# SET50 stock price prediction using LSTM

KMUTT SCIENTIFIC PROGRAMMING BOOTCAMP WITH PYTHON

# Team Members & Advisor

- Poomphob Suwannapichat (Phob)
- Narongrid Seesawad (Rid)
- Paramet Moonkaew (Met)
- Advisor: Dr. Teerasit Termsaithong

# Introduction

- Stocks
- SET50
- Forecasting model
- Deep learning
- LSTM

# Stocks

- **Part of ownership** in a particular company
- A divided-up unit of the value of company (**Shares**)
- Stock prices can **go up and down**
- Depends on **how well company doing**
- **Stock Market** - The places where to buy or sells shares of company.

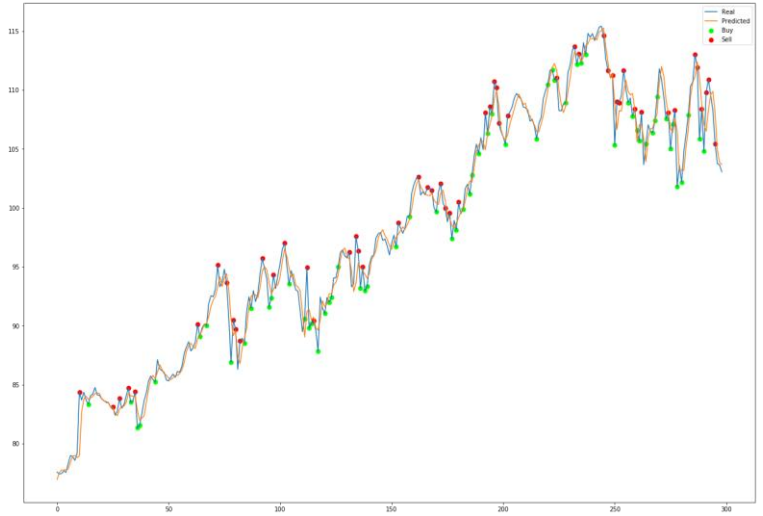

https://miro.medium.com/max/576/0*lubTctDQw1cFDxzo

# SET50

- Composite index
- Market Capitalization base
- **High liquidity**
- **High earning**
- **Top 50 companies**
- e.g., PTT, AOT, KBANK

# Forecasting model

- For the case of stock values study
  - **Long term forecasting**
  - **Financial time series**
  - **Higly non-linear data**
  - **Hidden pattern and underlying dynamics**

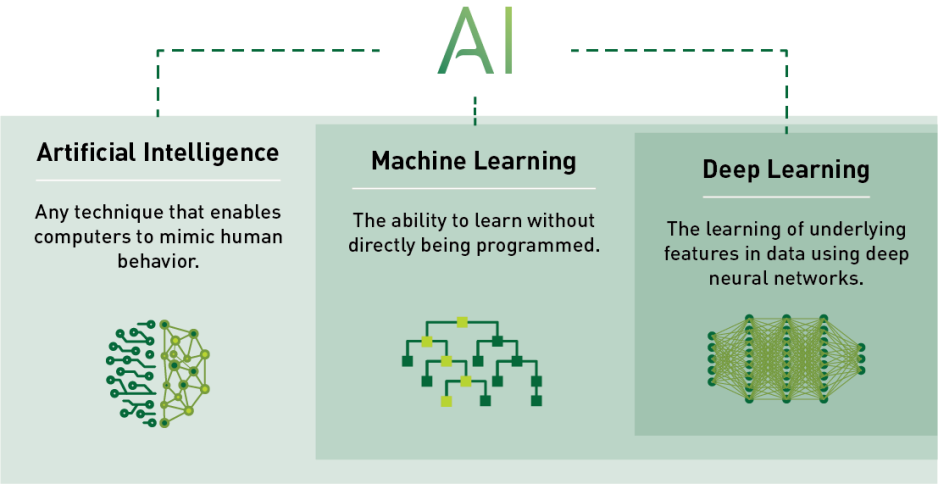


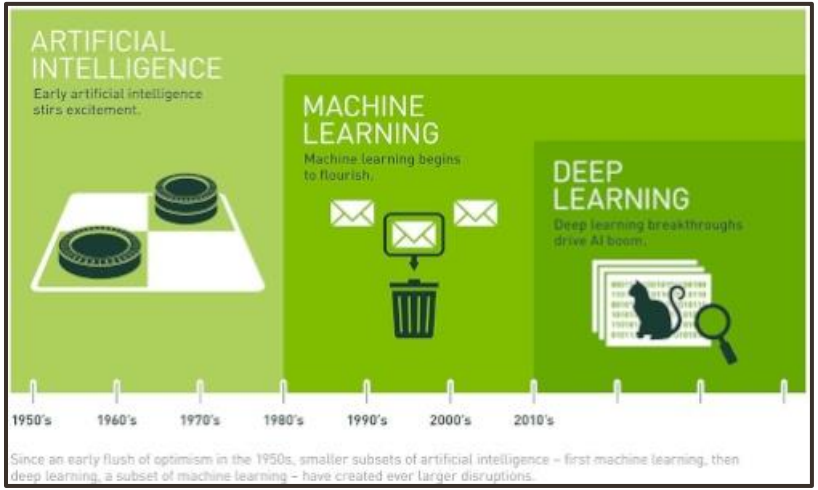


**‘DEEP LEARNING’**

https://miro.medium.com/max/2542/1*e8UBEaMt7Pw5d9gP0tLHjA.png

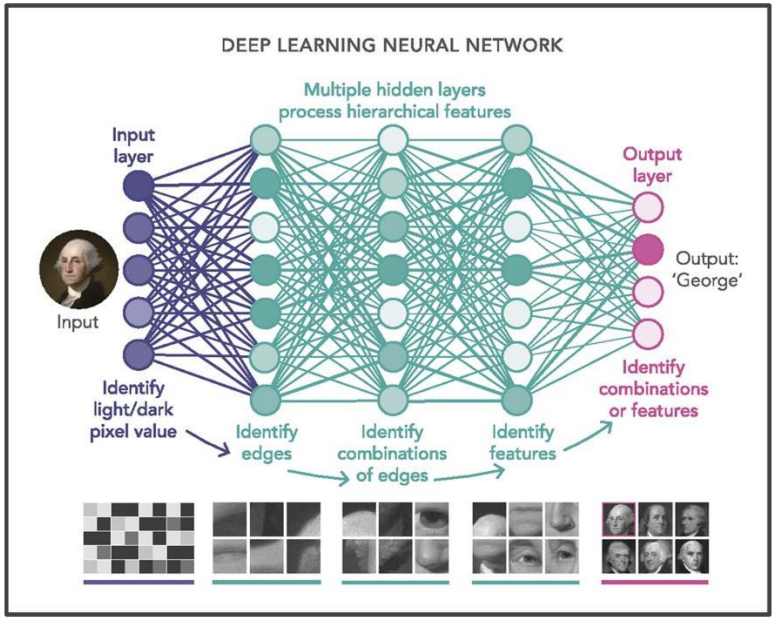
# Deep learning

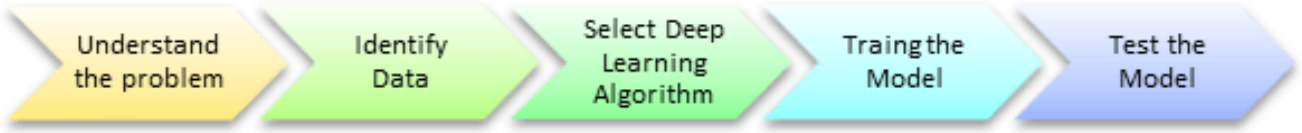



https://ahead.asia/wp-content/uploads/2018/05/AheadAsia-AI-to-DeepLearning.jpg

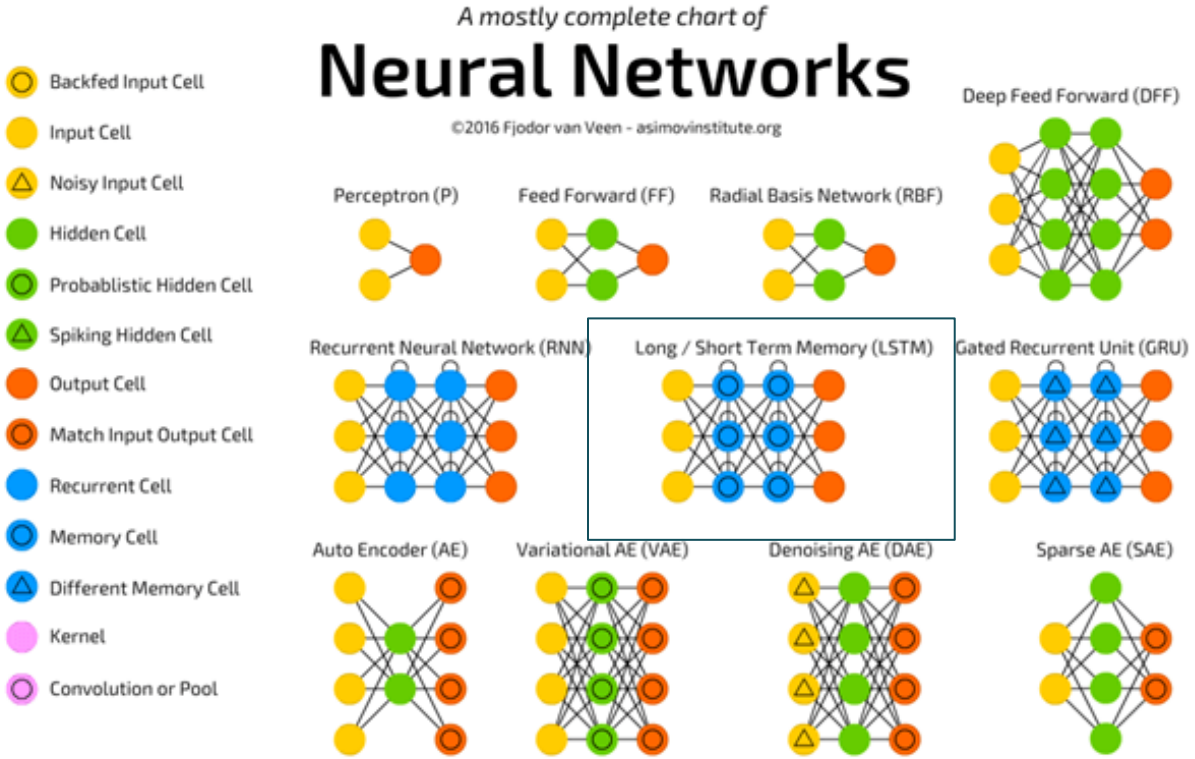
# Deep learning




https://www.pnas.org/content/pnas/116/4/1074/F2.large.jpg

# Deep learning

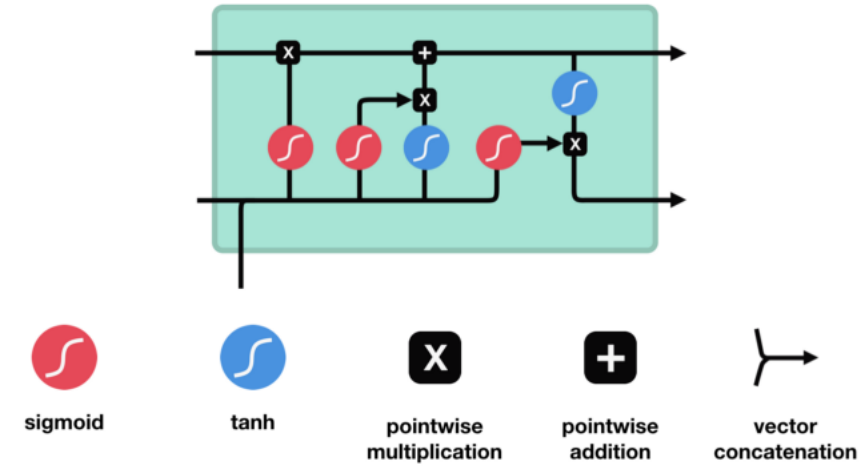



https://www.guru99.com/images/tensorflow/083018_0542_WhatisDeepl3.png

# LSTM MODEL

- **Avoiding** the long-term dependency problem
- Suitable to deal with a **time series with long term patterns.**

- **Stock price prediction** (Sreelekshmy Selvin et al., 2017)
- **Stock chart pattern recognition** (Marc Velay et al., 2018 )




**‘Keep or forget information’**

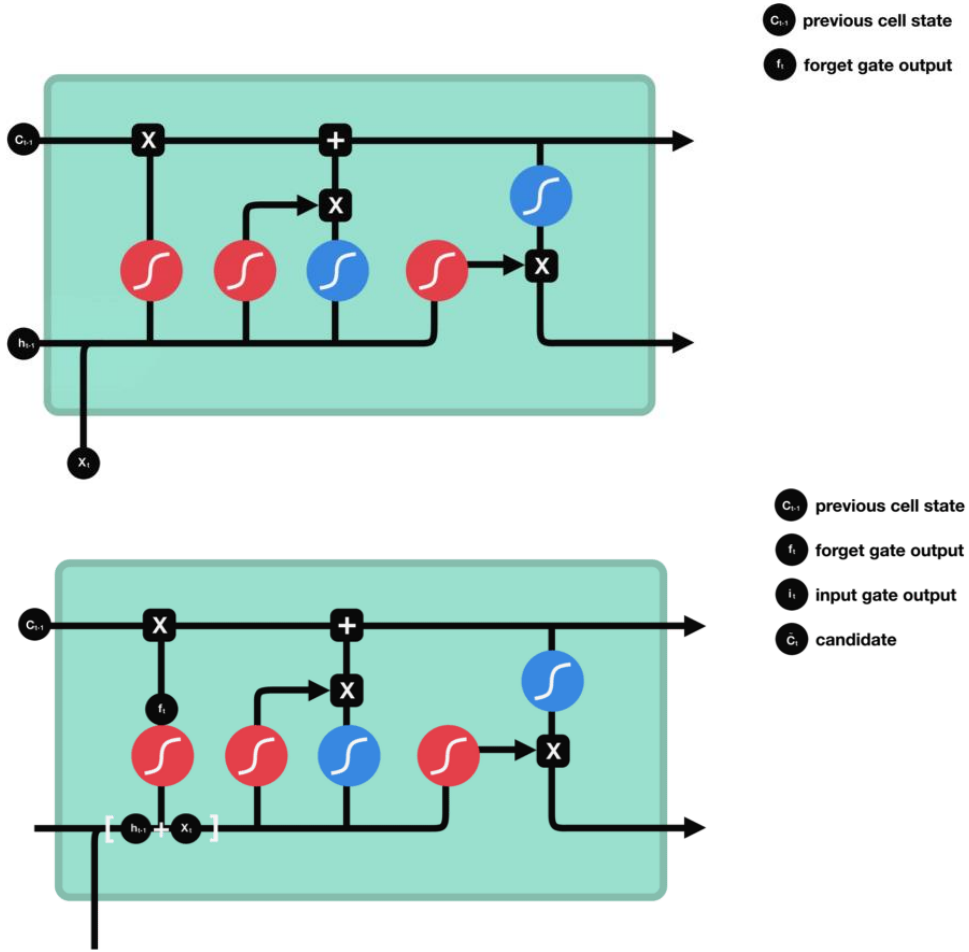
# LSTM MODEL

- **Forget gate layer**
  - What information should be thrown away or kept (Hidden state + Current input)

- **Input gate layer**

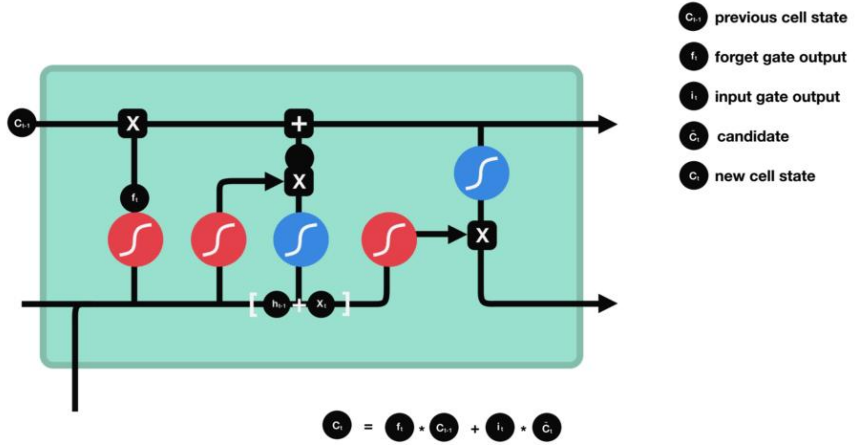
  - **Sigmoid**
    Keep or forget it, (0,1)

  - **Tanh activation**
    Regulate significant value, (-1,1)

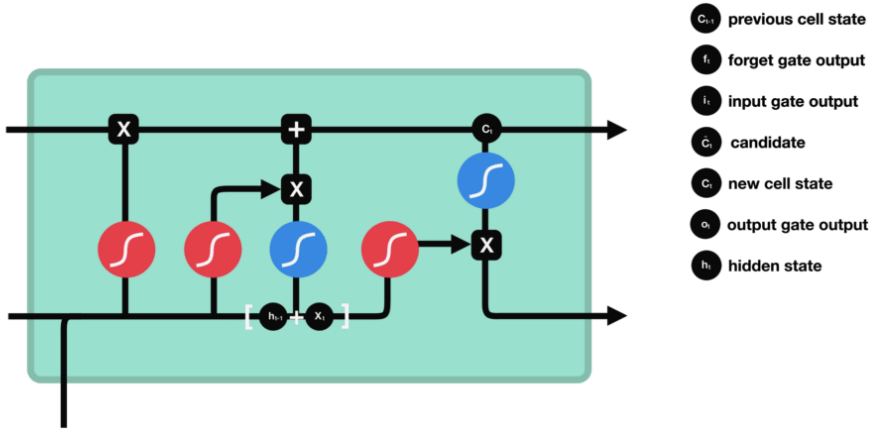


# LSTM MODEL

- **New Cell state**
  - pointwise multiplied by the forget vector
  - get addition with output from input gate




- **Output gate**
  - Decides what the next hidden state should be



# Objectives

- **To predict the stock price of SET50 using LSTM model**
- **To implement the backtesting model, for ensure the assumptions of the model are valid.**
- **To develop and build the model, which can be use with real trading**

# Methodology

- Preprocess data
- LSTM Model
- Backtesting

# Preprocess data

```
data = pd.read_excel("SET50 1 Day.xlsx",
    sheet_name=0,
    header=0,
    index_col="Date",
    keep_default_na=True
)
```

```
data
```

| Date | Open | High | Low | Close | Volume |
|---|---|---|---|---|---|
| 1996-06-17 | 969.10 | 970.80 | 966.90 | 967.10 | 0 |
| 1996-06-18 | 967.10 | 973.00 | 964.90 | 972.20 | 0 |
| 1996-06-19 | 972.40 | 975.50 | 966.30 | 966.80 | 0 |
| 1996-06-20 | 966.80 | 968.80 | 956.40 | 956.60 | 0 |
| 1996-06-21 | 956.40 | 959.00 | 953.70 | 956.30 | 0 |
| ... | ... | ... | ... | ... | ... |
| 2019-01-18 | 1058.03 | 1063.75 | 1055.77 | 1056.32 | 1058988 |
| 2019-01-19 | 1056.32 | 1062.23 | 1056.04 | 1060.95 | 542449 |
| 2019-01-21 | 1062.75 | 1069.80 | 1061.81 | 1066.79 | 585278 |
| 2019-01-22 | 1066.79 | 1069.92 | 1062.89 | 1068.26 | 905422 |
| 2019-01-23 | 1068.26 | 1073.73 | 1065.92 | 1073.73 | 693971 |

# Preprocess data

```
data["y"] = data["Close"].shift(periods=-1)
data = data.drop("Volume",axis = 1)
data = data.dropna()
data_np = data.to_numpy()
```

```
data_np
```

```
array([[ 969.1 ,  970.8 ,  966.9 ,  967.1 ,  972.2 ],
       [ 967.1 ,  973.  ,  964.9 ,  972.2 ,  966.8 ],
       [ 972.4 ,  975.5 ,  966.3 ,  966.8 ,  956.6 ],
       ...,
       [1056.32, 1062.23, 1056.04, 1060.95, 1066.79],
       [1062.75, 1069.8 , 1061.81, 1066.79, 1068.26],
       [1066.79, 1069.92, 1062.89, 1068.26, 1073.73]])
```

# Preprocess data

```
def windowed_dataset(data_np, timestep):
    X = np.array(data_np[0:timestep,:-1])
    X = np.expand_dims(X, axis=-1)
    X = np.reshape(X,(1,timestep,4))
    for i in range(data_np.shape[0] - timestep - 1):
        add = data_np[i:timestep+i,:-1]
        add = np.expand_dims(add, axis=-1)
        add = np.reshape(add,(1,timestep,4))
        X = np.concatenate((X,add),axis=0)
    Y = data_np[timestep:,-1]
    return X,Y
```

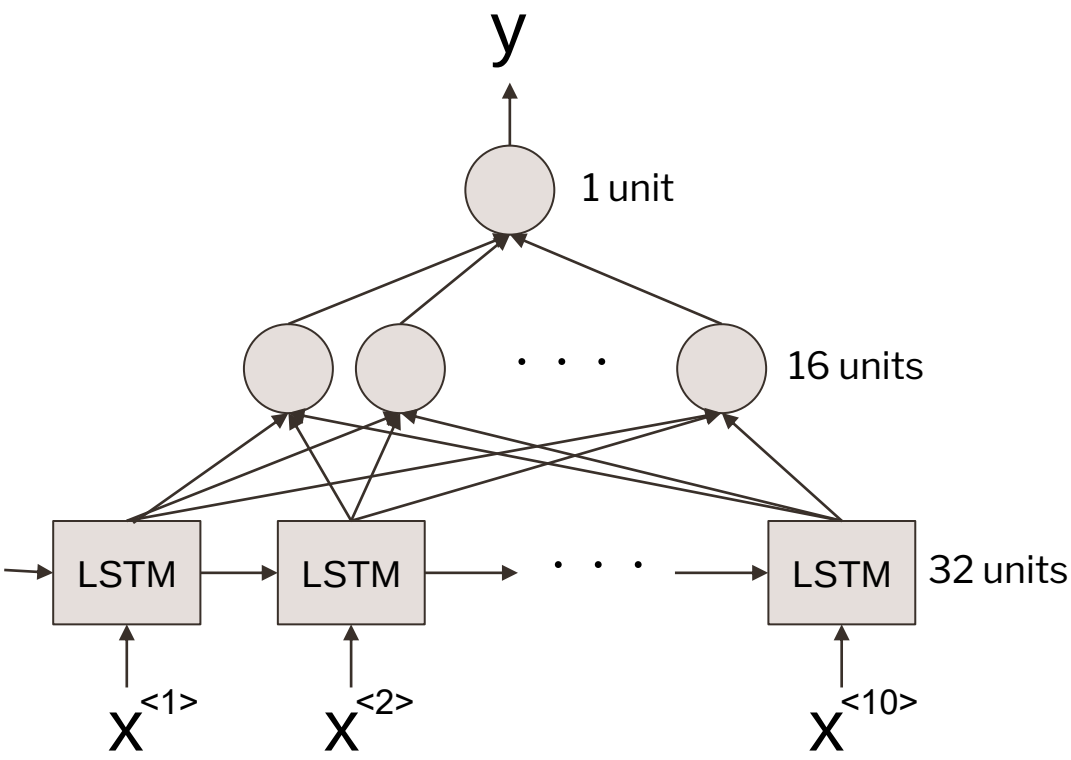
```
timestep = 10
```

```
X,Y = windowed_dataset(data_np, timestep)
X_val = X[-1000:-500,:,:]
Y_val = Y[-1000:-500]

X_test = X[-500:,:,:]
Y_test = Y[-500:]

X = X[:-1000,:,:]
Y = Y[:-1000]
```

# LSTM Model




```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_5 (LSTM)                (None, 32)                4736
_________________________________________________________________
dense_6 (Dense)              (None, 16)                528
_________________________________________________________________
dense_7 (Dense)              (None, 1)                 17
=================================================================
Total params: 5,281
Trainable params: 5,281
Non-trainable params: 0
_________________________________________________________________
```

# LSTM Model

```
model.compile(loss='mae',optimizer="adam")
history = model.fit(X,Y,
                    epochs=6,
                    batch_size=128,
                    verbose=1,
                    validation_data=(X_val,Y_val),
                    shuffle=False
                   )
```

```
Epoch 1/6
46/46 [==============================] - 1s 13ms/step - loss: 18.6380 - val_loss: 17.3014
Epoch 2/6
46/46 [==============================] - 0s 7ms/step - loss: 161.7208 - val_loss: 303.9629
Epoch 3/6
46/46 [==============================] - 0s 7ms/step - loss: 26.6796 - val_loss: 17.3866
Epoch 4/6
46/46 [==============================] - 0s 7ms/step - loss: 16.0701 - val_loss: 17.0256
Epoch 5/6
46/46 [==============================] - 0s 6ms/step - loss: 16.9069 - val_loss: 19.8780
Epoch 6/6
46/46 [==============================] - 0s 7ms/step - loss: 16.0735 - val_loss: 18.9587
```

# Backtesting

- Open Source: Zipline


- Our algorithm

```
money = 100000
stock = 0
worth = []
moneys = []
stocks = []
for i in range(5797,6797):
    yesterday_close = data.loc[i,"Close"]
    today_close = predict(i)
    today_high = data.loc[i+1,"High"]
    today_low = data.loc[i+1,"Low"]
    if yesterday_close > today_close and today_high > yesterday_close and today_low < yesterday_close:
        if stock > 2:
            money = money + yesterday_close*2
            stock = stock - 2
        elif stock == 1:
            money = money + yesterday_close
            stock = stock - 1
    elif yesterday_close < today_close and today_high > yesterday_close and today_low < yesterday_close:
        if money > 2*yesterday_close:
            money = money - yesterday_close*2
            stock = stock + 2
        elif yesterday_close < money:
            money = money - yesterday_close
            stock = stock + 1
    worth.append(stock*data.loc[i+1,"Close"] + money)
```


```
%load_ext zipline
```

```
The zipline extension is already loaded. To reload it, use:
  %reload_ext zipline
```

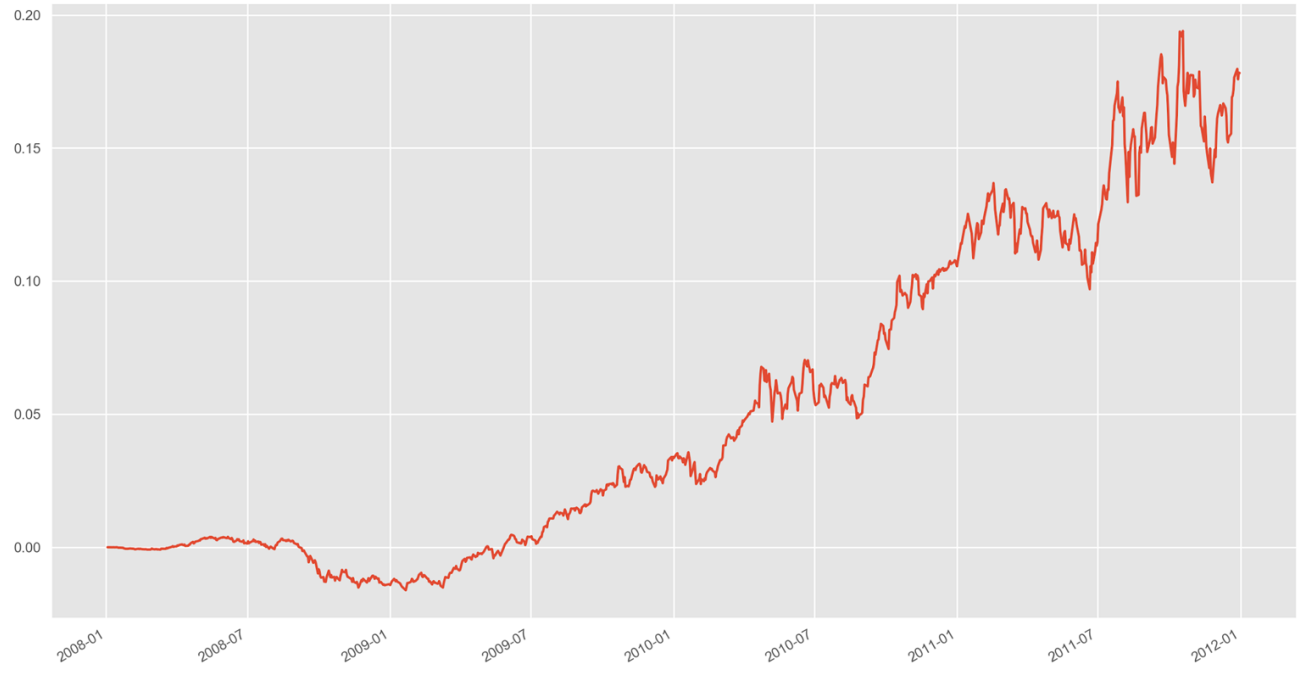
```
from zipline.api import order, record, symbol, set_benchmark

"""
def initialize(context):
    pass
"""
def initialize(context):
    # Which stock to trade
    context.stock = symbol('AAPL')
    set_benchmark(False)

def handle_data(context, data):
    order(symbol('AAPL'), 10)
    record(AAPL=data.current(symbol('AAPL'), 'price'))
```

# Zipline

```
1 %zipline --bundle quantopian-quandl --start 2008-1-1 --end 2012-1-1 -o backtest.pickle
```



Zipline

# Ingesting Data from .csv Files

```
register(
    'custom-csvdir-bundle',
    csvdir_equities(
        ['daily'],
        '/path/to/your/csvs',
    ),
    calendar_name='NYSE', # US equities
    start_session=start_session,
    end_session=end_session
)
```

```
ingest(environ,
       asset_db_writer,
       minute_bar_writer,
       daily_bar_writer,
       adjustment_writer,
       calendar,
       start_session,
       end_session,
       cache,
       show_progress,
       output_dir)
```

Collect data

↓

Register to Zipline

↓

Ingest the dataset

↓

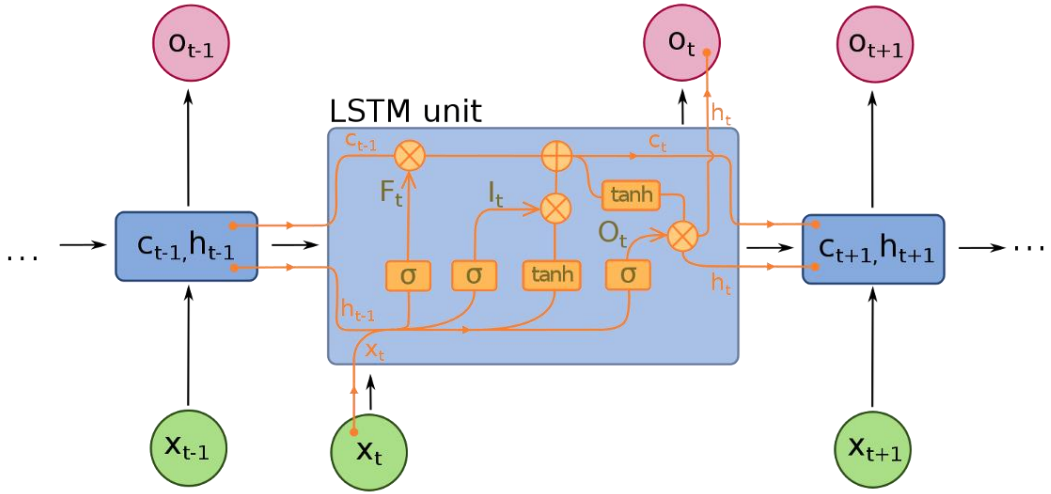
Backtest using Zipline

https://analyzingalpha.com/zipline-equity-bundle

# Our algorithm

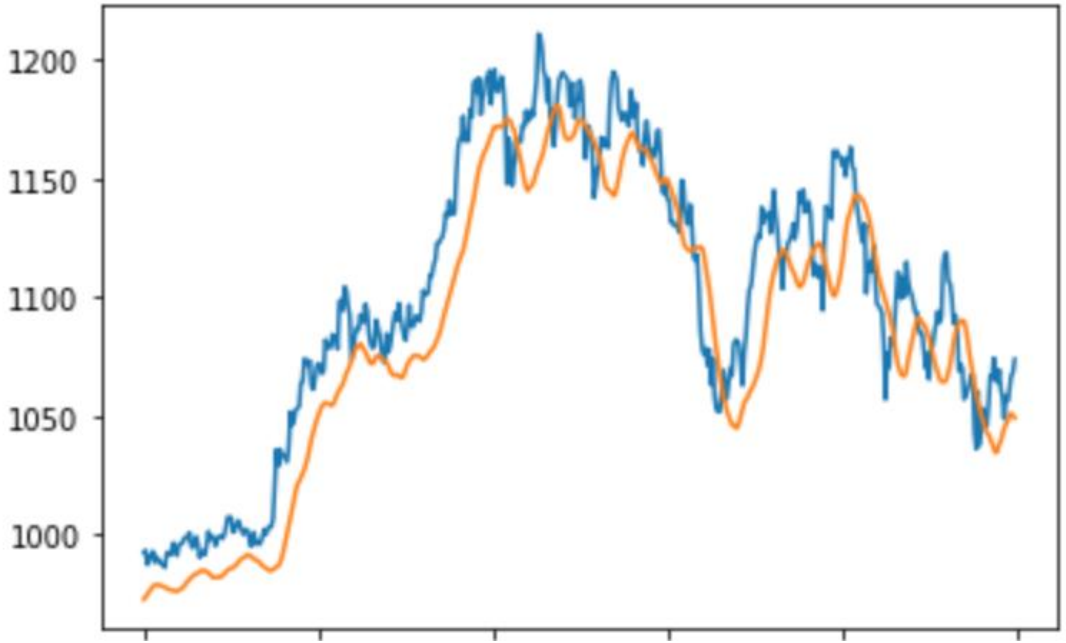
```
money = 100000
stock = 0
worth = []
moneys = []
stocks = []
for i in range(5797,6797):
    yesterday_close = data.loc[i,"Close"]
    today_close = predict(i)
    today_high = data.loc[i+1,"High"]
    today_low = data.loc[i+1,"Low"]
    if yesterday_close > today_close and today_high > yesterday_close and today_low < yesterday_close:
        if stock > 2:
            money = money + yesterday_close*2
            stock = stock - 2
        elif stock == 1:
            money = money + yesterday_close
            stock = stock - 1
    elif yesterday_close < today_close and today_high > yesterday_close and today_low < yesterday_close:
        if money > 2*yesterday_close:
            money = money - yesterday_close*2
            stock = stock + 2
        elif yesterday_close < money:
            money = money - yesterday_close
            stock = stock + 1
    worth.append(stock*data.loc[i+1,"Close"] + money)
```

# Results & Discussion



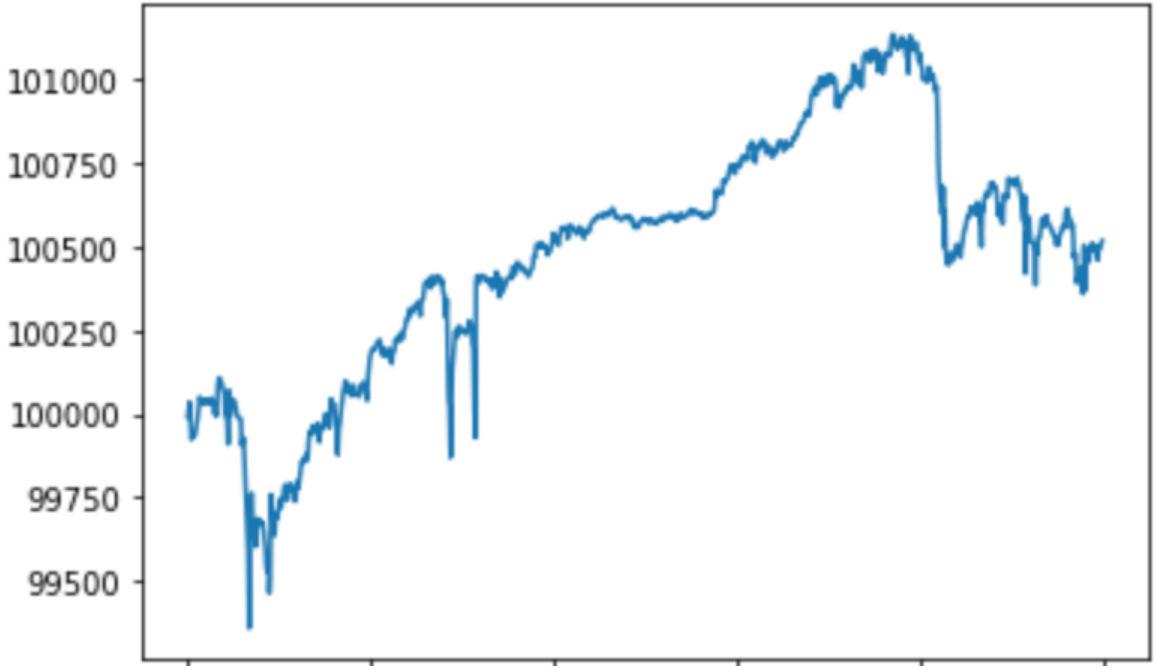
# Results & Discussion

- **Real value** vs **Predicted value**



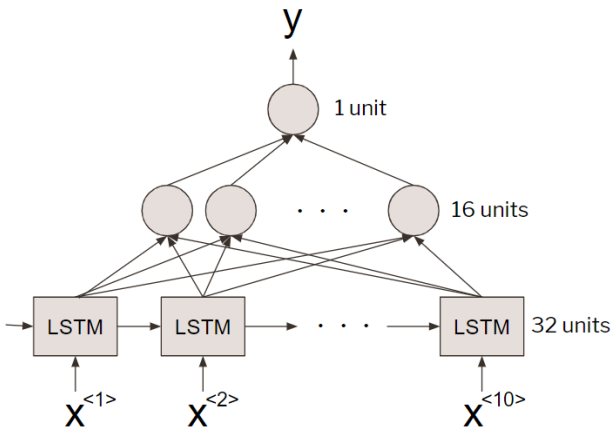
# Results & Discussion

- **Back testing**



# Results & Discussion

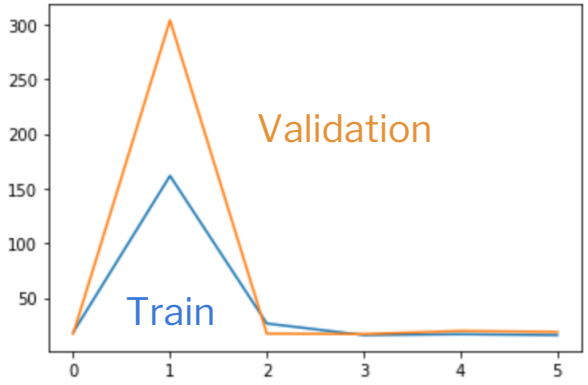
- **In training phase**, LSTM may give a constant predict value. It come from the random of weight in the beginning.
- **1 layer with 16 units** give the best performance by observation.
- **Zipline** does not suit with csv dataset but good at online data from Quantopian.



# Results & Discussion

- **Our algorithm** is working in a basic concept and there are more **deeper factors and conclusions** that we can't measure and launch in real treading.

- **Backtesting with our algorithm** (2 units/day) give the profit 500 THB at day 1,000.




```
money = 100000
stock = 0
worth = []
moneys = []
stocks = []
for i in range(5797,6797):
    yesterday_close = data.loc[i,"Close"]
    today_close = predict(i)
    today_high = data.loc[i+1,"High"]
    today_low = data.loc[i+1,"Low"]
    if yesterday_close > today_close and today_high > yesterday_close and today_low < yesterday_close:
        if stock > 2:
            money = money + yesterday_close*2
            stock = stock - 2
        elif stock == 1:
            money = money + yesterday_close
            stock = stock - 1
    elif yesterday_close < today_close and today_high > yesterday_close and today_low < yesterday_close:
        if money > 2*yesterday_close:
            money = money - yesterday_close*2
            stock = stock + 2
        elif yesterday_close < money:
            money = money - yesterday_close
            stock = stock + 1
    worth.append(stock*data.loc[i+1,"Close"] + money)
```

# Conclusion

- **The 1 layer with 16 units** may be not the good for other model. It depends on random values and chance of predicted values.

- **Presently, Our LSTM model algorithm is limited condition**. This model is recommended to modify more features for real trading.

- **Backtesting using Zipline** is working for online Quantopian dataset only.

- **Backtesting with our algorithm** gives less profit. Since our algorithm strategy buys or sells **only 2 units** in each day.

# THANK YOU

- Dr. Teerasit Termsaithong
- KMUTT Scientific programming bootcamp with python
- King Mongkut's University of Technology Thonburi (KMUTT)

# THANK YOU FOR YOUR ATTENTION


Q & A