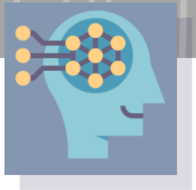


Anomaly Detection on Stock Price Data using LSTM Autoencoder

Faiza Qayyum
AD-20206810

Table of Content

- **Introduction**
- **Dataset**
- **Implementation**
- **LSTM explanation**
- **Results**

Project Source

https://github.com/alind-saxena/Anomaly_Detection/blob/main/Data%20Science/Anomaly%20Detection%20On%20Time%20Series%20Data%20-%20LSTM%20Autoencoder.ipynb

Introduction

- **Anomaly detection,** also known as ***outlier detection*** is the process of identifying *extreme points or observations* that are significantly deviating from the remaining data.
- Usually, these extreme points do have some exciting story to tell, by analyzing them, one can understand the extreme working conditions of the system.
- Some of the anomalies could be banking fraud in terms of transactions, or a sudden increase in the failure rate of fintech transactions due to the new software upgrade or surprising increase in purchase.

Supervised Anomaly Detection:

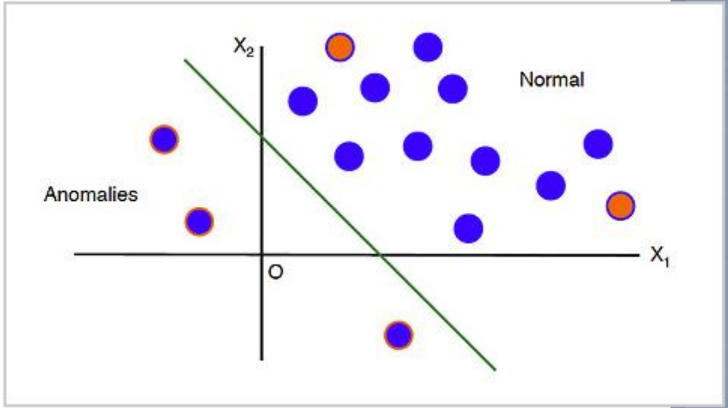
- Supervised learning is the scenario in which the model is trained on the labeled data, and trained model will predict the unseen data

Unsupervised Anomaly Detection:

- Whereas in unsupervised learning, no labels are available

Semi-supervised Anomaly Detection:

- Combination of supervised and unsupervised data



Steps to follow for Anomaly Detection

1. Construct an LSTM Autoencoder on the stock price data, assuming there are no anomalies.

2. Generate the error threshold on training dataset.

3. Detect Anomaly using the threshold on test dataset.

Data Set Overview

- Alphabet Inc. (Google) dataset.
- Stock prices on daily time interval
- The time period used is:

'2004-08-20', to '2020-12-24').

Date	Open	High	Low	Close	Adj Close	Volume
4/13/2021	2261.469971	2277.20996	2256.090088	2267.27002	2267.27002	1165500
4/14/2021	2275.159912	2277.98999	2249.189941	2254.840088	2254.840088	1011000
4/15/2021	2276.97998	2306.59692	2266	2296.659912	2296.659912	1373600
4/16/2021	2303	2306.43994	2284.449951	2297.76001	2297.76001	1129800
4/19/2021	2291.97998	2318.44995	2287.844971	2302.399902	2302.399902	1234400
4/20/2021	2307.889893	2309.6001	2271.709961	2293.629883	2293.629883	1088700
4/21/2021	2285.25	2295.32007	2258.570068	2293.290039	2293.290039	1196500
4/22/2021	2293.22998	2303.76196	2256.449951	2267.919922	2267.919922	1054800
4/23/2021	2283.469971	2325.82007	2278.209961	2315.300049	2315.300049	1433500
4/26/2021	2319.929932	2341.26001	2313.840088	2326.73999	2326.73999	1041700
4/27/2021	2336	2337.44995	2304.27002	2307.120117	2307.120117	1598600
4/28/2021	2407.14502	2452.37793	2374.850098	2379.909912	2379.909912	2986400
4/29/2021	2410.330078	2436.52002	2402.280029	2429.889893	2429.889893	1977700
4/30/2021	2404.48999	2427.13989	2402.159912	2410.120117	2410.120117	1957100
5/3/2021	2402.719971	2419.69995	2384.5	2395.169922	2395.169922	1689400
5/4/2021	2369.73999	2379.26001	2311.699951	2354.25	2354.25	1756000
5/5/2021	2368.419922	2382.19995	2351.409912	2356.73999	2356.73999	1090300
5/6/2021	2350.639893	2382.70996	2342.337891	2381.350098	2381.350098	1030900
5/7/2021	2400	2416.40991	2390	2398.689941	2398.689941	1163600
5/10/2021	2374.889893	2378	2334.72998	2341.659912	2341.659912	1300300
5/11/2021	2291.860107	2322	2283	2308.76001	2308.76001	1605500
5/12/2021	2261.709961	2285.37012	2230.050049	2239.080078	2239.080078	1746700
5/13/2021	2261.090088	2276.60107	2242.719971	2261.969971	2261.969971	1333500
5/14/2021	2291.830078	2321.13989	2283.320068	2316.159912	2316.159912	1331200
5/17/2021	2309.320068	2323.34009	2295	2321.409912	2321.409912	992100
5/18/2021	2336.906006	2343.1499	2303.159912	2303.429932	2303.429932	865100
5/19/2021	2264.399902	2316.76001	2263.52002	2308.709961	2308.709961	967500
5/20/2021	2328.040039	2360.34009	2321.090088	2356.090088	2356.090088	1191600
5/21/2021	2365.98999	2369	2342.370117	2345.100098	2345.100098	1141600
5/24/2021	2367	2418.47998	2360.110107	2406.669922	2406.669922	1062200
5/25/2021	2420	2432.88989	2402.98999	2409.070068	2409.070068	941900
5/26/2021	2412.834961	2442.94409	2412.514893	2433.530029	2433.530029	1092800

Import Libraries

1. Construct an LSTM Autoencoder on the stock price data, assuming there are no anomalies.
2. Generate the error threshold on training dataset.
3. Detect Anomaly using the threshold on test dataset

Import Libraries

Kera's:

High-level neural networks library

TensorFlow:

High performance numerical computation based library for ML applications

Numpy:

Fundamental scientific computations

Pandas:

Data preparation

Matplotlib:

Data visualization

```
from tensorflow import keras
from sklearn.preprocessing import StandardScaler
import numpy as np
import tensorflow as tf
import pandas as pd
import plotly.graph_objects as go
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, LSTM, Dropout, RepeatVector, TimeDistributed
```

1 Load the Data

The time period used is ('2004-08-19', '2020-12-24').

```
df = pd.read_csv('GOOG.csv')
```

```
df.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2004-08-19	49.813286	51.835709	47.800831	49.982655	49.982655	44871300
1	2004-08-20	50.316402	54.336334	50.062355	53.952770	53.952770	22942800
2	2004-08-23	55.168217	56.528118	54.321388	54.495735	54.495735	18342800
3	2004-08-24	55.412300	55.591629	51.591621	52.239193	52.239193	15319700
4	2004-08-25	52.284027	53.798351	51.746044	52.802086	52.802086	9232100

2 The project deals with the closing price for each day.

```
# Extract "Date" and "Close" feature colums from the dataframe.
df = df[['Date', 'Close']]
```

```
# Concise summary of a DataFrame
df.info()
```

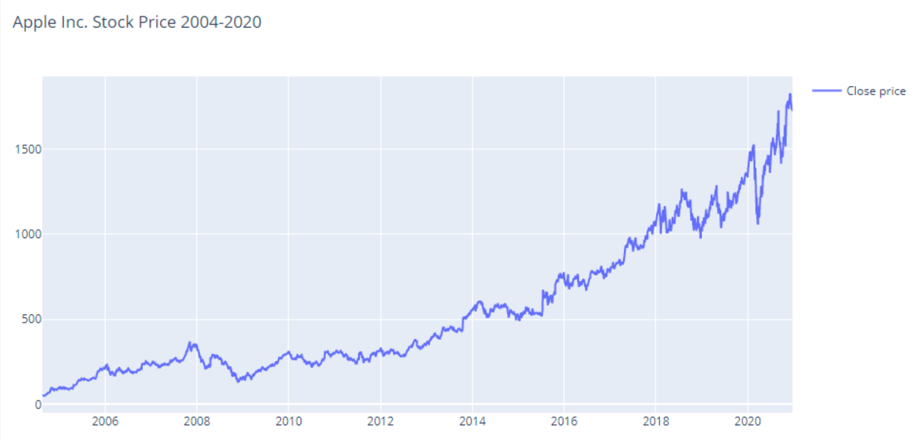
3 Data Time Period

```
df['Date'].min(), df['Date'].max()
```

```
('2004-08-19', '2020-12-24')
```

4 Visualize the data

```
fig = go.Figure()
fig.add_trace(go.Scatter(x=df['Date'], y=df['Close'], name='Close price'))
fig.update_layout(showlegend=True, title='Apple Inc. Stock Price 2004-2020')
fig.show()
```



Data Pre-processing

1 Train - test split

```
train = df.loc[df['Date'] <= '2017-12-24']
test = df.loc[df['Date'] > '2017-12-24']
train.shape, test.shape
```

```
((3362, 2), (756, 2))
```

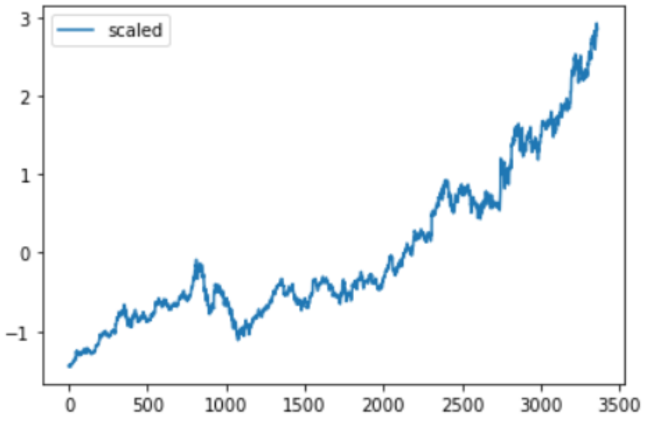
2 Data Scaling

- StandardScaler- Standardize features by removing the mean and scaling to unit variance

```
scaler = StandardScaler()
scaler = scaler.fit(np.array(train['Close']).reshape(-1,1))

train['Close'] = scaler.transform(np.array(train['Close']).reshape(-1,1))
test['Close'] = scaler.transform(np.array(test['Close']).reshape(-1,1))
```

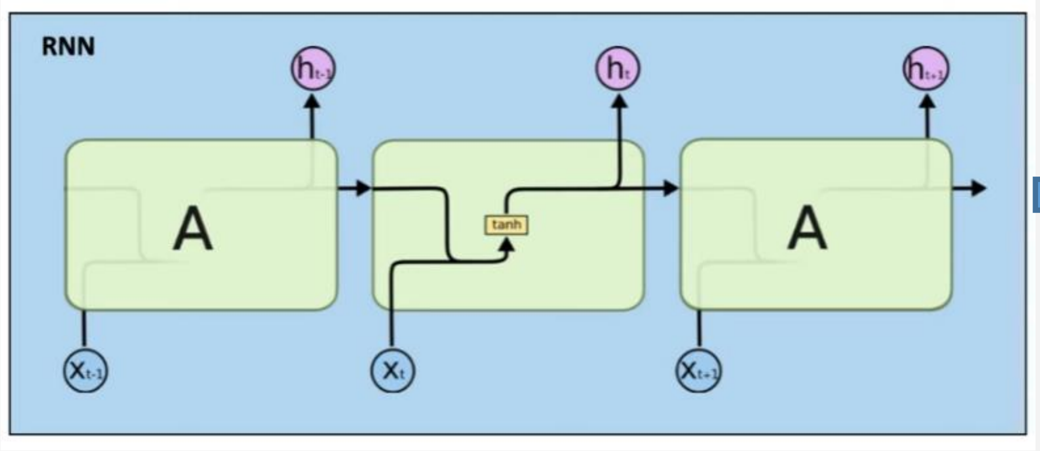
3 Visualize scaled data

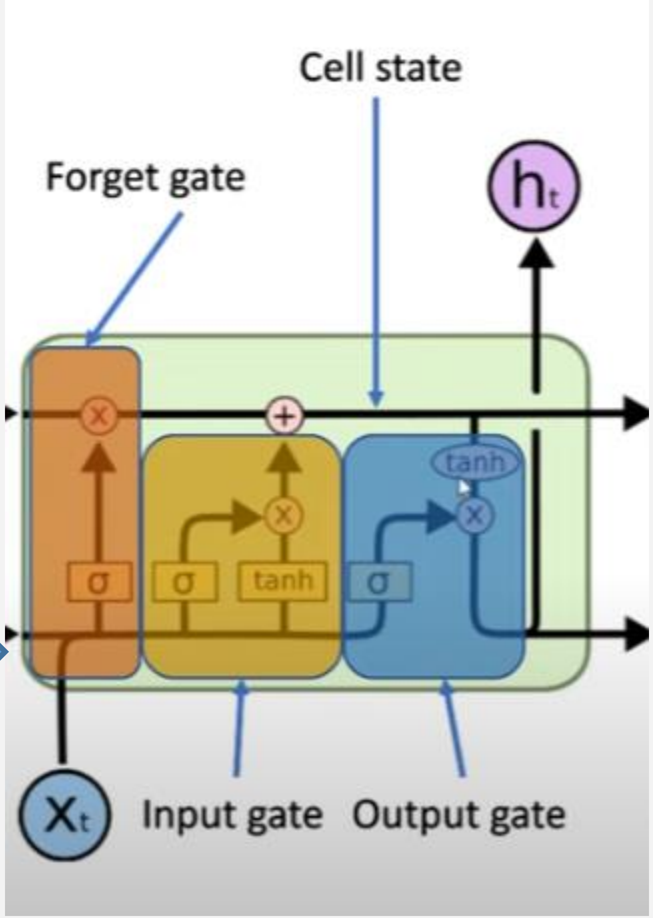
```
# Visualize scaled data
plt.plot(train['Close'], label = 'scaled')
plt.legend()
plt.show()
```



Long Short Term Memory (LSTM)

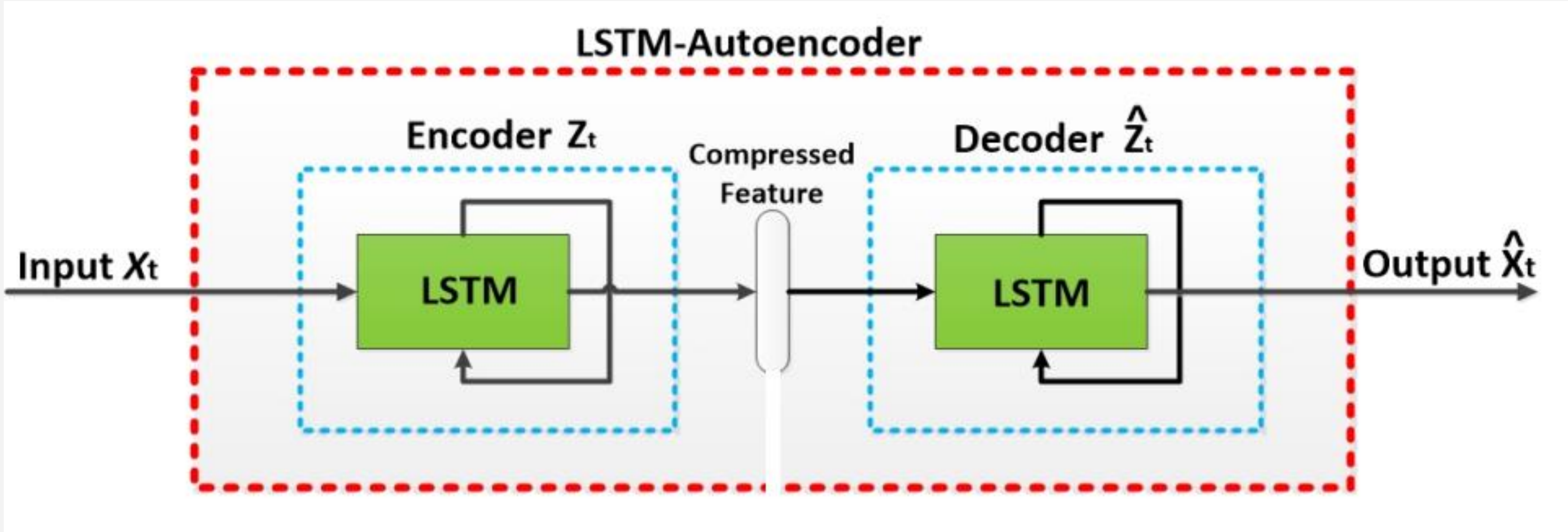
- It is very difficult to train RNNs to retain information over many time steps
- LSTM networks, add additional gating units in each memory cell.
- Forget gate
- Input gate
- Output gate
- Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.





Autoencoder LSTM

- An autoencoder is an artificial neural network that applies backpropagation, to produce the output vector similar to the inputs.
- It compresses the input data into a lower-dimensional space, then reconstructs the original data again from this representation.
- The Autoencoder is considered an unsupervised learning technique since it does not require a separate label value to train.
- In practice, the autoencoder is composed of two phases:
- Encoder: reduces the dimensions of the input data X
- Decoder: trained to obtain the output data similar to the original space,



Create Sequences

- Create sequences combining TIME_STEPS contiguous data values from the training data.
- Number of timestamps to look back
- TIME_STEP is set 30 as we want our network to have memory of 30 days.

```
TIME_STEPS=30

def create_sequences(X, y, time_steps=TIME_STEPS):
    X_out, y_out = [], []
    for i in range(len(X)-time_steps):
        X_out.append(X.iloc[i:(i+time_steps)].values)
        y_out.append(y.iloc[i+time_steps])

    return np.array(X_out), np.array(y_out)

X_train, y_train = create_sequences(train[['Close']], train['Close'])
X_test, y_test = create_sequences(test[['Close']], test['Close'])
print("Training input shape: ", X_train.shape)
print("Testing input shape: ", X_test.shape)
```

```
Training input shape:  (3332, 30, 1)
Testing input shape:  (726, 30, 1)
```

Build a Model

- The model will take input of shape **(batch_size, sequence_length, num_features)** and return output of the same shape.
- In this case, **sequence_length** is 30 and **num_features** is 1.
- Dropout is used for regularization

```
model = Sequential()
model.add(LSTM(128, activation = 'tanh', input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(Dropout(rate=0.2))
model.add(RepeatVector(X_train.shape[1]))
model.add(LSTM(128, activation = 'tanh', return_sequences=True))
model.add(Dropout(rate=0.2))
model.add(TimeDistributed(Dense(X_train.shape[2])))
model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.001), loss="mse")
model.summary()
```

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128)               66560
_________________________________________________________________
dropout (Dropout)            (None, 128)               0
_________________________________________________________________
repeat_vector (RepeatVector) (None, 30, 128)           0
_________________________________________________________________
lstm_1 (LSTM)                (None, 30, 128)           131584
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 128)           0
_________________________________________________________________
time_distributed (TimeDistri (None, 30, 1)             129
=================================================================
Total params: 198,273
Trainable params: 198,273
Non-trainable params: 0
_________________________________________________________________
```

Train Model

EPOCHS:
- Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

BATCH_SIZE:
- Since one epoch is too big to feed to the computer at once we divide it in several smaller batches.

Callback:
- An object that performs actions at various stages of training

Monitor:
- Quantity to be monitored.

Patience:
- Number of epochs with no improvement after which training will be stopped.

Mode:
- In min mode, training will stop when the quantity monitored has stopped decreasing

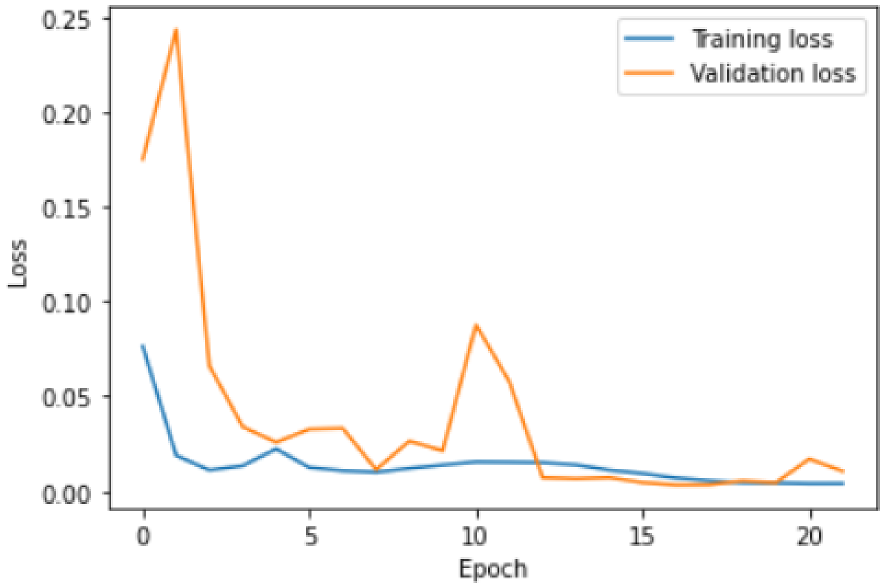
```
history = model.fit(X_train,
                    y_train,
                    epochs=100,
                    batch_size=32,
                    validation_split=0.1,
                    callbacks=[keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, mode='min')],
                    shuffle=False)
```

```
Epoch 1/100
94/94 [==============================] - 7s 79ms/step - loss: 0.0765 - val_loss: 0.1750
Epoch 2/100
94/94 [==============================] - 6s 66ms/step - loss: 0.0189 - val_loss: 0.2433
Epoch 3/100
94/94 [==============================] - 9s 101ms/step - loss: 0.0112 - val_loss: 0.0660
Epoch 4/100
94/94 [==============================] - 10s 103ms/step - loss: 0.0135 - val_loss: 0.0340
Epoch 5/100
94/94 [==============================] - 7s 74ms/step - loss: 0.0224 - val_loss: 0.0258
Epoch 6/100
94/94 [==============================] - 7s 74ms/step - loss: 0.0126 - val_loss: 0.0328
Epoch 7/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0109 - val_loss: 0.0332
Epoch 8/100
94/94 [==============================] - 6s 67ms/step - loss: 0.0101 - val_loss: 0.0115
Epoch 9/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0121 - val_loss: 0.0264
Epoch 10/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0139 - val_loss: 0.0214
Epoch 11/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0155 - val_loss: 0.0877
Epoch 12/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0154 - val_loss: 0.0579
Epoch 13/100
94/94 [==============================] - 7s 70ms/step - loss: 0.0152 - val_loss: 0.0072
Epoch 14/100
94/94 [==============================] - 7s 73ms/step - loss: 0.0140 - val_loss: 0.0066
Epoch 15/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0112 - val_loss: 0.0072
Epoch 16/100
94/94 [==============================] - 6s 68ms/step - loss: 0.0094 - val_loss: 0.0046
Epoch 17/100
94/94 [==============================] - 7s 70ms/step - loss: 0.0070 - val_loss: 0.0033
Epoch 18/100
94/94 [==============================] - 6s 69ms/step - loss: 0.0056 - val_loss: 0.0035
Epoch 19/100
94/94 [==============================] - 7s 70ms/step - loss: 0.0045 - val_loss: 0.0055
```

Plot Training - Validation loss

- The training loss **indicates how well the model is fitting the training data,**
- The validation loss indicates how well the model fits new data

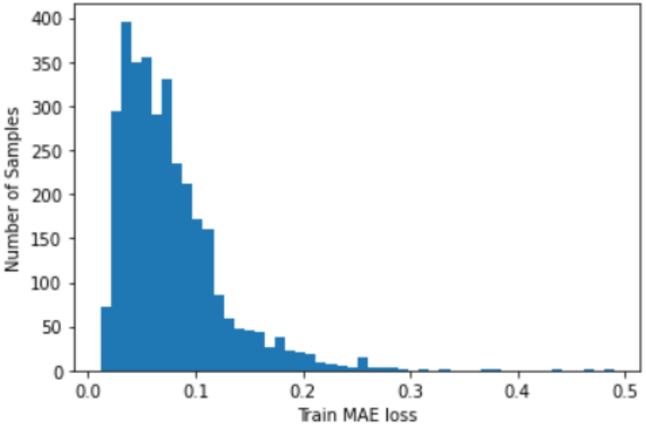
```
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend();
```



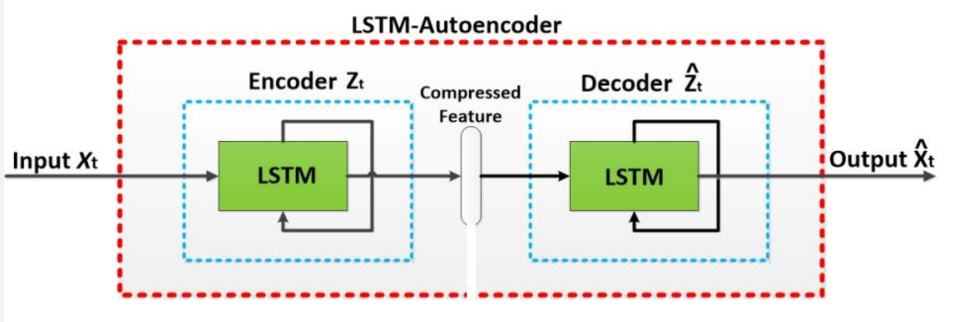
Mean Absolute Error Loss

```python
# Mean Absolute Error loss
X_train_pred = model.predict(X_train)
train_mae_loss = np.mean(np.abs(X_train_pred - X_train), axis=1)

plt.hist(train_mae_loss, bins=50)
plt.xlabel('Train MAE loss')
plt.ylabel('Number of Samples');

# Set reconstruction error threshold
threshold = np.max(train_mae_loss)

print('Reconstruction error threshold:',threshold)
```

```
Reconstruction error threshold: 0.48992275269204094
```



Reconstruction error threshold: 0.48992275269204094



Reconstruction Error

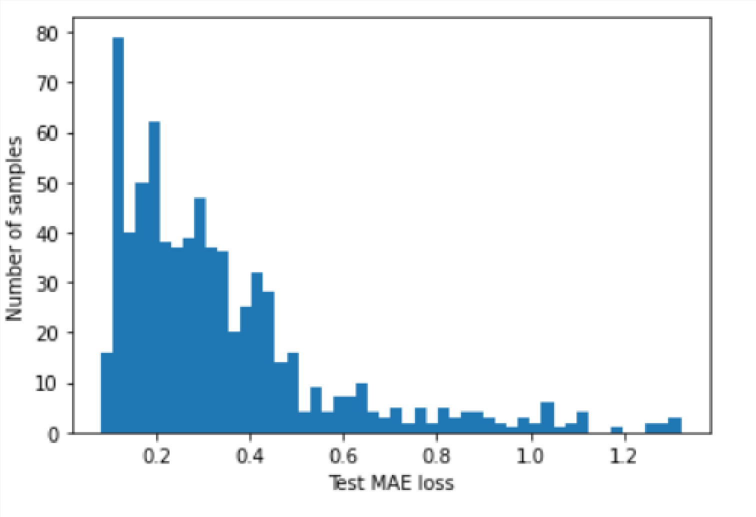
- The main goal of the autoencoder is to make the output vector similar to the original space, by minimizing the reconstruction error between them
- The reconstruction error can be obtained using a cross-entropy function or sum of squared errors (SSE)

$$SSE = \sum_{i=1}^{n} (X'_i - X_i)^2$$

Predict Anomalies on test data using threshold

```
X_test_pred = model.predict(X_test, verbose=1)
test_mae_loss = np.mean(np.abs(X_test_pred-X_test), axis=1)

plt.hist(test_mae_loss, bins=50)
plt.xlabel('Test MAE loss')
plt.ylabel('Number of samples')
```

```
23/23 [==============================] - 1s 25ms/step
Text(0, 0.5, 'Number of samples')
```



Results

```python
anomaly_df = pd.DataFrame(test[TIME_STEPS:])
anomaly_df['loss'] = test_mae_loss
anomaly_df['threshold'] = threshold
anomaly_df['anomaly'] = anomaly_df['loss'] > anomaly_df['threshold']
```

```python
anomaly_df.head()
```

	Date	Close	loss	threshold	anomaly
3391	2018-02-08	2.600211	0.298949	0.469175	False
3392	2018-02-09	2.754443	0.390377	0.469175	False
3393	2018-02-12	2.814672	0.407962	0.469175	False
3394	2018-02-13	2.815352	0.400946	0.469175	False
3395	2018-02-14	2.890214	0.394943	0.469175	False

Test loss vs. Threshold

Results

```
anomalies = anomaly_df.loc[anomaly_df['anomaly'] == True]
anomalies.head()
```

	Date	Close	loss	threshold	anomaly
3708	2019-05-15	3.292211	0.471546	0.469175	True
3721	2019-06-04	2.819393	0.510124	0.469175	True
3722	2019-06-05	2.773328	0.527452	0.469175	True
3723	2019-06-06	2.782345	0.538017	0.469175	True
3724	2019-06-07	2.874646	0.532241	0.469175	True


```
fig = go.Figure()
fig.add_trace(go.Scatter(x=anomaly_df['Date'], y=scaler.inverse_transform(anomaly_df['Close']), name='Close price'))
fig.add_trace(go.Scatter(x=anomalies['Date'], y=scaler.inverse_transform(anomalies['Close']), mode='markers', name='Anomaly'))
fig.update_layout(showlegend=True, title='Detected anomalies')
fig.show()
```

Thanks!