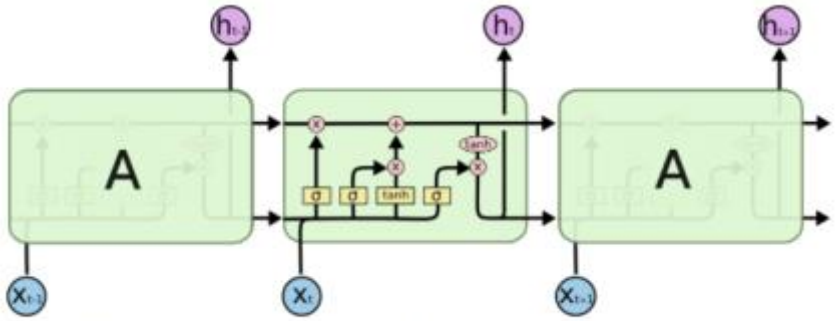
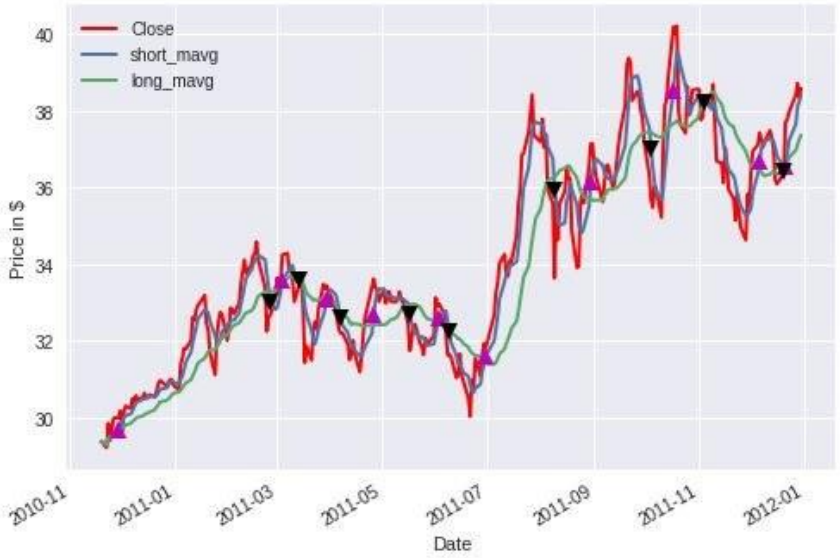
Stock Prediction and Algorithmic Trading

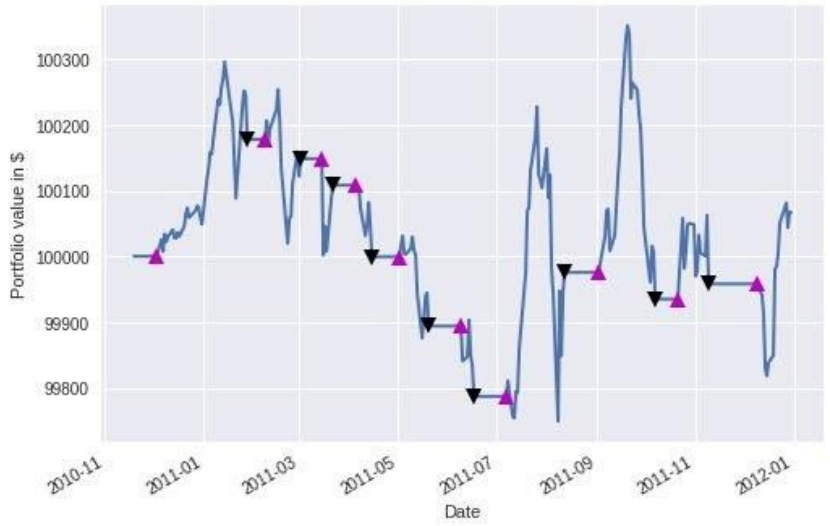
The Problem(s)

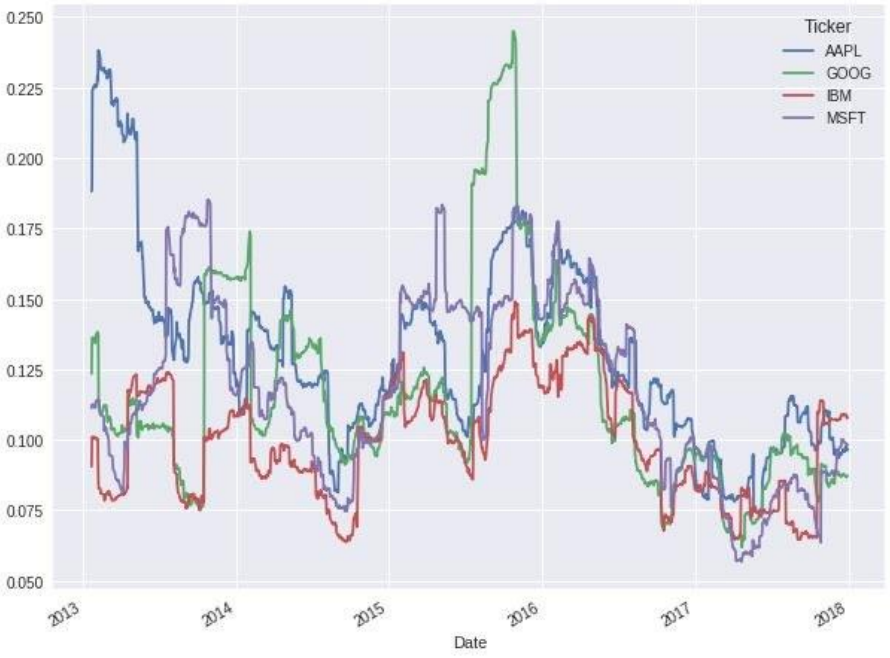
- *Financial Time Series data preparation*
- *Sequence Prediction -> LSTM*
- *Optimization*
- *Algorithmic Trading*
- *Backtesting*
- *Plots*
- *Results and Analysis*



The repeating module in an LSTM contains four interacting layers.




100300
100200
100100
100000
99900
99800
Portfolio value in $
2010-11
2011-01
2011-03
2011-05
2011-07
2011-09
2011-11
2012-01
Date


0.250
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
Ticker
AAPL
GOOG
IBM
MSFT
2013
2014
2015
2016
2017
2018
Date

Tools and Techniques

Environment


K

Data Retrieval APIs

We've used the pandas-datareader as our data-handler providing an interface for data used in the project. The pandas-datareader makes use of these sources or fetching Stock Prices and other related data

1 - Kaggle dataset

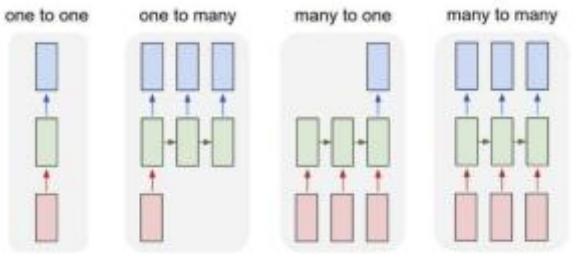
Model

Recurrent Neural Network

Recurrent Neural Networks are the state of the art algorithm for sequential data and among others used by Apples Siri and Googles Voice Search. It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for Machine Learning problems that involve sequential data.

Because of their internal memory, RNN's are able to remember important things about the input they received, which enables them to be very precise in predicting what's coming next making them the preferred algorithm for sequential data like time series, speech, text, audio, video, weather and today's emphasis financial data.

" RNNs are usedWhenever there is a sequence of data and that temporal dynamics that connects the data is more important than the spatial content of each individual frame." – Lex Fridman (MIT)

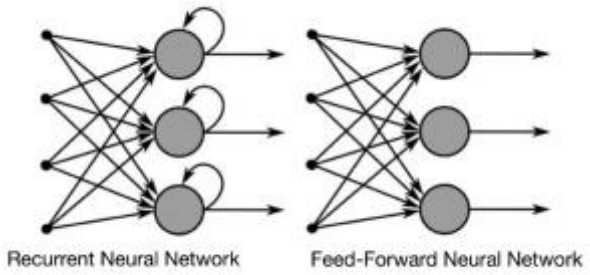


RNN's and Feed-Forward Neural Networks are both named after the way they channel information.

In a Feed-Forward neural network, the information only moves in one direction, from the input layer, through the hidden layers, to the output layer. Because a feed-forward network only considers the current input, it has no notion of order in time.

In a RNN, the information cycles through a loop. When it makes a decision, it takes into consideration the current input and also what it has learned from the inputs it received previously.

Therefore a Recurrent Neural Network has two inputs, the present and the recent past. This is important because the sequence of data contains crucial information about what is coming next, which is why a RNN can do things other algorithms can't.



Issues of standard RNN's

Exploding Gradients

We speak of „Exploding Gradients" when the algorithm assigns a stupidly high importance to the weights, without much reason. But fortunately, this problem can be easily solved if you truncate or squash the gradients.

Vanishing Gradients

We speak of „Vanishing Gradients" when the values of a gradient are too small and the model stops learning or takes way too long because of that. This was a major problem in the 1990s and much harder to solve than the exploding gradients. Fortunately, it was solved through the concept of LSTM by Sepp Hochreiter and Juergen Schmidhuber.

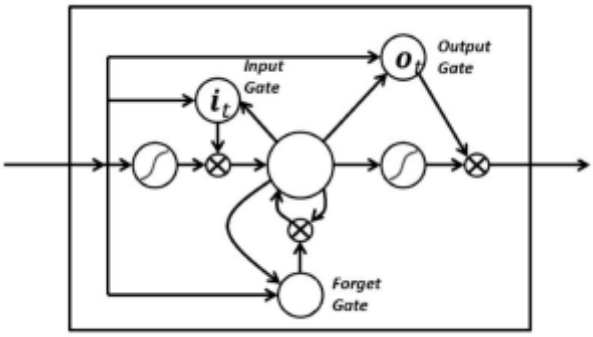
Long Short Term Memory

Long Short-Term Memory (LSTM) networks are an extension for recurrent neural networks, which basically extends their memory. Therefore it is well suited to learn from important experiences that have very long time lags in between.

This memory can be seen as a gated cell, where the cell decides whether or not to store or delete information based on the importance it assigns to the information. The assigning of importance happens through weights, which are also learned by the algorithm.

In an LSTM you have three gates: input, forget and output gate. These gates determine whether or not to let new input in (input gate), delete the information because it isn't important (forget gate) or to let it impact the output at the current time step (output gate).

The problematic issues of vanishing gradients is solved through LSTM because it keeps the gradients steep enough and therefore the training relatively short and the accuracy high.



Trading Strategies

A trading strategy is the method of buying and selling in markets that is based on predefined rules used to make trading decisions.

Technical trading strategies rely on technical indicators to generate trading signals. Technical traders believe all information about a given security is contained in its price and it moves in trends.

For example, a simple trading strategy may be a moving average crossover whereby a short-term moving average crosses above or below a long-term moving average.

Trading strategies are employed to avoid behavioral finance biases and ensure consistent results.

Profitable trading strategies are difficult to develop, however, and there is a risk of becoming over-reliant on a strategy.

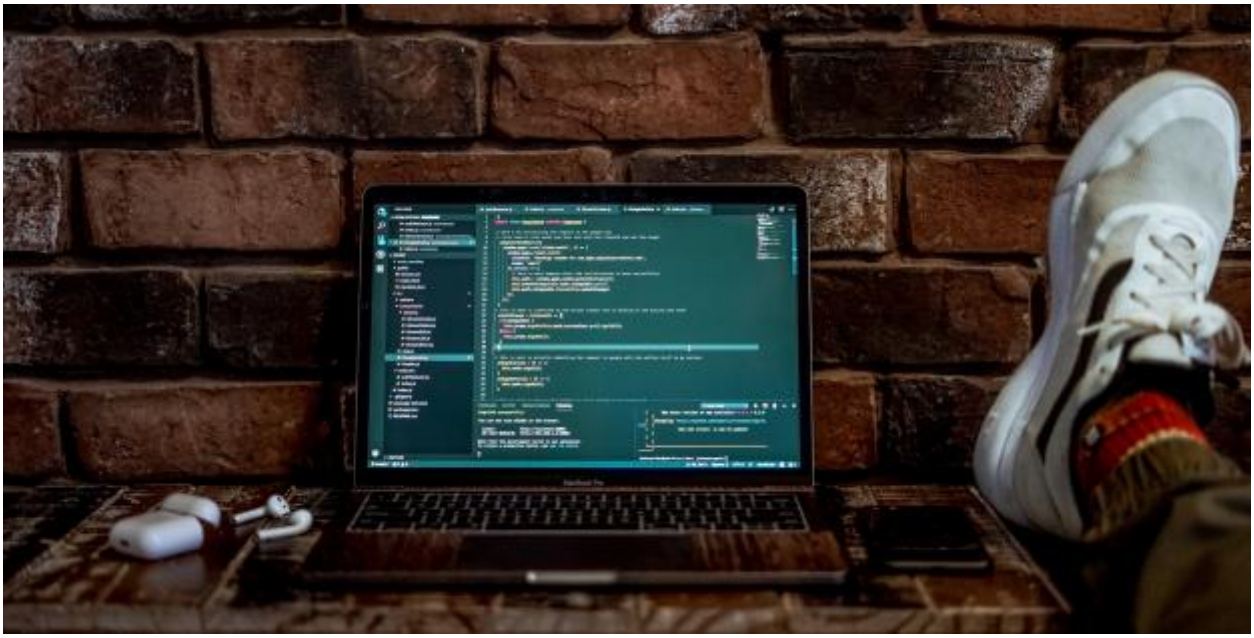
In general, there are two common trading strategies: the momentum strategy and the reversion strategy.

Firstly, the **momentum strategy** is also called divergence or trend trading. When you follow this strategy, you do so because you believe the movement of a quantity will continue in its current direction. Stated differently, you believe that stocks have momentum or upward or downward trends, that you can detect and exploit.

Some examples of this strategy are the moving average crossover, the dual moving average crossover, and turtle trading

Secondly, the **reversion strategy**, which is also known as convergence or cycle trading. This strategy departs from the belief that the movement of a quantity will eventually reverse. This might seem a little bit abstract, but will not be so anymore when you take the example. Take a look at the mean reversion strategy, where you actually believe that stocks return to their mean and that you can exploit when it deviates from that mean.

Implementations



Algorithmic Trading

We've implemented the moving crossover strategy in this project which generates signals based on two lookback periods:

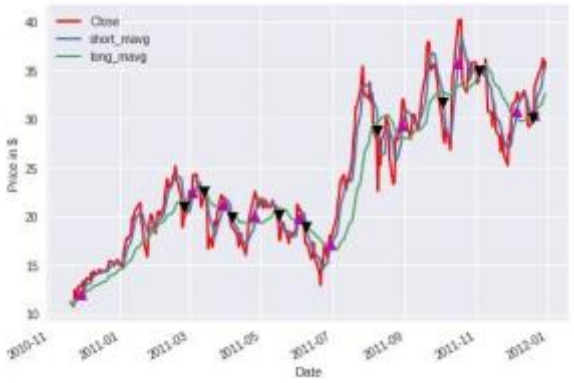
Short Window (5)

Long Window (20)

>Rolling averages are then calculated on both windows

>Signals are generated (0 : short, 1 :long)

([github])



	signal	short_mavg	long_mavg	positions
Date				
2010-11-19	0.0	11.184861	11.184861	NaN
2010-11-22	0.0	10.967988	10.967988	0.0
2010-11-23	0.0	11.459546	11.459546	0.0
2010-11-24	0.0	11.410005	11.410005	0.0
2010-11-26	0.0	11.690015	11.690015	0.0
2010-11-29	1.0	12.025260	11.885193	1.0
2010-11-30	1.0	12.542674	12.092764	0.0
2010-12-01	1.0	12.429903	12.066019	0.0
2010-12-02	1.0	12.821277	12.194045	0.0
2010-12-03	1.0	12.992216	12.341116	0.0
2010-12-06	1.0	13.116717	12.444977	0.0
2010-12-07	1.0	13.284080	12.589145	0.0

Backtesting

For backtesting, we generate a portfolio to manage orders and Profits (or losses).

>Initial capital = 1,00,000

>Positions = 100*signal : buy a hundred shares

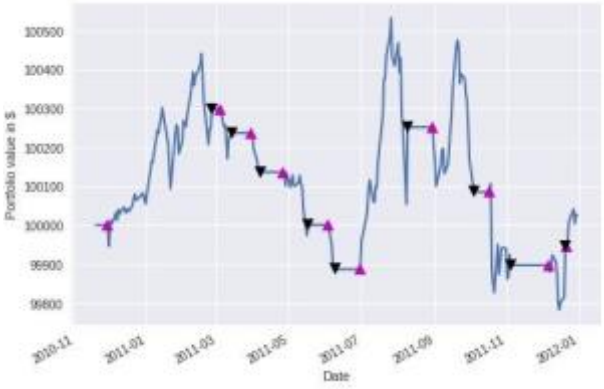
>Holdings

>Cash

>Total = portfolio['cash'] + portfolio['holdings']

>Returns = portfolio['total'].pct_change()

	AAPL	holdings	cash	total	returns
Date					
2010-11-19	0.000000	0.000000	100000.000000	100000.000000	NaN
2010-11-22	0.000000	0.000000	100000.000000	100000.000000	0.000000
2010-11-23	0.000000	0.000000	100000.000000	100000.000000	0.000000
2010-11-24	0.000000	0.000000	100000.000000	100000.000000	0.000000
2010-11-26	0.000000	0.000000	100000.000000	100000.000000	0.000000
2010-11-29	3018.427086	3018.427086	96981.572914	100000.000000	0.000000
2010-11-30	2963.939095	2963.939095	96981.572914	99945.512009	-0.000545
2010-12-01	3013.949203	3013.949203	96981.572914	99995.522118	0.000500
2010-12-02	3030.619612	3030.619612	96981.572914	100012.192726	0.000167
2010-12-03	3023.855782	3023.855782	96981.572914	100005.428696	-0.000068



Backtesting Pitfalls

With backtesting, a trader can simulate and analyze the risk and profitability of trading with a specific strategy over a period of time. However, when backtesting, it's a good idea to keep in mind that there are some pitfalls, which might not be obvious to the eye.

For example, there are external events, such as market regime shifts, which are regulatory changes or macroeconomic events, which definitely influence your backtesting. Also, liquidity constraints, such as the ban of short sales, could affect your backtesting heavily.

Next, there are pitfalls which you might introduce yourself when you, for example, overfit a model (optimization bias), when you ignore strategy rules because you think it's better like that (interference), or when you accidentally introduce information into past data (lookahead bias).

Sharpe ratio

To get to know whether your portfolio's returns are the result of the fact that you decided to make smart investments or to take a lot of risks.

The ideal situation is, of course, that the returns are considerable but that the additional risk of investing is as small as possible.

That's why, the greater the portfolio's Sharpe ratio, the better: the ratio between the returns and the additional risk that is incurred is quite OK.

Usually, a ratio greater than 1 is acceptable by investors, 2 is very good and 3 is excellent.

```
# Isolate the returns of your strategy
returns = portfolio['returns']

# annualized Sharpe ratio
sharpe_ratio = np.sqrt(252) * (returns.mean() / returns.std())

# Print the Sharpe ratio
print(sharpe_ratio)
```

```
0.03901387936157988
```

Kaggle NYSE Dataset

```
df = pd.read_csv("../input/prices-split-adjusted.csv", index_col = 0)
df["adj close"] = df.close # Moving close to the last column
df.drop(['close'], 1, inplace=True) # Moving close to the last column
df.head()
```

	symbol	open	low	high	volume	adj close
date						
2016-01-05	WLTW	123.430000	122.309998	126.250000	2163600.0	125.839996
2016-01-06	WLTW	125.239998	119.940002	125.540001	2386400.0	119.980003
2016-01-07	WLTW	116.379997	114.930000	119.739998	2489500.0	114.949997
2016-01-08	WLTW	115.480003	113.500000	117.440002	2006300.0	116.620003
2016-01-11	WLTW	117.010002	114.089996	117.330002	1408600.0	114.970001

- **prices.csv:** original, unaltered daily prices. Most of data ranges from 2010 to the end 2016 except for companies added late to the stock market index in which case the time frame is shorter. The data doesn't account for stock splits which count to about 140 in the said time.
- **prices-split-adjusted.csv:** same as prices with added adjustment for stock splits .
- **securities.csv:** general information about each company with division on its sectors
- **fundamentals.csv:** metrics extracted from annual SEC 10K filings (2012-2016). These an be used to derive the popular financial indicators that are needed.

```
symbols[:11] # Example of what is in symbols
```

```
['AMZN',
 'DFS',
 'WFM',
 'DLR',
 'KHC',
 'AMT',
 'NDAQ',
 'BSX',
 'GOOGL',
 'FITB',
 'AEP']
```

```
df = df[df.symbol == 'GOOG']
df.drop(['symbol'],1,inplace=True)
df.head()
```

	open	low	high	volume	adj close
date					
2010-01-04	312.304948	310.955001	313.580158	3927000.0	312.205308
2010-01-05	312.419511	309.610028	312.748278	6031900.0	310.830459
2010-01-06	311.761979	302.048370	311.761979	7987100.0	302.994813
2010-01-07	303.562685	295.218951	303.861575	12876600.0	295.941242
2010-01-08	294.895159	293.455551	300.499172	9483900.0	299.886470

```
def normalize_data(df):
    min_max_scaler = preprocessing.MinMaxScaler()
    df['open'] = min_max_scaler.fit_transform(df.open.values.reshape(-1,1))
    df['high'] = min_max_scaler.fit_transform(df.high.values.reshape(-1,1))
    df['low'] = min_max_scaler.fit_transform(df.low.values.reshape(-1,1))
    df['volume'] = min_max_scaler.fit_transform(df.volume.values.reshape(-1,1))
    df['adj close'] = min_max_scaler.fit_transform(df['adj close'].values.reshape(-1,1))
    return df
df = normalize_data(df)
df.head()
```

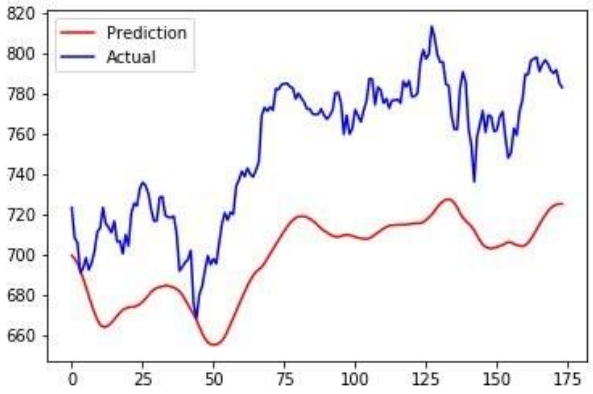
	open	low	high	volume	adj close
date					
2010-01-04	0.157047	0.161167	0.156390	0.131722	0.159399
2010-01-05	0.157238	0.158884	0.154995	0.202469	0.157092
2010-01-06	0.156140	0.146049	0.153341	0.268184	0.143942
2010-01-07	0.142436	0.134457	0.140094	0.432522	0.132105
2010-01-08	0.127950	0.131464	0.134455	0.318492	0.138726

```
def build_model(layers):
    d = 0.3
    model = Sequential()

    model.add(LSTM(256, input_shape=(layers[1], layers[0]), return_sequences=True))
    model.add(Dropout(d))

    model.add(LSTM(256, input_shape=(layers[1], layers[0]), return_sequences=False))
    model.add(Dropout(d))

    model.add(Dense(32,kernel_initializer="uniform",activation='relu'))
    model.add(Dense(1,kernel_initializer="uniform",activation='linear'))

    # adam = keras.optimizers.Adam(decay=0.2)

    start = time.time()
    model.compile(loss='mse',optimizer='adam', metrics=['accuracy'])
    print("Compilation Time : ", time.time() - start)
    return model
```

The model scores were evaluated as :

Score	MSE	RMSE
Train	0.00109	0.03 RMSE
Test	0.00938	0.10 RMSE

Plotting the actual and predicted prices together we get:



```
snp_data.head()
```

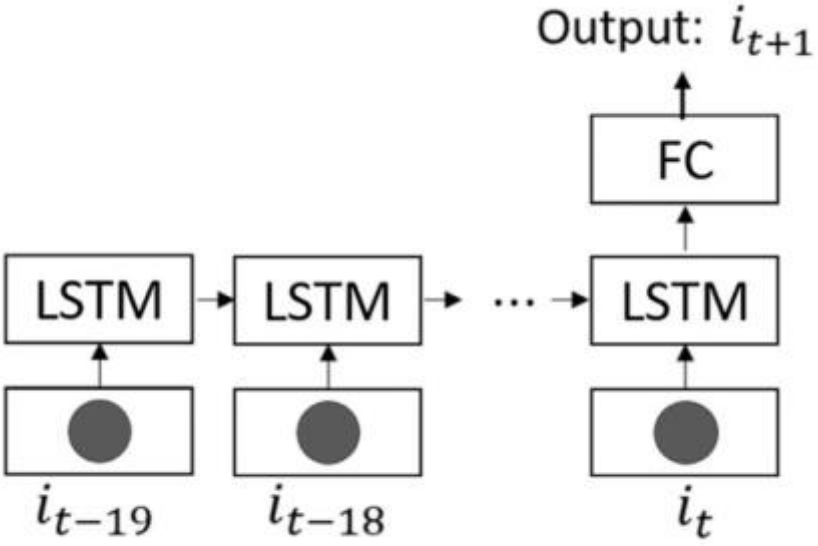
	Adj Close
Date	
2000-01-04	1399.420044
2000-01-05	1402.109985
2000-01-06	1403.449951
2000-01-07	1441.469971
2000-01-10	1457.599976

```
d=all_data[['Adj Close']].reset_index().pivot('Date', 'Ticker', 'Adj Close')
d
```

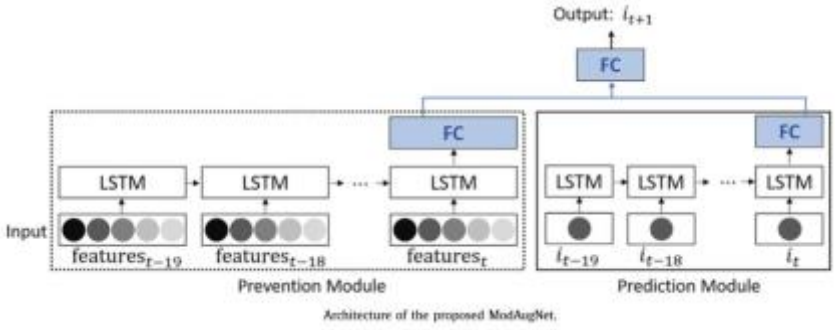
Ticker	AAPL	ADBE	AMGN	AMZN	ASML	CSCO	GILD	MSFT	NVDA
Date									
2000-01-04	2.440975	14.791295	48.689911	81.937500	26.826624	40.324173	1.513320	40.692646	3.506938
2000-01-05	2.476697	15.083735	50.365246	69.750000	26.085730	40.200638	1.501003	41.121727	3.391485
2000-01-06	2.262367	15.206868	51.202930	65.562500	24.480459	39.533516	1.603064	39.744217	3.170195
2000-01-07	2.369532	15.945663	56.961960	69.562500	25.036133	41.856102	1.847659	40.263588	3.223112
2000-01-10	2.327857	16.561329	60.417358	69.187500	27.413177	43.412739	2.088733	40.557178	3.328946
2000-01-11	2.208785	15.422350	55.653084	66.750000	26.517927	42.103184	1.897808	39.518398	3.194248
2000-01-12	2.076318	15.345394	53.244766	63.562500	27.011856	41.040722	1.881092	38.231228	3.112468
2000-01-13	2.304043	16.161148	55.967213	65.937500	28.200378	41.979649	1.984913	38.953846	3.242355

```
d.corrwith(snp_data['Adj Close'])
```

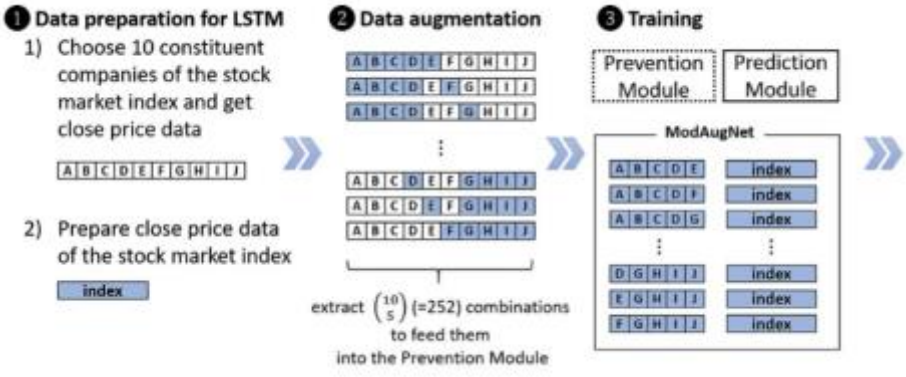
```
Ticker
ASML    0.939346
ADBE    0.931655
AMGN    0.920196
MSFT    0.920161
SBUX    0.913389
AAPL    0.899090
AMZN    0.870971
GILD    0.861092
NVDA    0.694488
CSCO    0.489332
```

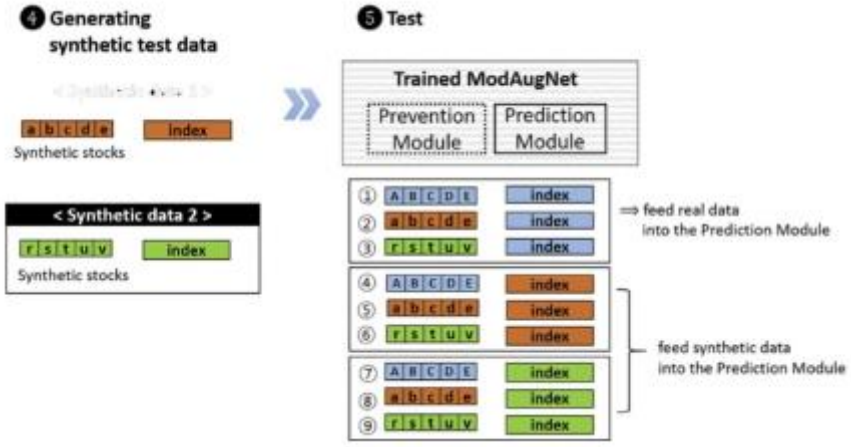


Architecture of the comparative model (called SingleNet).



Architecture of the proposed ModAugNet.





<Synthetic data 1>

Stock 1	Stock 2	Stock 3	Stock 4	Stock 5
80.2700*	19.3400*	8.6000*	7.3200*	25.0600*
95.0500**	58.5000**	80.0625**	46.3125**	107.2300**
93.9773	35.6673	42.8646	21.5123	42.5319
86.3682	40.0774	71.9137	32.7380	36.4135
90.8230	37.6242	32.6242	42.2130	50.0072
42.0761	32.7588	69.5146	8.0059	65.7760
54.4551	23.6496	47.7760	30.3630	60.7512
78.3708	22.1303	16.7528	7.4331	42.8505
45.1511	53.7986	29.9641	40.6810	87.0722
80.9823	58.3798	62.5843	25.7652	90.0763
52.9144	21.6258	13.5990	11.8126	75.0358
58.8015	32.4412	79.2704	21.6826	32.8648
80.6418	53.8477	72.9355	27.1330	66.6216
90.1018	41.7594	69.5793	7.8128	91.6469
50.9637	20.0238	32.1205	39.1383	64.9313
55.7780	47.6092	66.8461	30.8644	44.3372
74.5079	29.5871	11.4764	41.3768	57.5522
33.7314	44.9736	12.8056	26.9845	82.0032
88.5760	40.9349	15.5002	23.7163	69.8982
65.4187	53.4469	26.3636	15.0815	79.6973
83.7899	45.5478	16.6434	8.9368	39.2880
80.1720	46.2099	8.7943	46.0460	91.7732

<Synthetic data 2>

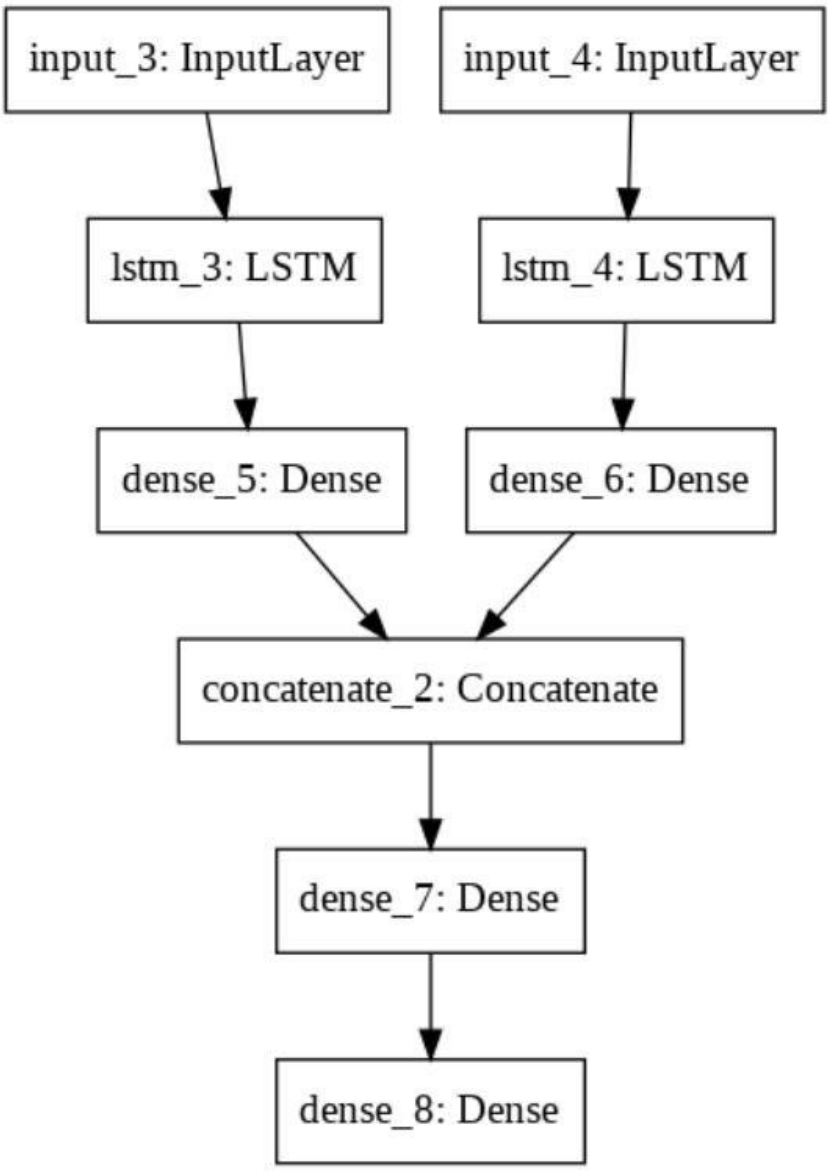
Stock 1	Stock 2	Stock 3	Stock 4	Stock 5
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10
10	10	10	10	10

Table 2
Test MSEs of ModAugNet-f.

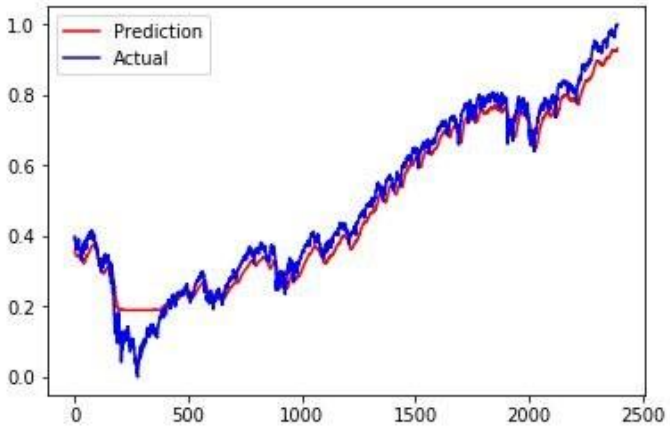
	Prevention Module	Prediction Module	Test MSE
(1)	5 stocks	S&P500	1665.31
(2)	Synthetic data 1		9910.59
(3)	Synthetic data 2		3630.42
(4)	5 stocks	Synthetic data 1	208,459.45
(5)	Synthetic data 1		2,379,50.25
(6)	Synthetic data 2		2,297,98.13
(7)	5 stocks	Synthetic data 2	818,944.20
(8)	Synthetic data 1		7,856,18.40
(9)	Synthetic data 2		9,757,42.48

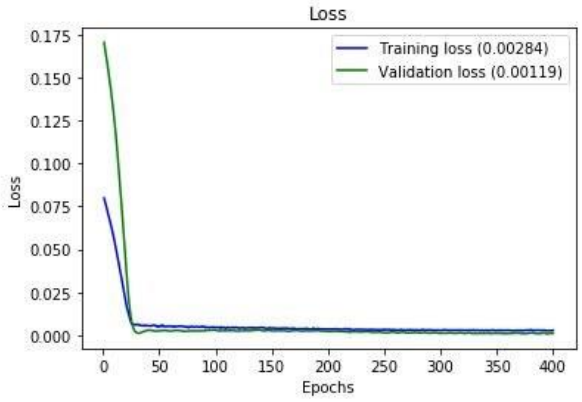
Table 3
Test MSEs of ModAugNet-c.

	Prevention Module	Prediction Module	Test MSE
(1)	5 stocks	S&P500	342.48
(2)	Synthetic data 1		
(3)	Synthetic data 2		
(4)	5 stocks	Synthetic data 1	389,239.25
(5)	Synthetic data 1		
(6)	Synthetic data 2		
(7)	5 stocks	Synthetic data 2	784,151.16
(8)	Synthetic data 1		
(9)	Synthetic data 2		


input_3: InputLayer
input_4: InputLayer
lstm_3: LSTM
lstm_4: LSTM
dense_5: Dense
dense_6: Dense
concatenate_2: Concatenate
dense_7: Dense
dense_8: Dense

```
import matplotlib.pyplot as plt2

plt2.plot(model.predict(testX),color='red', label='Prediction')
plt2.plot(testY,color='blue', label='Actual')
plt2.legend(loc='best')
plt2.show()
```






The End