)
```

```
train_data.loc[:, 'Low-N'] = (train_data['Low'] - min_price) / (max_price-␣
↪min_price)
train_data.loc[:, 'Close-N'] = (train_data['Close'] - min_price) / (max_price-␣
↪min_price)
train_data.loc[:, 'Volume-N'] = (train_data['Volume'] - min_volume) /␣
↪(max_volume- min_volume)
```

[20]:
```
class MarketDataset(Dataset):

    def __init__(self, data, history_len):
        self.data = data
        self.history_len = history_len

    def __len__(self):
        self.len = len(self.data) - self.history_len
        return self.len

    def __getitem__(self, index):
        x_cols = ['Open-N', 'High-N', 'Low-N', 'Close-N', 'Volume-N']
        y_cols = ['Close-N']
        x = self.data.iloc[index: index+self.history_len, :][x_cols].values
        y = self.data.iloc[index+self.history_len, :][y_cols].values.
↪astype('float')
        x = torch.tensor(x).float()
        y = torch.tensor(y).float()
        return x, y
```

[21]:
```
# verify dataset instances
train_dataset = MarketDataset(train_data, history_len=48)
print(len(train_dataset))
x, y = train_dataset[21118]
print(x.shape, y.shape)
```

```
21120
torch.Size([48, 5]) torch.Size([1])
```

[22]:
```
# test dataset instances
# print(x, y)
# train_data.tail()
```

[23]:
```
# test data_loader
train_data_loader = DataLoader(train_dataset, batch_size=4)
X, Y = next(iter(train_data_loader))
print(X.shape, Y.shape)
```

```
torch.Size([4, 48, 5]) torch.Size([4, 1])
```

0.5 5. Define model

[24]:
```
class LSTNet(nn.Module):

    def __init__(self):
        super(LSTNet, self).__init__()
        self.num_features = 5
        self.conv1_out_channels = 32
        self.conv1_kernel_height = 7
        self.recc1_out_channels = 64
        self.skip_steps = [4, 24]
        self.skip_reccs_out_channels = [4, 4]
        self.output_out_features = 1
        self.ar_window_size = 7
        self.dropout = nn.Dropout(p = 0.2)


        self.conv1 = nn.Conv2d(1, self.conv1_out_channels,
                               kernel_size=(self.conv1_kernel_height, self.
↪num_features))
        self.recc1 = nn.GRU(self.conv1_out_channels, self.recc1_out_channels,␣
↪batch_first=True)
        self.skip_reccs = {}
        for i in range(len(self.skip_steps)):
            self.skip_reccs[i] = nn.GRU(self.conv1_out_channels, self.
↪skip_reccs_out_channels[i], batch_first=True)
        self.output_in_features = self.recc1_out_channels + np.dot(self.
↪skip_steps, self.skip_reccs_out_channels)
        self.output = nn.Linear(self.output_in_features, self.
↪output_out_features)
        if self.ar_window_size > 0:
            self.ar = nn.Linear(self.ar_window_size, 1)

    def forward(self, X):
        """
        Parameters:
        X (tensor) [batch_size, time_steps, num_features]
        """
        batch_size = X.size(0)

        # Convolutional Layer
        C = X.unsqueeze(1) # [batch_size, num_channels=1, time_steps,␣
↪num_features]
        C = F.relu(self.conv1(C)) # [batch_size, conv1_out_channels,␣
↪shrinked_time_steps, 1]
        C = self.dropout(C)
```

```
        C = torch.squeeze(C, 3) # [batch_size, conv1_out_channels,␣
↪shrinked_time_steps]

        # Recurrent Layer
        R = C.permute(0, 2, 1) # [batch_size, shrinked_time_steps,␣
↪conv1_out_channels]
        out, hidden = self.recc1(R) # [batch_size, shrinked_time_steps,␣
↪recc_out_channels]
        R = out[:, -1, :] # [batch_size, recc_out_channels]
        R = self.dropout(R)
        #print(R.shape)

        # Skip Recurrent Layers
        shrinked_time_steps = C.size(2)
        for i in range(len(self.skip_steps)):
            skip_step = self.skip_steps[i]
            skip_sequence_len = shrinked_time_steps // skip_step
            # shrinked_time_steps shrinked further
            S = C[:, :, -skip_sequence_len*skip_step:] # [batch_size,␣
↪conv1_out_channels, shrinked_time_steps]
            S = S.view(S.size(0), S.size(1), skip_sequence_len, skip_step) #␣
↪[batch_size, conv1_out_channels, skip_sequence_len,␣
↪skip_step=num_skip_components]
            # note that num_skip_components = skip_step
            S = S.permute(0, 3, 2, 1).contiguous() # [batch_size,␣
↪skip_step=num_skip_components, skip_sequence_len, conv1_out_channels]
            S = S.view(S.size(0)*S.size(1), S.size(2), S.size(3))  #␣
↪[batch_size*num_skip_components, skip_sequence_len, conv1_out_channels]
            out, hidden = self.skip_reccs[i](S) #␣
↪[batch_size*num_skip_components, skip_sequence_len,␣
↪skip_reccs_out_channels[i]]
            S = out[:, -1, :] # [batch_size*num_skip_components,␣
↪skip_reccs_out_channels[i]]
            S = S.view(batch_size, skip_step*S.size(1)) # [batch_size,␣
↪num_skip_components*skip_reccs_out_channels[i]]
            S = self.dropout(S)
            R = torch.cat((R, S), 1) # [batch_size, recc_out_channels +␣
↪skip_reccs_out_channels * num_skip_components]
            #print(S.shape)
        #print(R.shape)

        # Output Layer
        O = F.relu(self.output(R)) # [batch_size, output_out_features=1]

        if self.ar_window_size > 0:
            # set dim3 based on output_out_features
```

```
            AR = X[:, -self.ar_window_size:, 3:4] # [batch_size,␣
 ↪ar_window_size, output_out_features=1]
            AR = AR.permute(0, 2, 1).contiguous() # [batch_size,␣
 ↪output_out_features, ar_window_size]
            AR = self.ar(AR) # [batch_size, output_out_features, 1]
            AR = AR.squeeze(2) # [batch_size, output_out_features]
            O = O + AR

        return O
```

[25]:
```
# test model
model = LSTNet()

for X, Y in train_data_loader:
    print(X.shape)
    out = model(X)
    print(Y.shape, out.shape)
    break
```

```
torch.Size([4, 48, 5])
torch.Size([4, 1]) torch.Size([4, 1])
```

0.6 6. Train model

[26]:
```
history_len = 48
batch_size = 8

epochs = 10

lr = 0.01
weight_decay = 0.01
```

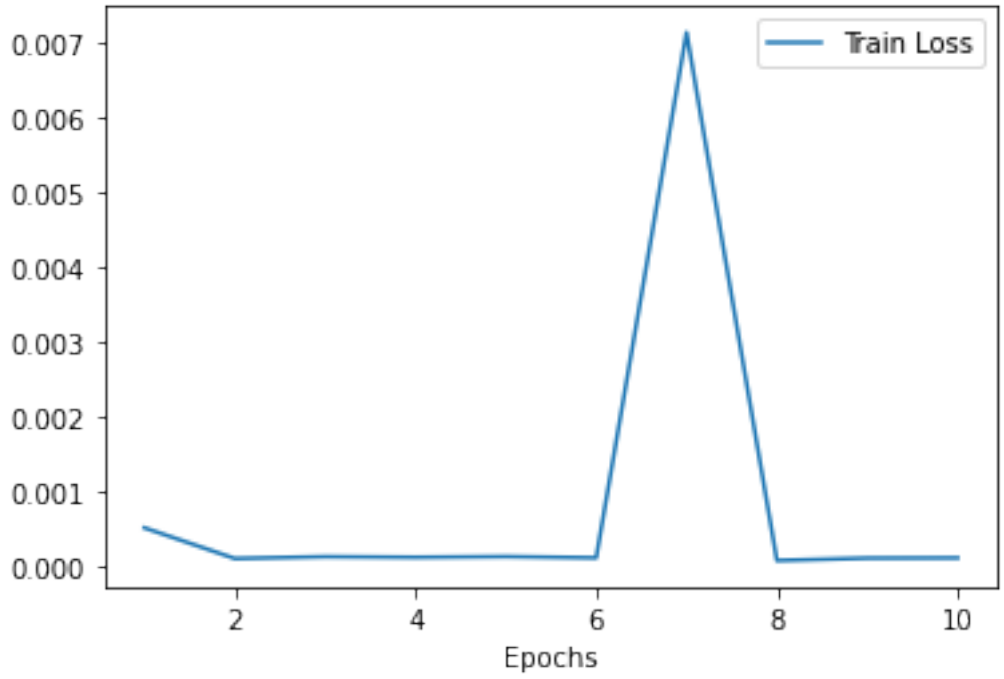
[27]:
```
train_dataset = MarketDataset(train_data, history_len=history_len)
train_data_loader = DataLoader(train_dataset, batch_size=batch_size,␣
 ↪shuffle=True)

model = LSTNet()

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
```

[27]:
```
train_loss_list = []
for epoch in tqdm(range(epochs)):

    epoch_loss_train = 0
    for i, batch in tqdm(enumerate(train_data_loader, start=1),
```

```
                        leave=False, desc="Train",␣
↪total=len(train_data_loader)):

        X, Y = batch
        optimizer.zero_grad()
        Y_pred = model(X)
        loss = criterion(Y_pred, Y)
        loss.backward()
        optimizer.step()

        with open('Log/Running-Loss.txt', 'a+') as file:
            file.write(f'{loss.item()}\n')
        epoch_loss_train += loss.item()

    epoch_loss_train = epoch_loss_train / len(train_data_loader)
    train_loss_list.append(epoch_loss_train)

    with open('Log/Epoch-Loss.txt', 'a+') as file:
        file.write(f'{epoch_loss_train}\n')
```

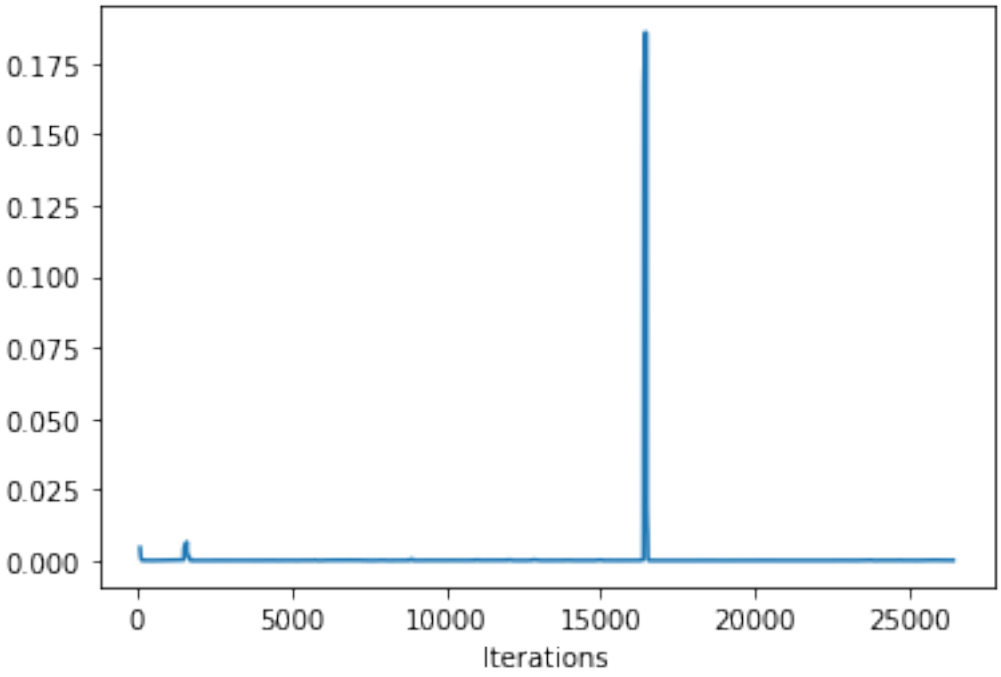
```
HBox(children=(FloatProgress(value=0.0, max=10.0), HTML(value='')))

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de

HBox(children=(FloatProgress(value=0.0, description='Train', max=2640.0, style=ProgressStyle(de
```

[44]:
```
plt.plot(range(1, len(train_loss_list)+1), train_loss_list, label='Train Loss')
plt.xlabel("Epochs")
plt.legend()
plt.show()
```



[29]:
```
with open('Log/Running-Loss.txt', 'r') as file:
    running_loss = pd.Series(file.readlines())
    running_loss_ma = running_loss.rolling(window=100).mean()
    print(len(running_loss_ma))
    plt.plot(running_loss_ma, label='Running Loss')
    plt.xlabel("Iterations")
    plt.show()
```

26400



[30]:
```
model_name = f"LSTNet-E{epochs}"
model_path = f"Models/{model_name}.pth"
torch.save(model.state_dict(), model_path)
```

0.7 7. Load model

[28]:
```
model_name = f"LSTNet-E{epochs}"
model_path = f"Models/{model_name}.pth"
model = LSTNet()
model.load_state_dict(torch.load(model_path))
model.eval()
```

[28]:
```
LSTNet(
  (dropout): Dropout(p=0.2, inplace=False)
  (conv1): Conv2d(1, 32, kernel_size=(7, 5), stride=(1, 1))
  (recc1): GRU(32, 64, batch_first=True)
  (output): Linear(in_features=176, out_features=1, bias=True)
  (ar): Linear(in_features=7, out_features=1, bias=True)
)
```

0.8 8. Make predictions

Predict hourly OHLCV from Jun 1 2020 to Jun 11 2020

[29]:
```
data.loc[:, 'Open-N'] = (data['Open'] - min_price) / (max_price- min_price)
data.loc[:, 'High-N'] = (data['High'] - min_price) / (max_price- min_price)
data.loc[:, 'Low-N'] = (data['Low'] - min_price) / (max_price- min_price)
data.loc[:, 'Close-N'] = (data['Close'] - min_price) / (max_price- min_price)
data.loc[:, 'Volume-N'] = (data['Volume'] - min_volume) / (max_volume-␣
↪min_volume)
data.head()
```

[29]:

	Open-Time	Open	High	Low	Close	Volume
0	2018-01-01 00:00:00	0.000623	0.000625	0.000614	0.000618	130791.0
1	2018-01-01 01:00:00	0.000618	0.000625	0.000611	0.000622	89072.0
2	2018-01-01 02:00:00	0.000622	0.000625	0.000620	0.000622	101296.0
3	2018-01-01 03:00:00	0.000622	0.000624	0.000619	0.000622	83701.0
4	2018-01-01 04:00:00	0.000622	0.000628	0.000622	0.000625	103000.0

	Open-N	High-N	Low-N	Close-N	Volume-N
0	0.021938	0.022318	0.019779	0.020873	0.025258
1	0.020873	0.022388	0.019193	0.021767	0.017201
2	0.021770	0.022493	0.021181	0.021681	0.019562
3	0.021679	0.022148	0.020932	0.021809	0.016164
4	0.021753	0.023107	0.021676	0.022391	0.019891

[30]:
```
batch_size = 16
dataset = MarketDataset(data, history_len=history_len)
data_loader = DataLoader(dataset, batch_size=batch_size, shuffle=False)
```

[31]:
```
for i, batch in tqdm(enumerate(data_loader, start=1),leave=False,␣
↪total=len(data_loader)):

    X, Y = batch
    Y_pred = model(X).detach().numpy()
    if i == 1:
        predictions = Y_pred
    else:
        predictions = np.concatenate((predictions, Y_pred), axis=0)

#predictions[:, 0:4] = predictions[:, 0:4] * (max_price - min_price) + min_price
#predictions[:, 4] = predictions[:, 4] * (max_volume - min_volume) + min_volume
predictions = predictions * (max_price - min_price) + min_price

#columns = ['Open', 'High', 'Low', 'Close', 'Volume']
columns = ['Close']
predictions = pd.DataFrame(predictions, columns=columns)
print(predictions.shape)
predictions.head()
```

```
HBox(children=(FloatProgress(value=0.0, max=1336.0), HTML(value='')))
```

```
(21370, 1)
```

[31]:
```
      Close
0  0.000621
1  0.000616
2  0.000617
3  0.000612
4  0.000610
```

[32]:
```
print(data.shape)
print(predictions.shape)
```

```
(21418, 11)
(21370, 1)
```

[33]:
```
cols = ['Open-Time', 'Open', 'High', 'Low', 'Close', 'Volume']
data = data[cols]
data.tail()
```

[33]:

	Open-Time	Open	High	Low	Close	Volume
21413	2020-06-11 05:00:00	0.001763	0.001767	0.001762	0.001762	29215.63
21414	2020-06-11 06:00:00	0.001762	0.001769	0.001760	0.001766	18447.87
21415	2020-06-11 07:00:00	0.001766	0.001769	0.001764	0.001769	49849.23
21416	2020-06-11 08:00:00	0.001769	0.001771	0.001767	0.001770	39773.38
21417	2020-06-11 09:00:00	0.001770	0.001771	0.001768	0.001770	3454.59

[34]:
```
predictions['Open-Time'] = pd.Series(predictions.index).apply(lambda x:␣
↪data['Open-Time'][x+history_len])
cols = ['Open-Time', 'Close']
predictions = predictions[cols]
predictions.tail()
```

[34]:

	Open-Time	Close
21365	2020-06-11 05:00:00	0.001759
21366	2020-06-11 06:00:00	0.001759
21367	2020-06-11 07:00:00	0.001760
21368	2020-06-11 08:00:00	0.001761
21369	2020-06-11 09:00:00	0.001762

0.9 9. Visualise Predictions

[35]:
```
min_time = datetime.strptime('2020-06-01 00:00:00', '%Y-%m-%d %H:%M:%S')
max_time = datetime.strptime('2020-06-10 23:00:00', '%Y-%m-%d %H:%M:%S')
data_mask = (data['Open-Time'] >= min_time) & (data['Open-Time'] <= max_time)
predictions_mask = (predictions['Open-Time'] >= min_time) &␣
↪(predictions['Open-Time'] <= max_time)
```

```
print(data_mask.sum(), predictions_mask.sum())
```

```
240 240
```

[36]:
```
actuals_df = data[data_mask].copy().set_index('Open-Time', drop=True)
predictions_df = predictions[predictions_mask].copy().set_index('Open-Time',␣
↪drop=True)
```

[37]:
```
actuals_df.head()
```

[37]:

Open-Time	Open	High	Low	Close	Volume
2020-06-01 00:00:00	0.001809	0.001813	0.001800	0.001812	73009.76
2020-06-01 01:00:00	0.001811	0.001826	0.001808	0.001824	54387.68
2020-06-01 02:00:00	0.001824	0.001834	0.001822	0.001834	64901.54
2020-06-01 03:00:00	0.001834	0.001836	0.001822	0.001826	55693.64
2020-06-01 04:00:00	0.001825	0.001825	0.001814	0.001815	50167.58

[38]:
```
predictions_df.head()
```

[38]:

Open-Time	Close
2020-06-01 00:00:00	0.001818
2020-06-01 01:00:00	0.001815
2020-06-01 02:00:00	0.001815
2020-06-01 03:00:00	0.001816
2020-06-01 04:00:00	0.001815

[39]:
```
lines = []

line = {
    'x':actuals_df.index,
    'y':actuals_df['Close'],
    'name':'Close-Actuals'
}
lines.append(line)

line = {
    'x':predictions_df.index,
    'y':predictions_df['Close'],
    'name':'Close-Predictions'
}
lines.append(line)
```

[40]:
```
fig = go.Figure(lines)
iplot(fig)
```

[42]:
```
from IPython.display import Image
Image(filename='Prediction-Visualisation.png')
```

[42]:

[ ]: