Deep-learning-based predictive model in financial engineering

Chengjie Lin

09/29/2017

Bullet points

1. Currently used machine learning techniques in financial engineering
2. Deep learning and LSTM (long-short term memory)
3. Open source codes and libraries we can experiment with
4. Future steps
5. Reference

Financial products

- Stocks
- Futures
- Option
- Foreign exhcnage
- Bonds
- Fund
- Etc….

High frequency VS long term

human beings are already all but excluded from high frequency trading.
“Machines will likely not do well in assessing regime changes (market turning points) and forecasts which involve interpreting more complicated human responses such as those of politicians and central bankers, understanding client positioning, or anticipating crowding,” says J.P. Morgan

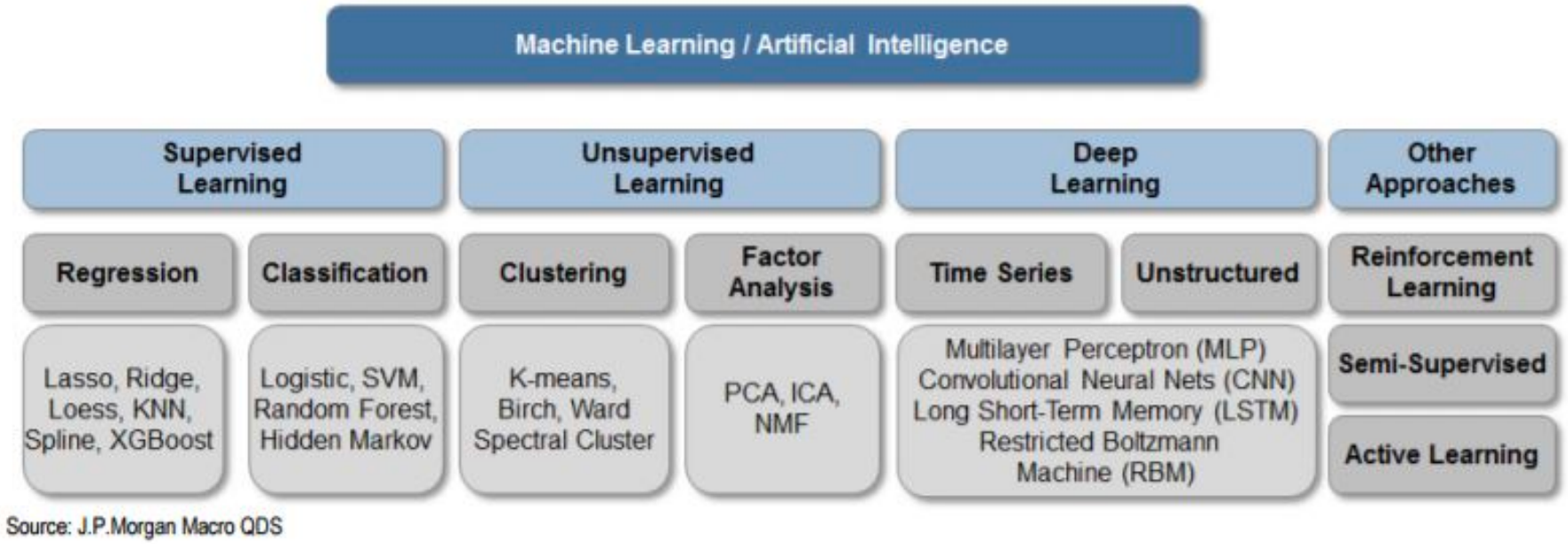
However, what if machine can….
Quickly fetch and analyze news feeds and tweets, process earnings statements, scrape websites, and trade on these instantaneously.

Too much data

- Problem
- Too much! Today's data include anything from data generated by individuals (social media posts, product reviews, search trends, etc.), to data generated by business processes (company exhaust data, commercial transaction, credit card data, etc.) and data generated by sensors (satellite image data, foot and car traffic, ship locations, etc.).

Different ML technique, Different purposes

Figure 39: Classification of Machine Learning techniques



Source: J.P.Morgan Macro QDS

- Supervised learning will be used to make trend-based predictions using sample data
- Unsupervised learning will be used to identify relationships between a large number of variables
- Deep learning systems will undertake tasks that are hard for people to define but easy to perform
- Reinforcement learning will be used to choose a successive course of actions to maximize the final reward

Some coding languages and data analysis packages

C++	
Package	**Description**
OpenCV	Real-time computer vision (Python, Java interface also available)
Caffe	Clean, readable and fast Deep Learning framework
CNTK	Deep Learning toolkit by Microsoft
DSSTNE	Deep neural networks using GPUs with emphasis on speed and scale
LightGBM	High performance gradient boosting
CRF++, CRFSuite	Segmenting/labeling sequential data & other Natural Language Processing tasks

JAVA	
Package	**Description**
MALLET	Natural language processing, document classification, clustering etc.
H20	Distributed learning on Hadoop, Spark; APIs available in R, Python, Scala, REST/JSON
Mahout	Distributed Machine Learning
MLlib in Apache Spark	Distributed Machine Learning library in Spark
Weka	Collection of Machine Learning algorithms
Deeplearning4j	Scalable Deep Learning for industry with parallel GPUs

PYTHON	
Package	**Description**
NLTK	Platform to work with human language data
XGBoost	Extreme Gradient Boosting (Tree) Library
scikit-learn	Machine Learning built on top of SciPy
keras	Modular neural network library based on Theano/Tensorflow
Lasagne	Lightweight library to build and train neural networks in Theano
Theano /Tensorflow	Efficient multi-dimensional arrays operations
MXNet	Lightweight, Portable, Flexible Distributed/Mobile Deep Learning with Dynamic, Mutation-aware Dataflow Dep Scheduler; for Python, R, Julia, Go, Javascript and more
gym	Reinforcement learning from OpenAI
NetworkX	High-productivity software for complex networks
PyMC3	Markov Chain Monte Carlo sampling toolkit
statsmodels	Statistical modeling and econometrics

Deep learning in financial engineering

- The finance literature has historically focused on stochastic models and their mathematical analysis. However, unlike in physics or other sciences, there are no fundamental laws in finance for deriving a model

Deep learning in stock market as example

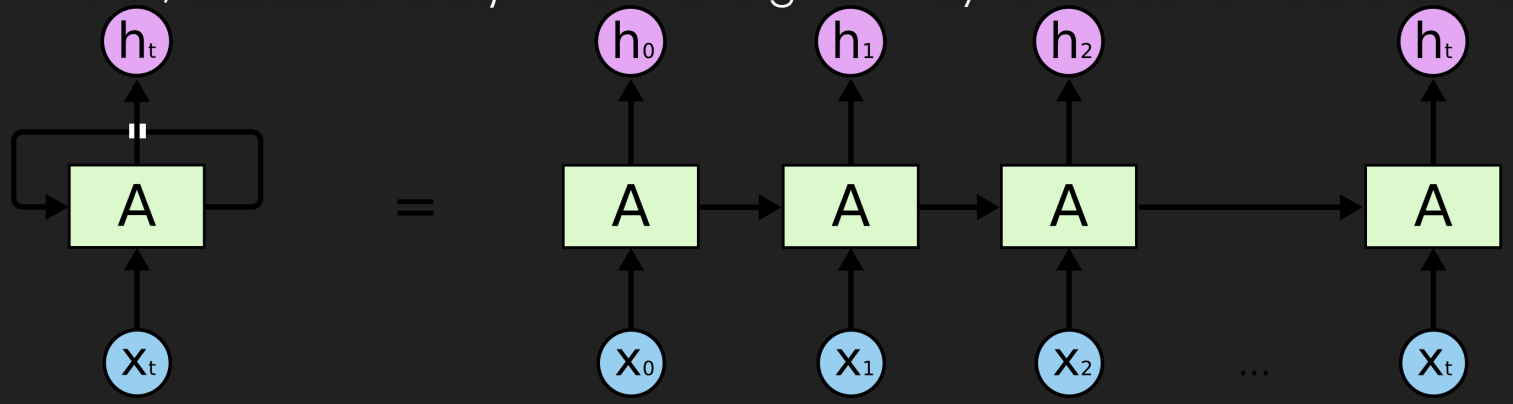
- Today, stocks are frequently traded via electronic exchanges (e.g., NASDAQ and NYSE Arca). Traders continuously submit, cancel, and execute buy and sell orders in the exchange's limit order book. Market events are often reported at the nanosecond granularity, and therefore the limit order book data generated over time is very large (terabytes to petabytes). Deep learning can model key quantities, such as the probability distribution of future price movements given the current state of supply and demand in the market.

LSTMs for Time Series Prediction in Finance

- In contrast to a number of current curricula which focus on either the mathematics-heavy side of neural networks, beginning with an introduction to backpropagation for feedforward networks, or those which focus on constructing huge black-box multi-layered "deep" neural network models, we seek to introduce and motivate the usefulness of a specific subclass of networks, Long Short-Term Memory Networks, or LSTMs, as applied to time series prediction problems in quantitative finance.
- What makes forecasting TS for financial data so hard? Strong trends have high reversal rates, there exists large amounts of noisy movement, stochasticity, usually modeled with Brownian motion etc., and low seasonality or ability to make similar assumptions about movements.

Brief Intro of LSTM

- An LSTM is a variety of Recurrent Neural Network (RNN), which is itself a flavor of ANNs, the general class of artificial neural networks.
- **What is a RNN?**
  - Recurrent neural networks are networks with loops. These loops (recurrencies) give rise to persistent information, or, in other words, create a stateful network. Not only does the behavior of the network depend on the set of inputs, but also on all prior sets of inputs (to some reasonable limit, discussed later) which have given way to the current state of the network.



LSTMs for TS forecasting

- LSTMs are particularly well suited to time-series prediction because they can "learn" and "remember" in long-term memory things like market regimes
- LSTMs, though not on the scale of so-called "deep" (some, convolutional) neural networks, are difficult to train, mainly because of the computationally expensive nature of time-based gradient descent, the size of the networks, and the amount of data over which they must be trained. Further complicating, since LSTMs are stateful, many problems require "online" training, meaning that they cannot be trained all at once by highly optimized, vectorized calculations in a batch with error computation and gradient descent over groups.
- Running training on a personal server with a GPU。
- Kera in python

Some open source codes

MFE230P: Data Science for Finance

MASTERS OF FINANCIAL ENGINEERING \ UC BERKEL

Data Science for Finance is a survey course on

Deep Learning Stock Value Predictor

Project Overview

Investment firms, hedge funds, and automated trading systems have used programming and advanced modeling to interact with and profit from the stock market since computerization of the exchanges in the 1970s[1]. Whether by

Neural-Network-with-Financial-Time-Series-Data

Introduction:

Time series is an important part of financial analysis. Today, you have more data, more data sources, and highe
frequency of data. New sources include new exchanges, social media and news sources. Today, delivery frequ
has been increased from dozens of messages every day to hundreds of thousands of messages per second.

Trading Gym project Tensorflow and kera

OpenAI Gym Environment for Trading

Environment for reinforcement-learning algorithmic trading models

The Trading Environment provides an environment for single-instrument trading using historical bar data.

See here for a jupyter notebook describing basic usage and illustrating a (sometimes) winning strategy based on policy gradients implemented on tensorflow.

Something interesting I encountered

- In terms of China stock market
  - https://zhuanlan.zhihu.com/p/27112144

Future step

- Run open source test code first and literature reviews
- Build models
- Optimize by multiple means (TRUST-TECH might involve)
- Other financial products might involve

Reference

https://news.efinancialcareers.com/uk-en/285249/machine-learning-and-big-data-j-p-morgan
http://bigtheta.io/2017/04/06/notes-on-lstms-in-finance.html
https://sinews.siam.org/Details-Page/deep-learning-models-in-finance-2

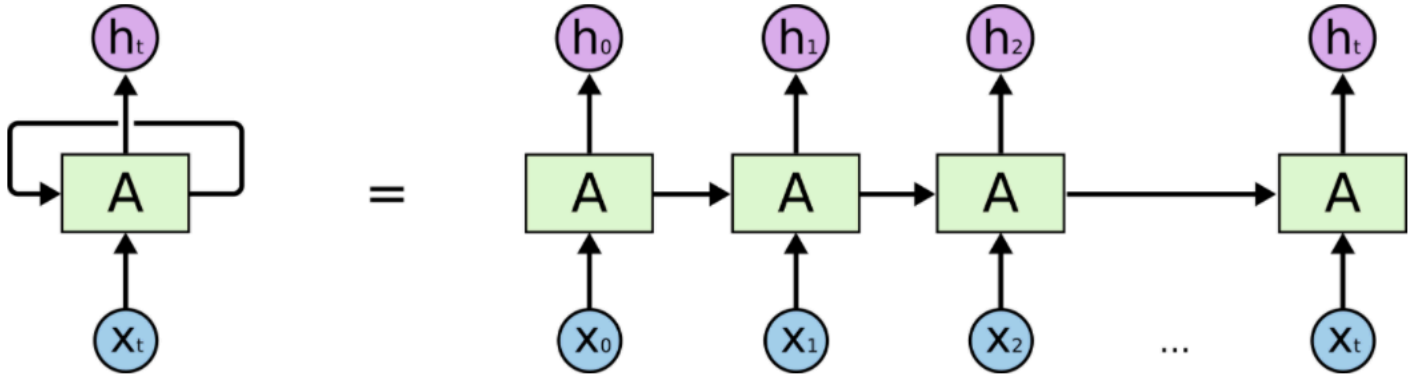
Deep-learning-based predictive model in financial engineering

Chengjie Lin

10/19/2017

Understanding LSTM Networks

- **Recurrent Neural Networks**
  - Traditional neural networks can't do this, and it seems like a major shortcoming. For example, imagine you want to classify what kind of event is happening at every point in a movie. It's unclear how a traditional neural network could use its reasoning about previous events in the film to inform later ones.
  - Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist.

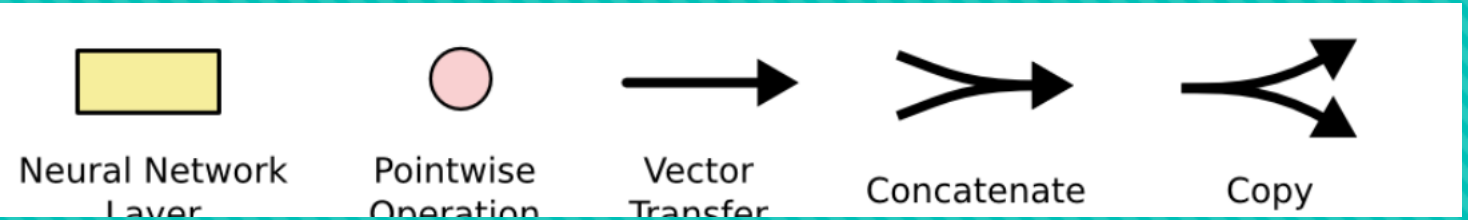


An unrolled recurrent neural network.

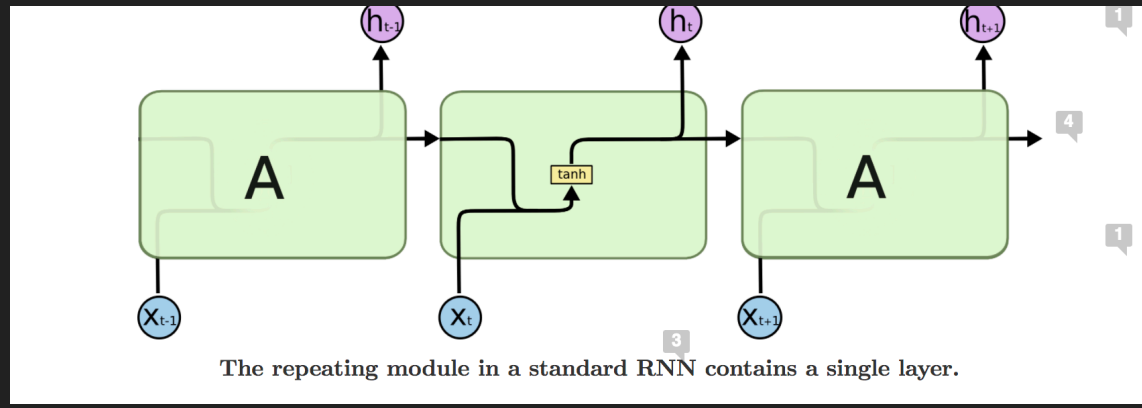
The Problem of Long-Term Dependencies

- Sometimes, we only need to look at recent information to perform the present task. For example, consider a language model trying to predict the next word based on the previous ones. If we are trying to predict the last word in "the clouds are in the *sky*," we don't need any further context – it's pretty obvious the next word is going to be sky. In such cases, where the gap between the relevant information and the place that it's needed is small, RNNs can learn to use the past information.
- But there are also cases where we need more context. Consider trying to predict the last word in the text "I grew up in France… I speak fluent *French*." Recent information suggests that the next word is probably the name of a language, but if we want to narrow down which language, we need the context of France, from further back. It's entirely possible for the gap between the relevant information and the point where it is needed to become very large.

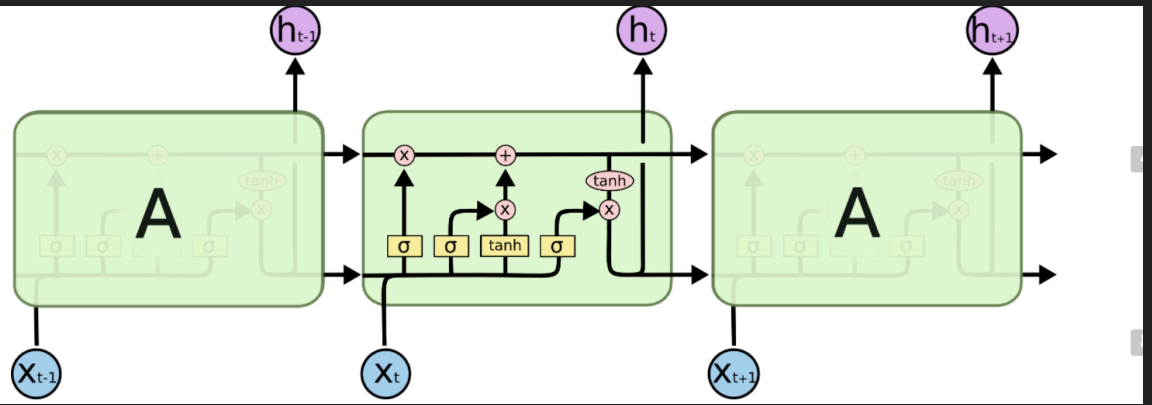
LSTM Networks



- LSTMs are explicitly designed to avoid the long-term dependency problem.
- LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in a standard RNN contains a single layer.



Open source codes testing and analysis

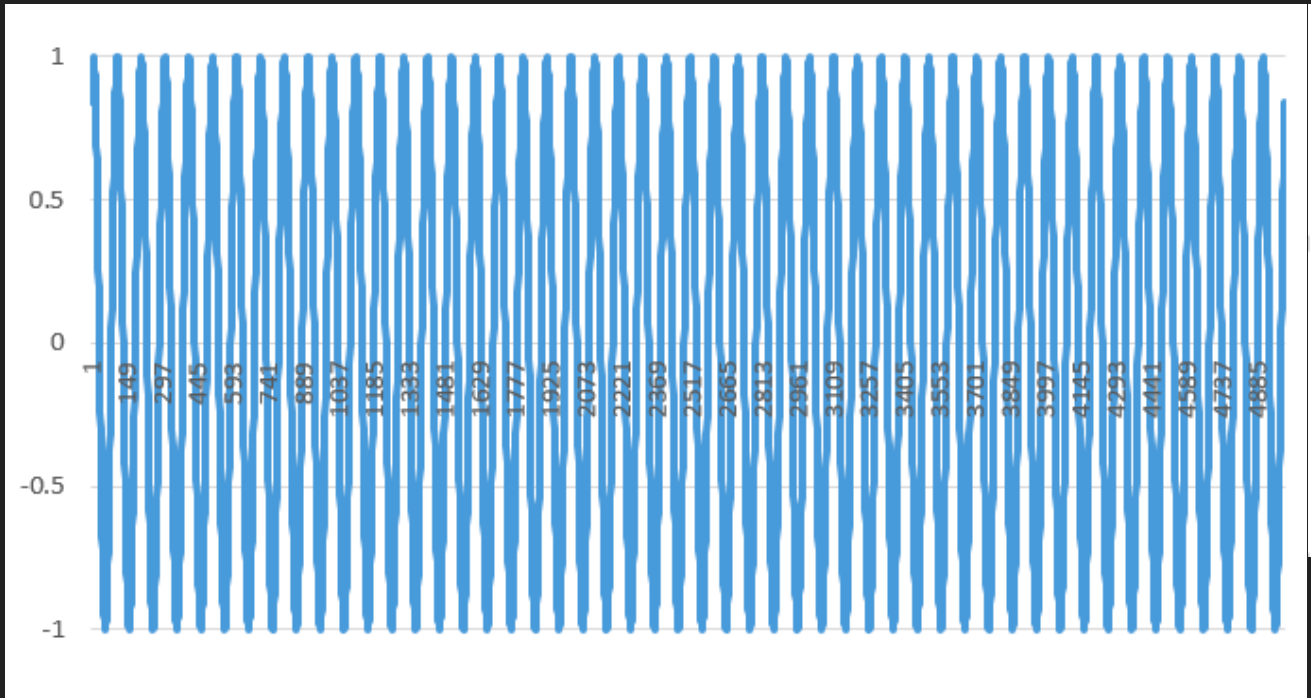
- **LSTM Neural Network for Time Series Prediction**
- **Deep-Learning-Time-Series-Prediction-using-LSTM-Recurrent-Neural-Networks**
- **US Stock Market Prediction by LSTM**
- **Predicting-Stock-Recurrent-Neural-Network** // different optimizer
- **AWS instance with GPU / Google Cloud**

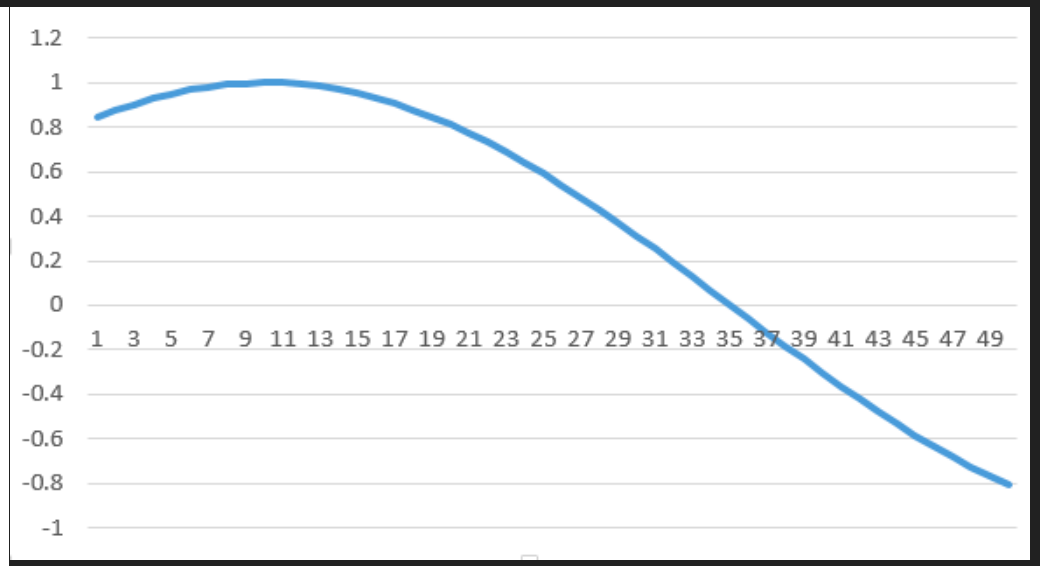
LSTM Neural Network for Time Series Prediction

- Check the codes
- Literature review and code analysis
- Find the main issue discussed here

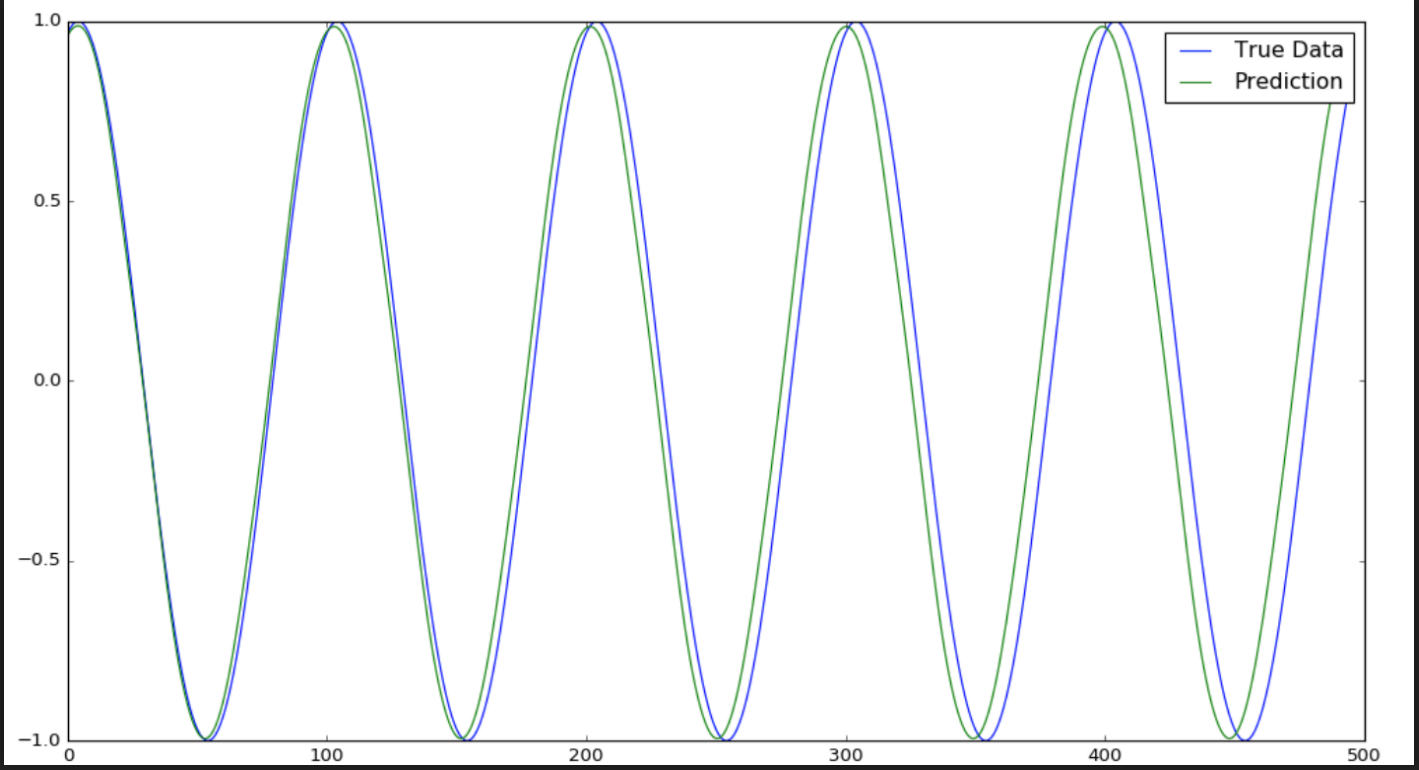
A SIMPLE SIN WAVE

- start with the most basic thing we can think of that's a time series; your bog standard sin wave function.





Good results with structured data

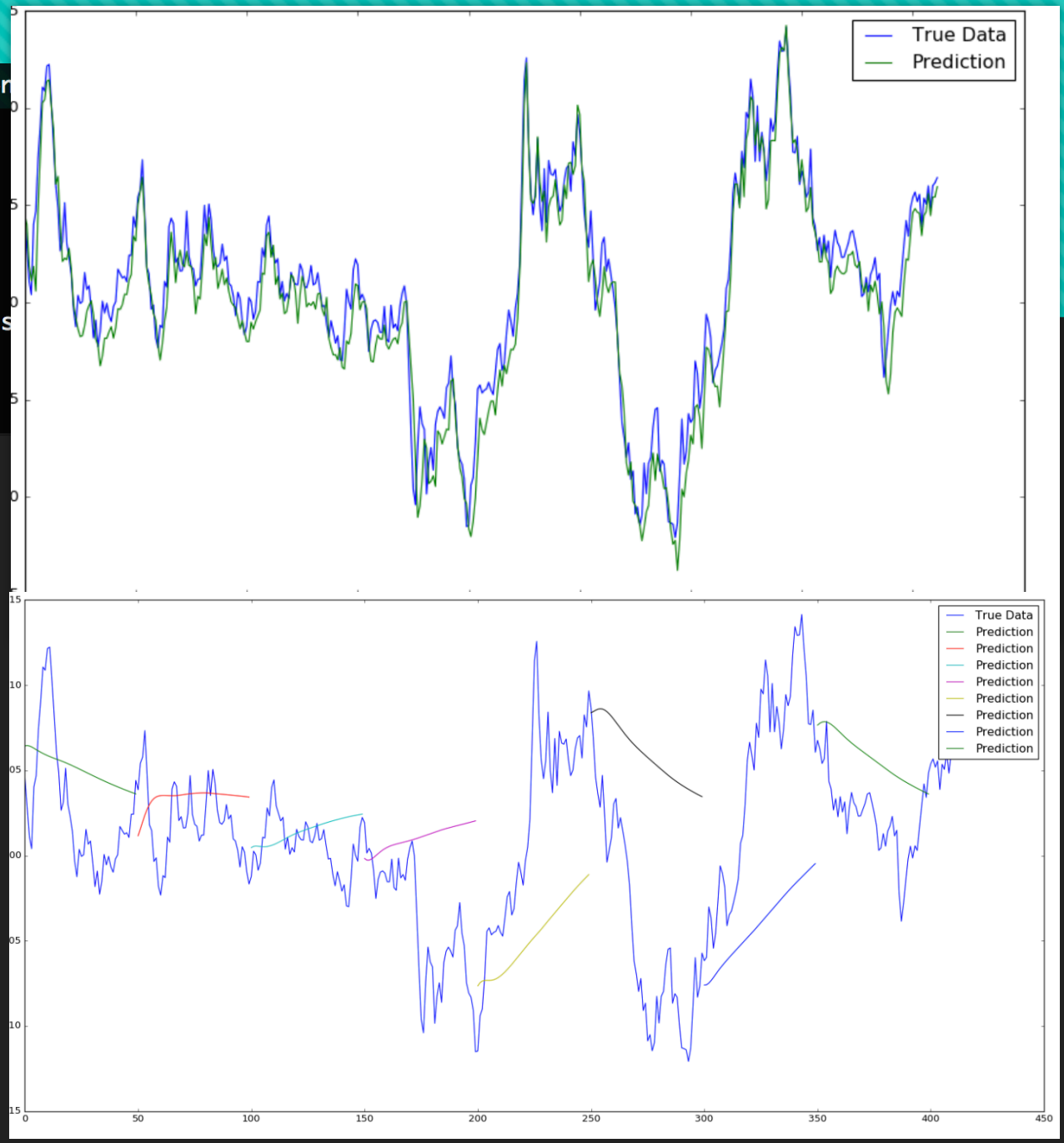


Now to the stock

- A stock time series is unfortunately not a function that can be mapped. It can best described more as a random walk, which makes the whole prediction thing considerably harder. But what about the LSTM identifying any underlying hidden trends?
- Running the adjusted returns of a stock index through a network would make the optimization process shit itself and not converge to any sort of optimums for such large numbers. So we need to normalize it.

```
(tensorflow) Chengjies-MBP:LSTM-Neural-Network-for-Time-Series-Prediction
 chengjielin$ python run.py
Using TensorFlow backend.
> Loading data...
> Data Loaded. Compiling...
> Compilation Time :  0.026814937591552734
Train on 3523 samples, validate on 186 samples
Epoch 1/1
3523/3523 [==============================] - 9s - loss: 0.0019 - val_loss
1e-04
Training duration (s) :  28.476176023483276
```

Start out with one with 1 epoch and one with 50 epochs, we investigate this further by limiting our prediction sequence to 50 future time steps and then shifting the initiation window by 50 each time, in effect creating many independent sequence predictions of 50 time steps:



Issues and tips

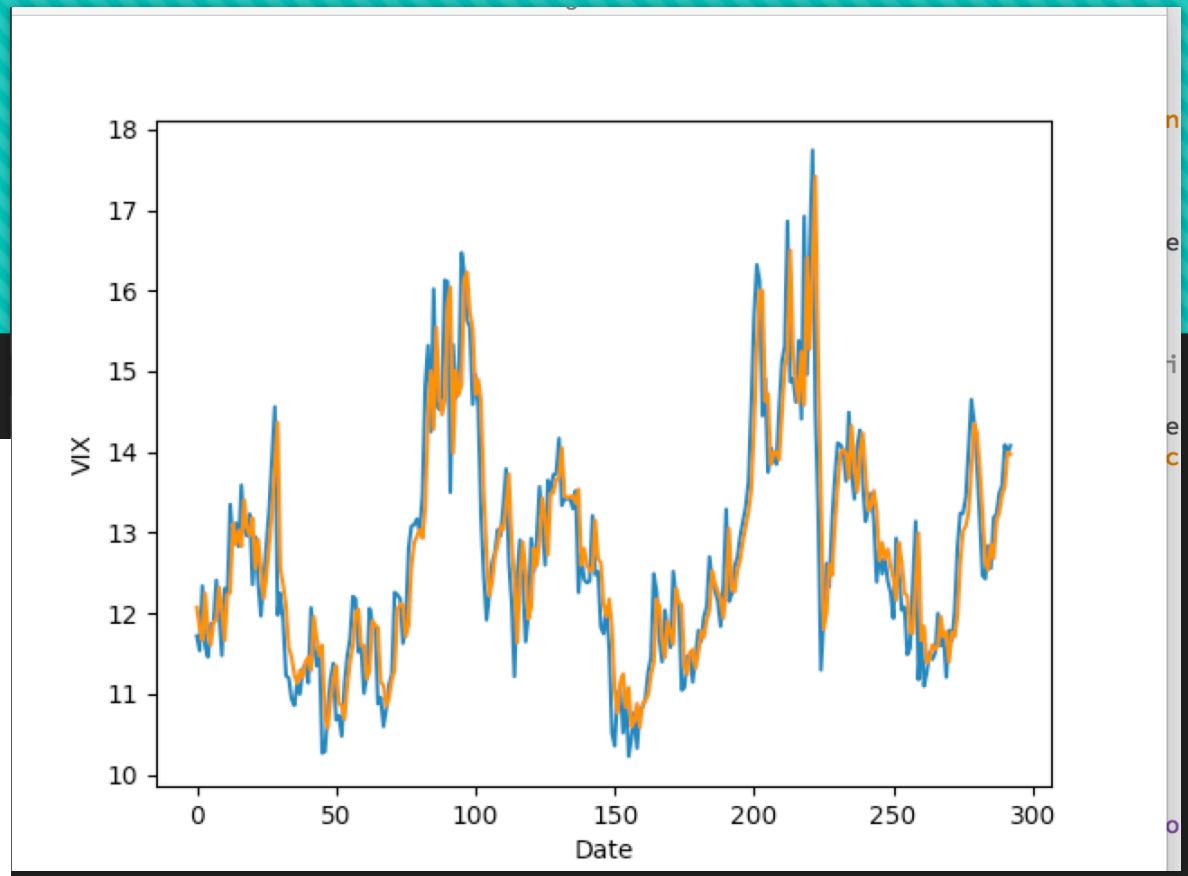
- Epochs are large, very slow for stock predicting
- Only consider the data, while it is far from enough due to the fact that there are so many underlying factors that influence daily price fluctuations; from fundamental factors of the underlying companies, macro events, investor sentiment and market noise etc.
- Model selection
- Optimizer selection

Deep-Learning-Time-Series-Prediction-using-LSTM-Recurrent-Neural-Networks

- Historical data for VOLATILITY S&P 500 (^VIX) from Jan. 02, 2005 to Sep. 26, 2016.
- Similar to the previous one
- VIX is the ticker symbol for the Chicago Board Options Exchange (CBOE) Volatility Index, which shows the market's expectation of 30-day volatility. It is constructed using the implied volatilities of a wide range of S&P 500 index options. This volatility is meant to be forward looking, is calculated from both calls and puts, and is a widely used measure of market risk, often referred to as the "investor fear gauge."

```
Epoch 43/50
2509/2509 [==============================] - 1s - loss: 7.5037 - val_loss: 3.90
4
Epoch 44/50
2509/2509 [==============================] - 2s - loss: 7.4085 - val_loss: 4.25
1
Epoch 45/50
2509/2509 [==============================] - 2s - loss: 7.3652 - val_loss: 3.69
9
Epoch 46/50
2509/2509 [==============================] - 2s - loss: 7.0111 - val_loss: 3.57
8
Epoch 47/50
2509/2509 [==============================] - 2s - loss: 6.9409 - val_loss: 3.57
1
Epoch 48/50
2509/2509 [==============================
7
Epoch 49/50
2509/2509 [==============================
```

```
1.18397389215087891e+01  1.15399999999999915e+01
1.16880722045898437e+01  1.23399999999999986e+01
1.22790641784667968e+01  1.15899999999999986e+01
1.17096242904663085e+01  1.14600000000000085e+01
1.15621547698974609e+01  1.18699999999999922e+01
1.18735160827636718e+01  1.18800000000000078e+01
1.18835744857788085e+01  1.24100000000000014e+01
1.23018226623535156e+01  1.20099999999999979e+01
1.20123596191406250e+01  1.14800000000000043e+01
1.16006317138671875e+01  1.23100000000000050e+01
1.22312259674072265e+01  1.22500000000000000e+01
1.22279911041259765e+01  1.33499999999999964e+01
1.31436414718627929e+01  1.28699999999999922e+01
1.28157138824462890e+01  1.31199999999999922e+01
1.30295877456665039e+01  1.28300000000000007e+01
1.28448076248168945e+01  1.35899999999999986e+01
1.34932537078857421e+01  1.30399999999999915e+01
1.30850963592529296e+01  1.29600000000000085e+01
1.29899864196777343e+01  1.32300000000000043e+01
1.32472772598266601e+01  1.23599999999999943e+01
1.25282821655273437e+01  1.29499999999999929e+01
1.29610500335693359e+01  1.23900000000000057e+01
1.25303268432617187e+01  1.19700000000000064e+01
1.21398983001708984e+01  1.24199999999999993e+01
1.24671611785888671e+01  1.28699999999999922e+01
1.28550777435302734e+01  1.33100000000000050e+01
1.32422113418579101e+01  1.39299999999999972e+01
1.38152647018432617e+01  1.45600000000000050e+01
```



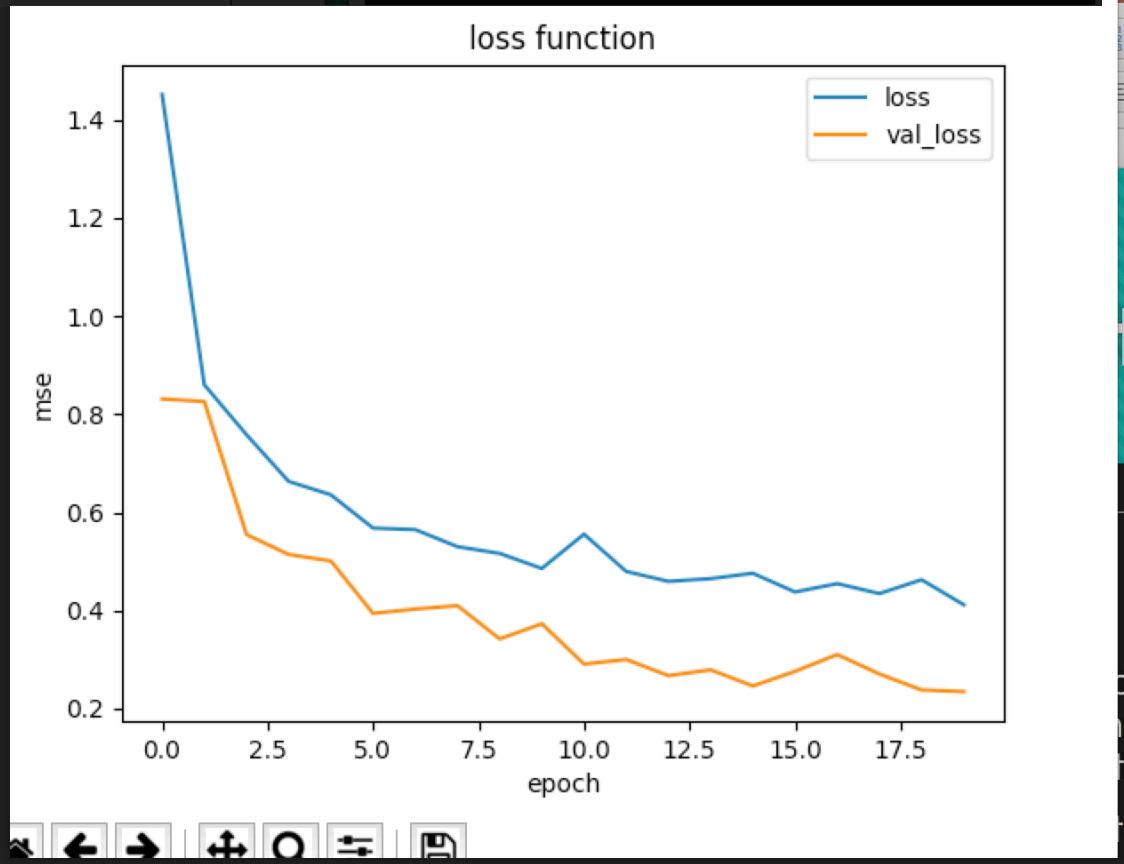
US Stock Market Prediction by LSTM

- **Input**
  - Get data from quandl (wiki database)
  - Features set = ['*Adj. Open*', '*Adj. High*', '*Adj. Low*', '*Adj. Close*', '*Adj. Volume*'] (default feature set)
  - Create more features by default feature set (3-days MA, 5-days MA, or some technical indicators...)
- **Output**
  - Here we can use two types of output, regression and classification.
  - Regression: loss function is 'mse' for model
  - Classification: it can be easily done by applying KMeans algorithm to find the catogories. The loss function is 'categorical_crossentropy' for the model
  - There is another parameter in my model. 'out_type' could be 'MA' or 'close'. It means the target is moving average or just close price for how many days(*pred_len*) later.

- **Preprocessing**
  - Divide the data for each row into price and volume.
  - Do standard normalization for price and volume separately.
    Here is an assumption: if winodw_len is large enough, it will be a Gaussian Distribution. Normalize to zero mean and unit variance.
  - Normalization for row data and do some data reshape
    Use sklearn preprocessing.StandardScaler()
  - Rearrange to original format
  - Save the *scaler_price* (it is necessary while doing the inverse transformation later)
- **Cross validation**
  - *In this part, time-series data will be shuffled. It means there is no stateful information for each row data.*
  - Split *(X, y)* into *(X_train, y_train) + (X_test, y_test)*
  - *(X_train, y_train)* is for training the LSTM model.
  - *(X_test, y_test)* is for test later.

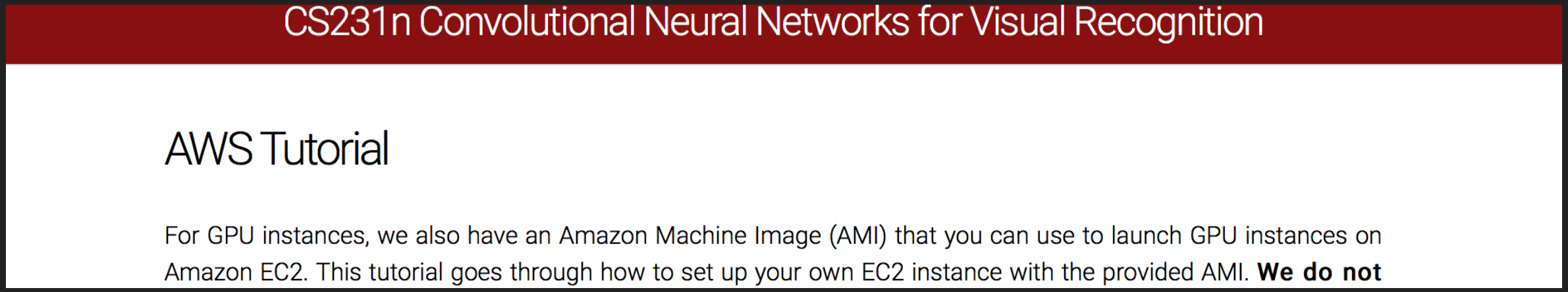
1/4/10	18.078991	18.5373188	18.0608394	18.3694166	2171500
1/5/10	18.0880668	18.442023	18.0880668	18.4238714	1481700
1/6/10	18.3331134	18.4783262	18.2968102	18.4147956	2446700
1/7/10	18.3784924	18.4238714	18.0517636	18.169749	2832800
1/8/10	18.1788248	18.4238714	18.0426878	18.3603408	1449200
1/11/10	18.3694166	18.5146293	18.2423554	18.4057198	1788100
1/12/10	18.2968102	18.4238714	17.988233	18.0880668	1732800
1/13/10	18.0154604	18.2877344	17.9247024	18.260507	2277300
1/14/10	18.169749	18.4147956	18.0880668	18.3784924	1497900
1/15/10	18.351265	18.5146293	18.1788248	18.2968102	2327300
1/19/10	18.2968102	18.3694166	18.1878098	18.2695828	1052300
1/20/10	18.169749	18.2423554	17.8702476	18.0880668	1920500
1/21/10	18.1425216	18.1425216	17.1532595	17.2167901	4474300
1/22/10	17.1623353	17.2712449	16.59056	16.6450148	3887000
1/25/10	16.8719098	16.8719098	16.5633326	16.6540906	2642600
1/26/10	16.4725746	16.726697	16.4271956	16.4271956	2569300
1/27/10	16.4453472	16.7176212	16.4181198	16.6540906	3089100
1/28/10	16.7085454	16.8356066	16.454423	16.5542568	1790100
1/29/10	16.5814842	16.7630002	16.3273619	16.3273619	3012900
2/1/10	16.4725746	16.8809856	16.3818167	16.7176212	4002900
2/2/10	16.7539244	16.9898952	16.6994696	16.862834	1849600
2/3/10	17.4618367	17.4890641	16.9898952	17.2984723	6138500
2/4/10	17.1804869	17.1804869	16.635939	16.6586285	3812000
2/5/10	16.5996358	17.0715774	16.5996358	17.0625016	2936700
2/8/10	17.1260321	17.1986385	16.3273619	16.3818167	4219400
2/9/10	16.5996358	16.8083792	16.3908925	16.5179536	5746300
2/10/10	16.4634988	16.5724084	16.2638313	16.4181198	3922900
2/11/10	16.3636651	16.6268632	16.3273619	16.4725746	3077100
2/12/10	16.3636651	16.6995604	16.2184523	16.5996358	4129200
2/16/10	16.7448486	16.8991372	16.5814842	16.8356066	3088200
2/17/10	16.8900614	16.9626678	16.681318	16.8537582	2380900
2/18/10	16.5996358	16.9263646	16.5996358	16.8900614	2217700
2/19/10	16.8809856	16.998971	16.7811518	16.908213	2067300
2/22/10	16.8809856	16.9263646	16.7993034	16.817455	1220800
2/23/10	16.8809856	16.9354404	16.6903938	16.7357728	2142800
2/24/10	16.7902276	16.9808194	16.6631664	16.8809856	2183500
2/25/10	16.6087116	16.9717436	16.59056	16.9445162	1968800

```
Epoch 20/20
1373/1373 [==============================] - 21s - loss: 0.4117 - val_loss
48
 ############## validation on test data ##############
scaled data mse:  0.267498801123
 ############## validation on train/test lately data ##############
scaled data mse:  0.200576760378
 ############## validation on valid data ##############
scaled data mse:  0.126092320544
 ############## validation on lately data ##############
scaled data mse:  nan
```



AWS GPU Instance with Jupyter Notebook Google Cloud with GPUs

- http://cs231n.github.io/aws-tutorial/
- http://cs231n.github.io/gce-tutorial/
- http://efavdb.com/deep-learning-with-jupyter-on-aws/



CS231n Convolutional Neural Networks for Visual Recognition

AWS Tutorial

For GPU instances, we also have an Amazon Machine Image (AMI) that you can use to launch GPU instances on Amazon EC2. This tutorial goes through how to set up your own EC2 instance with the provided AMI. **We do not**

Cost….Maybe a finance-oriented image

p2.xlarge	4	12	61	EBS Only	$0.9 per Hour
p2.8xlarge	32	94	488	EBS Only	$7.2 per Hour
p2.16xlarge	64	188	732	EBS Only	$14.4 per Hour
g3.4xlarge	16	47	122	EBS Only	$1.14 per Hour
g3.8xlarge	32	94	244	EBS Only	$2.28 per Hour
g3.16xlarge	64	188	488	EBS Only	$4.56 per Hour

Future work

- Get a better understanding of the codes
- Optimize
- Strategy thinking and develop
- Literature review
- Try to get it work with larger epochs
- Feasibility

Something Interesting…..
stock-predict-by-RNN-LSTM

- Theano backend
- CUDA
- OpenBlas. In scientific computing, OpenBLAS is an open source implementation of the BLAS (Basic Linear Algebra Subprograms) API with many hand-crafted optimizations for specific processor types.
- NYSE
- https://github.com/blockchain99/stock-predict-by-RNN-LSTM

11.3 Presentation

- 1. What have been done: Run several models and equipped with knowledge
  - LSTM Neural Network for Time Series Prediction
  - Deep-Learning-Time-Series-Prediction-using-LSTM-Recurrent-Neural-Networks
  - US Stock Market Prediction by LSTM
- 2. Decided on the strategy
- 3. Paper we will focus on and derive our model with
- 4. Advice from Professor : Command + soldier model
- 5. We set up the work environment- AWS+IPython Notebook
- 6. Goal

Commander + Soldiers

- Current soldier type: Wavelet Transform+ Stacked autoencoder +LSTM
- Future soldier type: CNN in K curve of stock; Restricted Boltzmann Machine; Deep belief network etc.
- Long term project and can add more soldiers into the project if the framework is built.

Papers we currently use to create our soldiers

- Using Financial Reports to Predict Stock Market Trends With Machine Learning Techniques
  - Deep belief network ; Restricted Boltzmann Machine; NLP
- A deep learning framework for financial time series using stacked autoencoders and long-short term memory
  - WT+SAE+LSTM

WT+SAE+LSTM

- Wavelet Transform: Denoise and preprocess the data
  - Get rid of some useless high frequency and low frequency
- SAE: Find deep features and decrease the dimensions (Can be related to PCA in some sense)
- LSTM: Widely used for time series and well explained in my previous presentations

Our working environment

- AWS (N. California)
- AMI: cs231n_caffe_torch7_keras_lasagne_v2
- Coding environment: IPython Notebook

Goal

- Build a good model
- Actual apply it to trading process
- A good paper

WT+SAE+LSTM in details

W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," PLoS ONE, vol. 12, no. 7, pp. e0180944–24

- DL framework can be multiple ones combined: WT (wavelet transforms), SAEs (stacked auto-encoders), LSTM (long-short term memory)
- 3 steps: 1) decomposing stock price tine series by WT to eliminate noise. 2) SAEs to generate deep high-level features for predicting the stock price 3) denoising features are fit into LSTM to forecast the next day's closing price

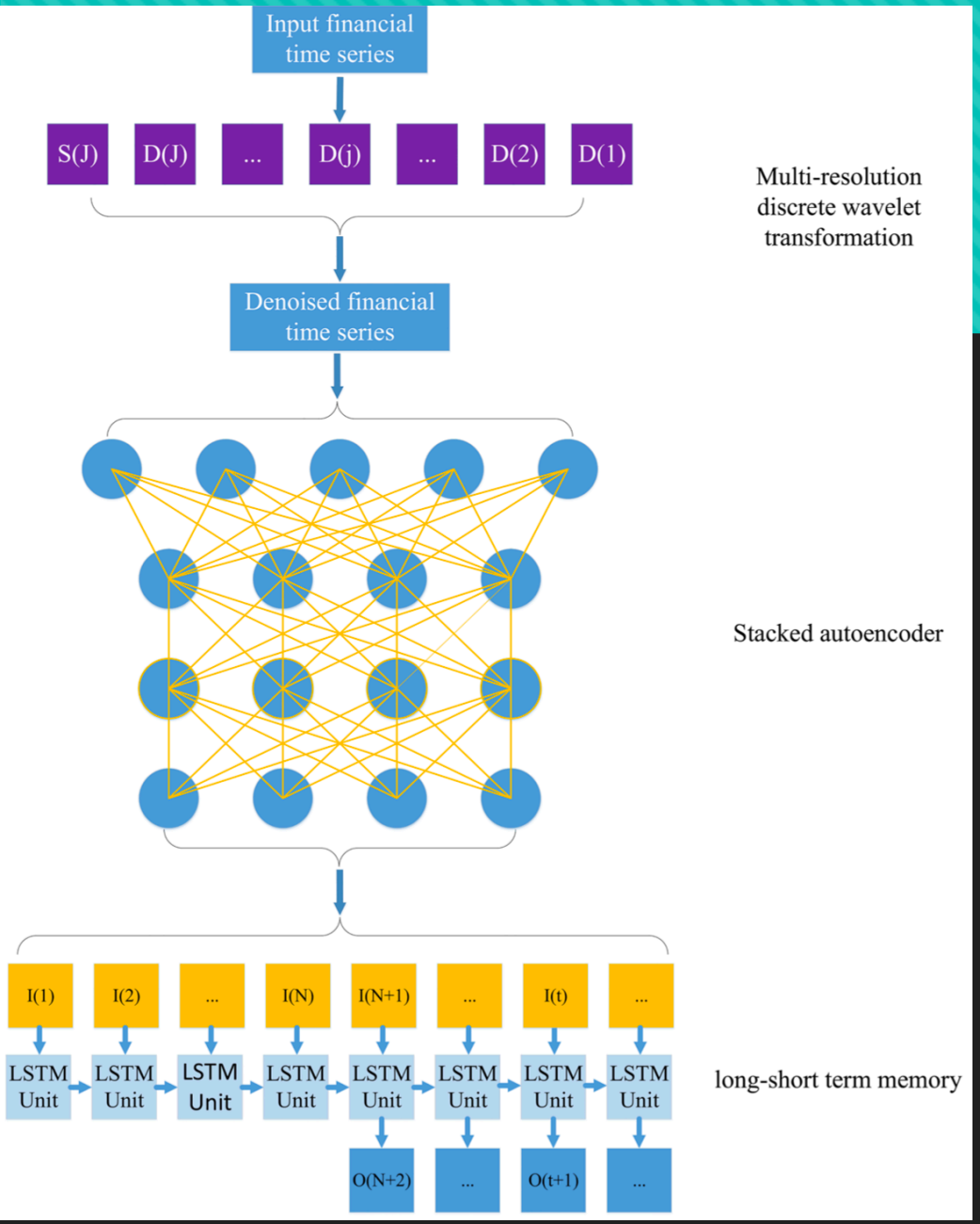
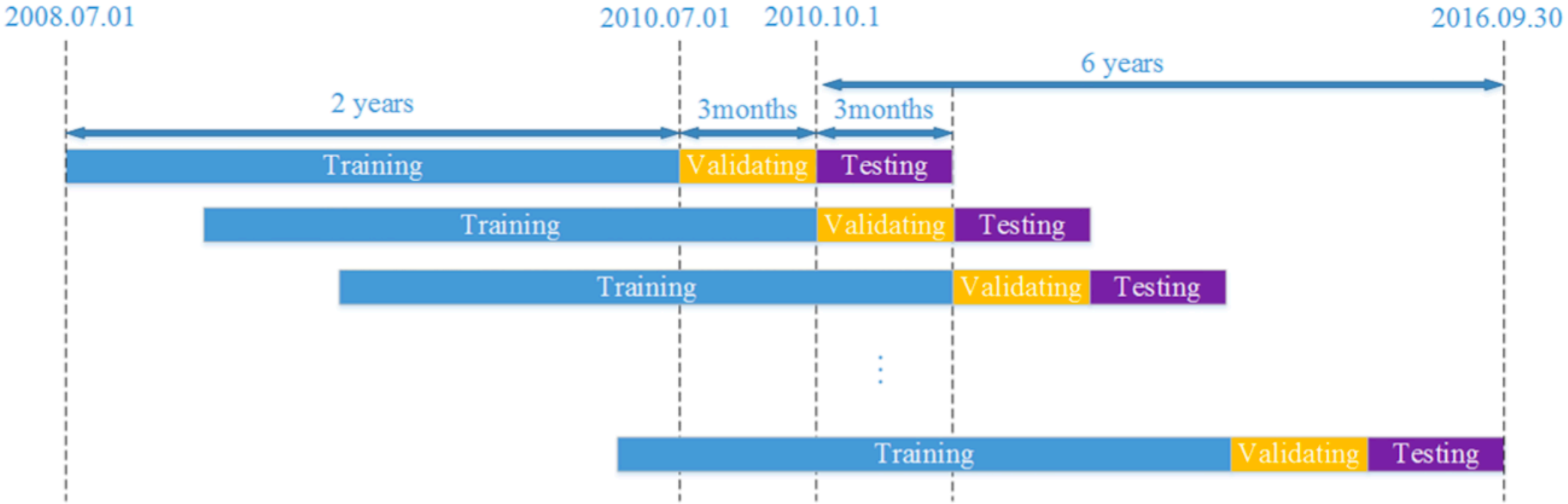


Table 1. Description of the input variables.

Name	Definition/Implication
	Panel A. Daily Trading Data
Open/Close Price	nominal daily open/close price
High/Low Price	nominal daily highest/lowest price
Trading volume	Daily trading volume
	Panel B. Technical Indicator
MACD	Moving average convergence divergence: displays trend following characteristics and momentum characteristics.
CCI	Commodity channel index: helps to find the start and the end of a trend.
ATR	Average true range: measures the volatility of price.
BOLL	Bollinger Band: provides a relative definition of high and low, which aids in rigorous pattern recognition
EMA20	20 day Exponential Moving Average
MA5/MA10	5/10 day Moving Average
MTM6/MTM12	6/12 month Momentum: helps pinpoint the end of a decline or advance
ROC	Price rate of change: shows the speed at which a stock's price is changing
SMI	Stochastic Momentum Index: shows where the close price is relative to the midpoint of the same range.
WVAD	Williams's Variable Accumulation/Distribution: measures the buying and selling pressure.
	Panel C. Macroeconomic Variable
Exchange rate	US dollar Index
Interest rate	Interbank Offered Rate

Stock Price Forecasting - input



6 market indices and corresponding index are chosen to examine the performance of the proposed model

Table 3. Predictive accuracy in developing markets.

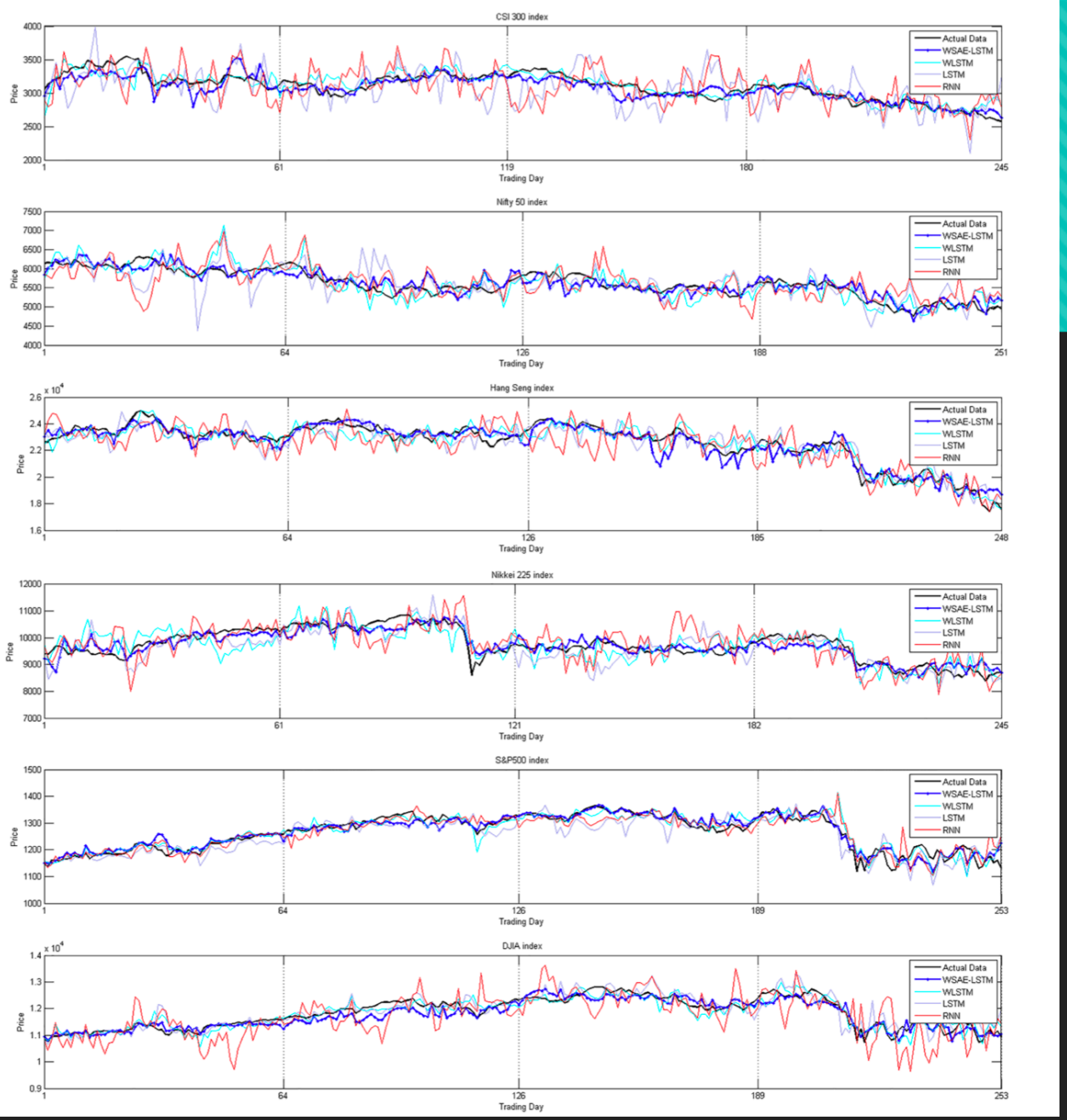
	CSI 300 index							Nifty 50 index						
Year	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Average	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Average
Panel A. MAPE														
WSAEs-LSTM	0.025	0.014	0.016	0.011	0.033	0.016	0.019	0.024	0.019	0.019	0.019	0.018	0.017	0.019
WLSTM	0.025	0.029	0.021	0.020	0.038	0.033	0.028	0.034	0.038	0.030	0.025	0.020	0.029	0.029
LSTM	0.067	0.077	0.047	0.036	0.053	0.055	0.056	0.043	0.034	0.035	0.035	0.027	0.029	0.034
RNN	0.062	0.087	0.052	0.060	0.059	0.075	0.066	0.051	0.038	0.034	0.032	0.036	0.035	0.038
Panel B. R														
WSAEs-LSTM	0.861	0.959	0.955	0.957	0.975	0.957	0.944	0.878	0.834	0.665	0.972	0.774	0.924	0.841
WLSTM	0.841	0.801	0.919	0.864	0.977	0.803	0.868	0.803	0.533	0.312	0.958	0.722	0.844	0.695
LSTM	0.440	0.273	0.629	0.742	0.962	0.656	0.617	0.596	0.601	0.027	0.904	0.515	0.772	0.569
RNN	0.614	0.363	0.670	0.343	0.943	0.555	0.581	0.506	0.661	0.263	0.929	0.278	0.704	0.557
Panel C. Theil U														
WSAEs-LSTM	0.017	0.009	0.011	0.007	0.023	0.011	0.013	0.016	0.013	0.013	0.013	0.012	0.012	0.013
WLSTM	0.018	0.019	0.014	0.013	0.024	0.022	0.018	0.023	0.025	0.021	0.016	0.014	0.018	0.019
LSTM	0.042	0.049	0.030	0.024	0.031	0.036	0.035	0.030	0.023	0.025	0.024	0.018	0.020	0.024
RNN	0.044	0.053	0.037	0.041	0.036	0.046	0.043	0.034	0.024	0.024	0.021	0.026	0.023	0.025

https://doi.org/10.1371/journal.pone.0180944.t003

Table 6. Profitability performance of each model.

Year	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Average	Year 1	Year 2	Year 3	Year 4	Year 5	Year 6	Average
Panel A. Developing market														
	CSI 300 Index							**Nifty 50 Index**						
WSAEs-LSTM	46.428	59.580	65.685	71.897	63.951	70.613	63.026	56.911	51.710	66.765	37.107	31.882	28.134	45.418
WLSTM	26.766	21.942	36.502	20.304	89.984	42.283	39.630	13.835	19.254	13.679	22.464	44.032	28.436	23.617
LSTM	-15.802	-16.802	24.082	1.345	57.903	53.479	17.368	26.247	16.509	0.921	-8.459	37.180	18.279	15.113
RNN	-3.310	1.750	8.827	-0.747	47.308	-8.630	7.533	-7.757	-10.558	19.691	22.685	29.492	-7.435	7.686
Buy-and-hold	-19.630	-12.595	3.065	0.292	35.619	3.430	1.697	-23.564	13.926	-1.510	30.885	-2.787	5.477	3.738
Panel B. Relatively developed market														
	Hang Seng Index							**Nikkei 225 Index**						
WSAEs-LSTM	75.844	81.890	54.685	49.715	64.727	59.949	64.468	53.463	37.846	81.032	48.832	59.419	76.030	59.437
WLSTM	25.697	19.221	37.410	23.149	59.030	51.220	35.955	27.774	10.196	43.143	32.411	31.252	62.569	34.558
LSTM	16.381	25.733	21.892	1.545	23.680	62.717	25.325	14.032	31.507	52.324	-15.605	8.774	26.272	19.551
RNN	5.287	40.810	-7.114	-17.024	27.845	48.463	16.378	-7.329	-6.967	8.318	-15.389	42.258	50.950	11.974
Buy-and-hold	-27.321	16.341	7.472	-2.821	-9.264	9.455	-1.023	-19.371	1.568	49.339	9.790	6.365	-6.728	6.827
Panel C. Developed market														
	S&P 500 Index							**DJIA Index**						
WSAEs-LSTM	71.316	48.351	39.239	9.940	61.187	45.950	45.997	76.742	55.433	41.700	47.034	82.503	80.522	63.989
WLSTM	40.246	25.726	21.964	15.906	31.839	18.168	25.641	47.180	35.174	30.682	29.532	30.605	54.885	38.009
LSTM	-7.633	23.138	20.710	-2.678	2.010	34.254	11.633	33.868	22.951	17.919	-5.952	4.493	37.578	18.476
RNN	18.319	11.930	11.261	-8.475	12.266	3.607	8.152	4.787	6.763	-2.094	8.730	37.368	17.814	12.228
Buy-and-hold	-12.271	22.755	13.212	13.747	-5.358	9.819	6.984	-7.860	19.070	9.725	9.620	-7.144	9.356	5.461

https://doi.org/10.1371/journal.pone.0180944.t006



$$MAPE = \frac{\sum_{t=1}^{N} \left| \frac{y_t - y_t^*}{y_t} \right|}{N}$$

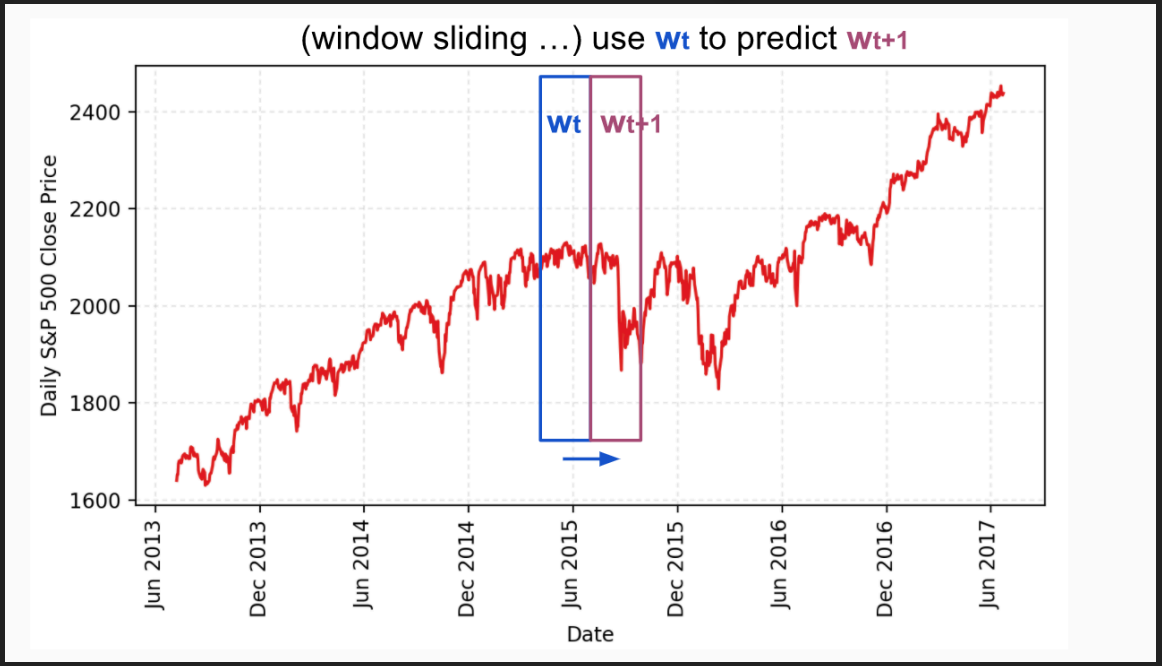
$$R = \frac{\sum_{t=1}^{N} (y_t - \overline{y_t})(y_t^* - \overline{y_t^*})}{\sqrt{\sum_{t=1}^{N} (y_t - \overline{y_t})^2 (y_t^* - \overline{y_t^*})^2}}$$

$$Theil\ U = \frac{\sqrt{\frac{1}{N}\sum_{t=1}^{N} (y_t - y_t^*)^2}}{\sqrt{\frac{1}{N}\sum_{t=1}^{N} (y_t)^2} + \sqrt{\frac{1}{N}\sum_{t=1}^{N} (y_t^*)^2}}$$

Predictive accuracy performance

How can we predict future stock price based on historic data

- We can understand the stock prediction issue by modeling the stock S&P 500 prices. The stock prices is a time series of length N, defined as $p_0, p_1, \ldots, p_{N-1} p_0, p_1, \ldots, p_{N-1}$ in which *pi* is the close price on day *i*, $0 \leq i < N_0$. Imagine that we have a sliding window of a fixed size w (aka input size) and every time we move the window to the right by size w, so that there is no overlap between data in all the sliding windows.



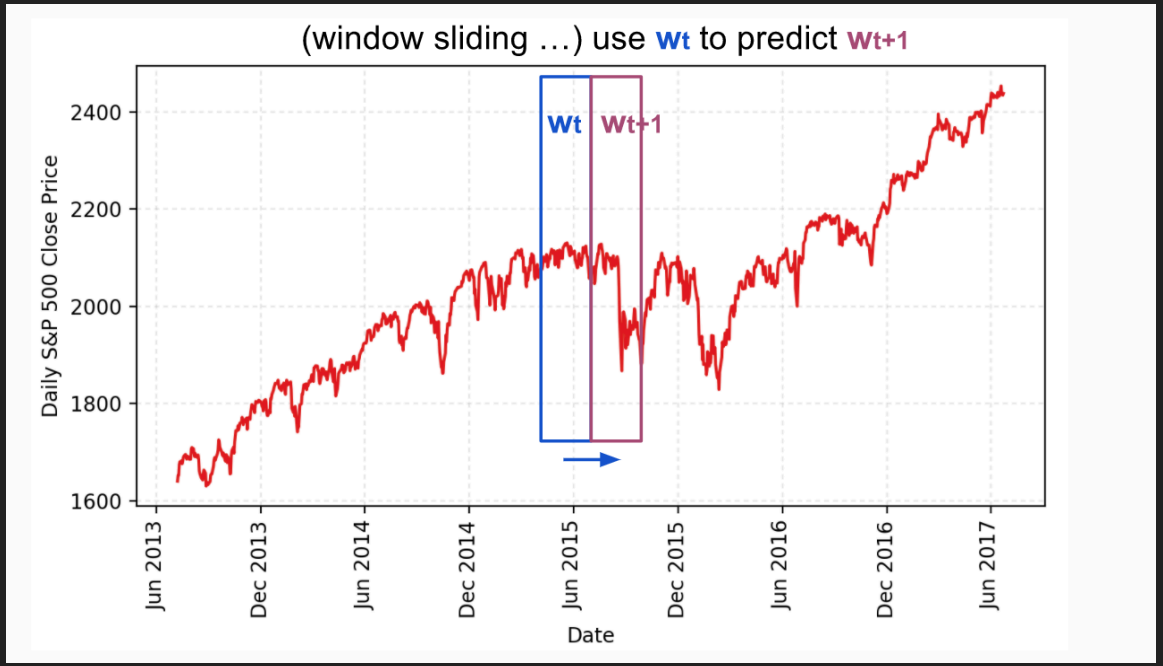
How can we predict future stock price based on historic data

use

- *W0=(p0,p1,…,pw−1)*
- *W1=(pw,pw+1,…,p2w−1)…*
- *Wt=(ptw,ptw+1,…,p(t+1)w−1)*

to predict the prices in the following window *wt+1:*

- *Wt+1=(p(t+1)w,p(t+1)w+1,…,p(t+2)w−1)*

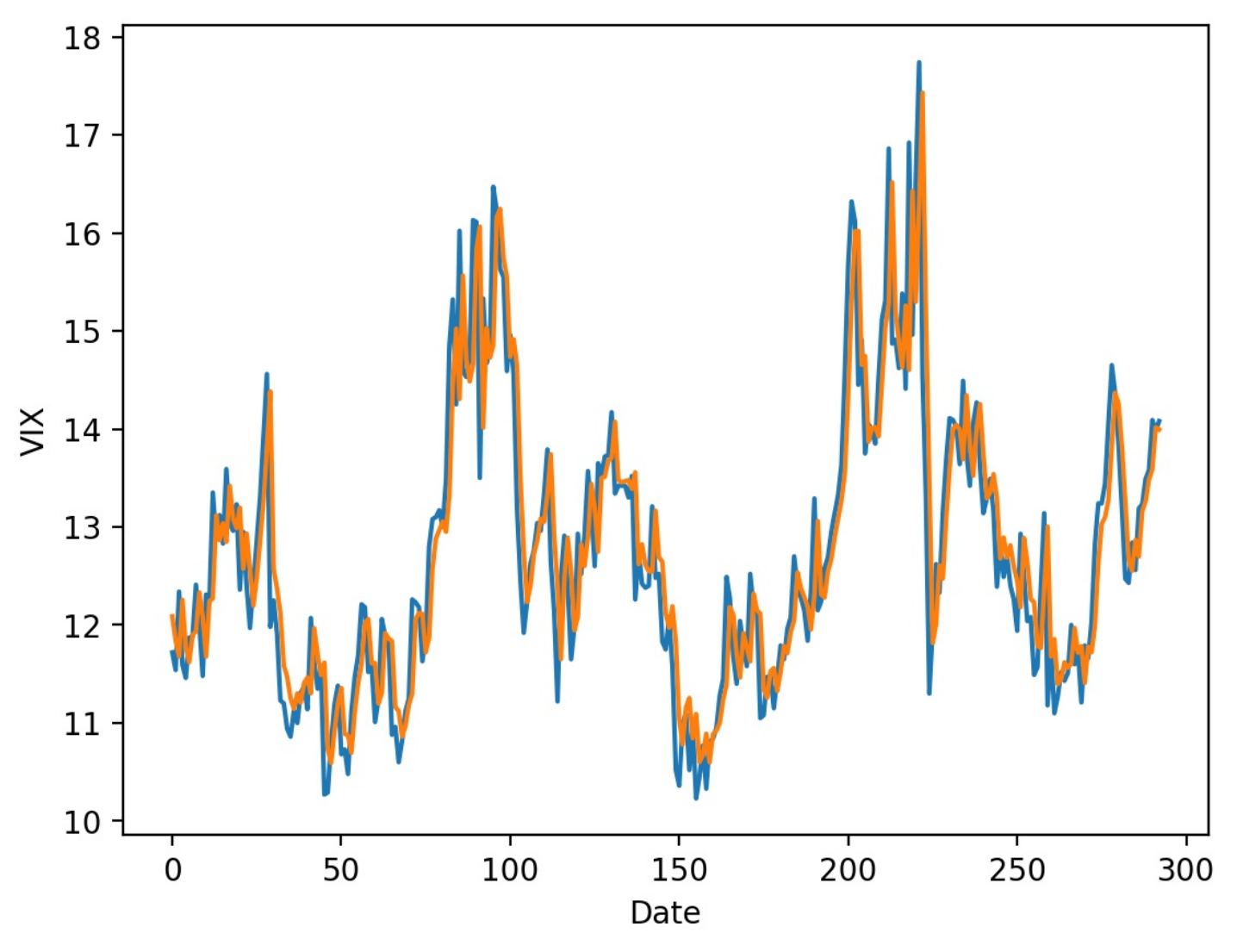


LSTM*2 +linear dense layer

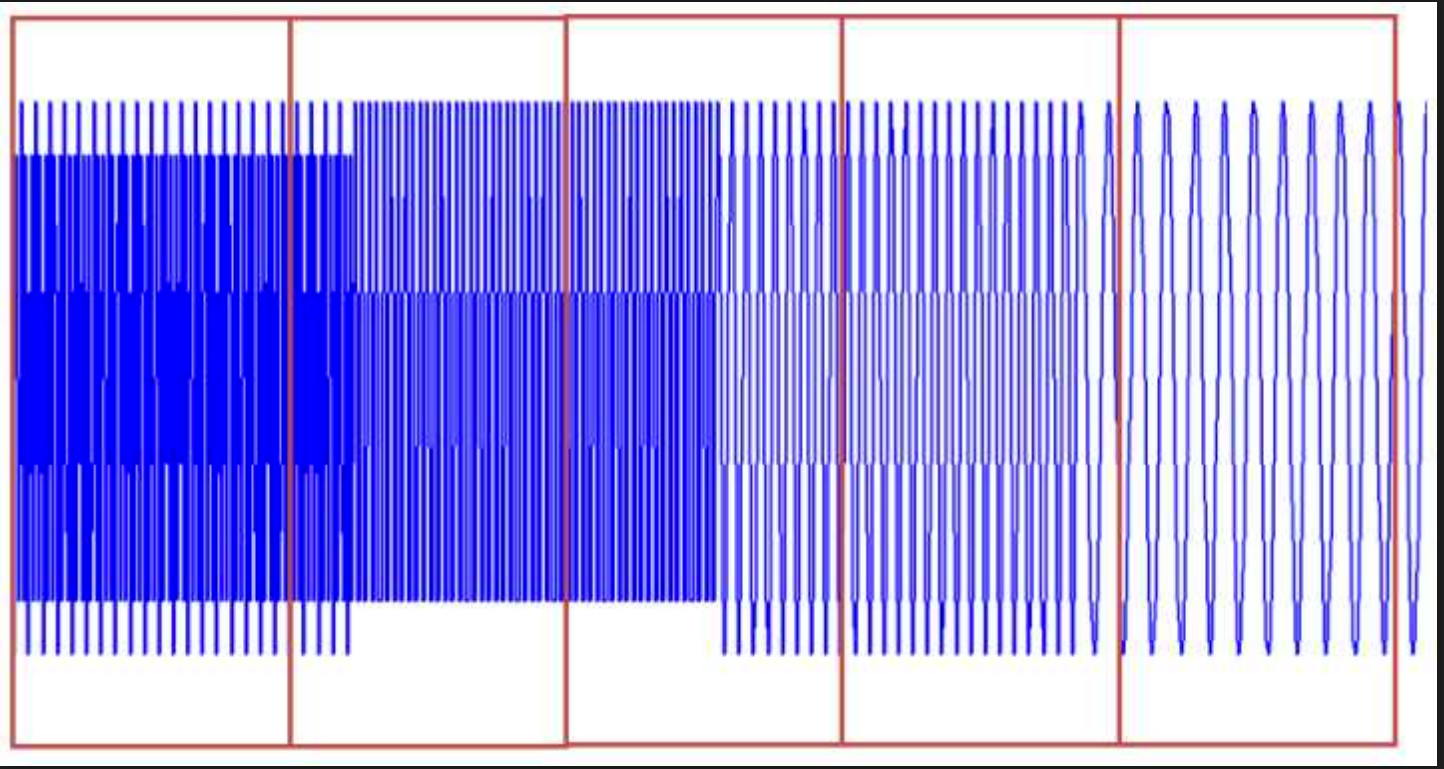
batch_size=512, nb_epoch=50, validation_split=0.05, verbose=1

- RNN: Variable length input
- CNN: Consistent length(image)

Input .csv: 01.03.2005-09.26.2016

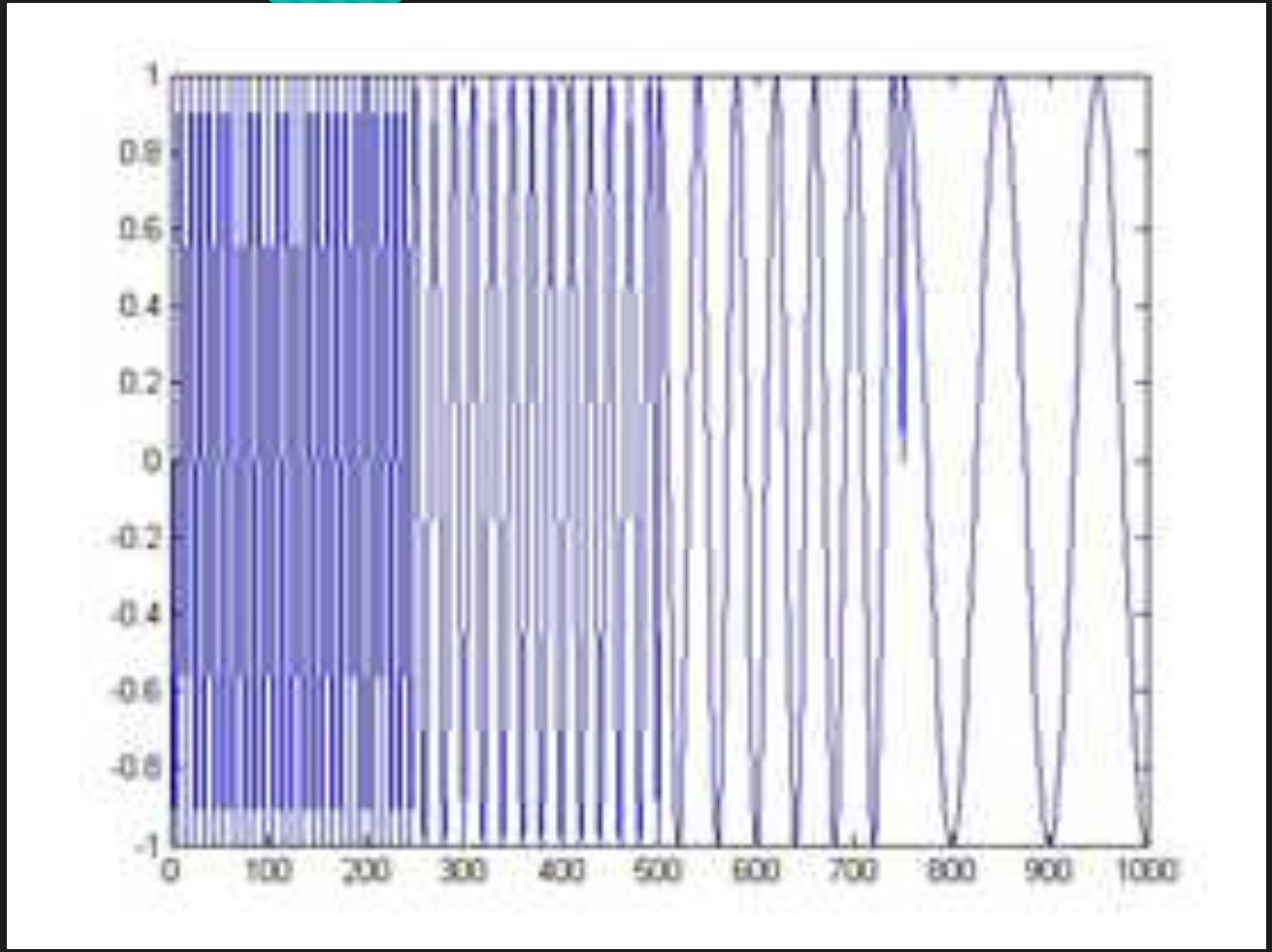


Fourier Transform



Window too large->time
Window too small->frequence

Wavelet Tranform



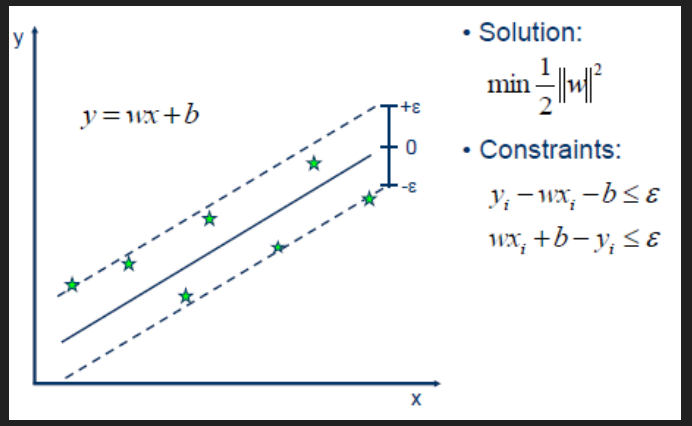
Scale
Transform

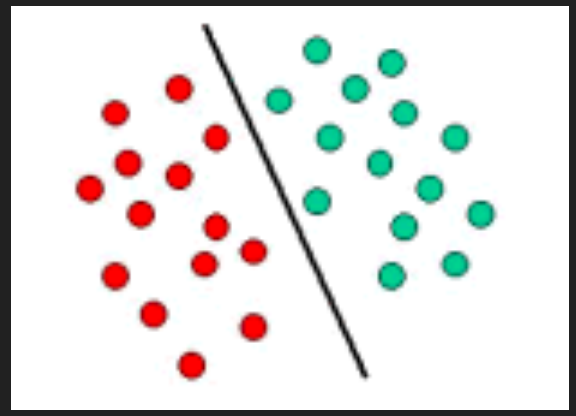
11.20

- SVM with stock price
- SVM with textual information
- Recap

SVM

- Support Vector Machines are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. A schematic example is shown in the illustration below. In this example, the objects belong either to class GREEN or RED. The separating line defines a boundary on the right side of which all objects are GREEN and to the left of which all objects are RED. Any new object (white circle) falling to the right is labeled, i.e., classified, as GREEN (or classified as RED should it fall to the left of the separating line).
- Support Vector Regression
- Support Vector Classification





Support Vector Classification

- Traditionally, we use linear statistical time series models such as ARIMA. However, the variance underlying the movement of stocks and other assets makes linear techniques suboptimal, and non-linear models like ARCH tend to have lower predictive error.
- We focus on a specific machine learning technique known as Support Vector Machines (SVM). Our goal is to use SVM at time t to predict whether a given stock's price is higher or lower on day t +m
- Reference: *Predicting Stock Price Direction using Support Vector Machines* from Princeton

Input

- Specific Stock Type as suggested
- The recent price volatility and momentum of the individual stock(stock price) and of the technology sector(index).
- Four parameters to the model - the recent price volatility and momentum of the individual stock and of the technology sector
- These parameters are calculated using daily closing prices for each stock from the years 2007 through 2014

Features

- Stock price volatility. This is an average over the past n days of percent change in the given stock's price per day
- Stock Momentum. This is an average of the given stock's momentum over the past n days. Each day is labeled 1 if closing price that day is higher than the day before, and −1 if the price is lower than the day before.
- Index volatility. This is an average over the past n days of percent change in the index's price per day.
- Index Momentum This is an average of the index's momentum over the past n days. Each day is labeled 1 if closing price that day is higher than the day before, and −1 if the price is lower than the day before.

How to choose n?

- Different n for Stock and Index , n1 for the index, n2 for the specific stock
- $n1,n2 \in \{5,10,20,90,270\}$
- These represent one week, two weeks, one month, one quarter
- Combination of n1 and n2 to build the features
- Feature calculation starting from begins since max(n1,n2)+1 day

Which day to make prediction on?

- We are calculating the features to try and predict the price direction m days in the future, where $m \in \{1,5,10,20,90,270\}$. As a result we skip the last m dates since we do not have the price m days after them. There are a total of 2014 trading days between 2007 and 2014, so we have a total of $2014-d-m$ days.

Cross validation

- We then split X and Y into the 70% training and 30% testing sets as, which we call Xtrain, ytrain, Xtest, ytest. We supply the feature vectors Xtrain as well as their corresponding output vectors ytrain to the SVM model. This is the training phase. We then supply only the testing feature vectors Xtest and have the model predict their corresponding output vectors. We then compare this output to ytrain.

Calculate features

```
def calcMomentum(numDays, priceArray):
    global daysAhead
    # now calculate momentum
    momentumArray = []
    movingMomentumArray = []
    for i in range(1, numDays + 1):
        movingMomentumArray.append(1 if priceArray[i] > priceArray[i-1] else -1)
    momentumArray.append(np.mean(movingMomentumArray))
    for i in range(numDays+1, len(priceArray) - daysAhead):
        del movingMomentumArray[0]
        movingMomentumArray.append(1 if priceArray[i] > priceArray[i-1] else -1)
        momentumArray.append(np.mean(movingMomentumArray))
```

Find technology company

```
def makeModelAndPredict(permno, numDays, sectorVolatility, sectorMomentum, splitNumber):
    global df
    global daysAhead
    # get price volatility and momentum for this company
    companyData = df[df['PERMNO'] == permno]
    companyPrices = list(companyData['PRC'])

    volatilityArray = calcPriceVolatility(numDays, companyPrices)
    momentumArray = calcMomentum(numDays, companyPrices)

    splitIndex = splitNumber - numDays
```

Permno is the unique code, use this to find the specific technology company

Make models

```
X_train = np.array(X[0:splitIndex]).astype('float64')
X_test = np.array(X[splitIndex:]).astype('float64')
y_train = np.array(Y[0:splitIndex]).astype('float64')
y_test = np.array(Y[splitIndex:]).astype('float64')

# fit the model and calculate its accuracy
rbf_svm = svm.SVC(kernel='rbf')
rbf_svm.fit(X_train, y_train)
score = rbf_svm.score(X_test, y_test)
```

Splite the input
Choose rbf kernel
Gaussian) radial basis function kernel

Result. M=1

- For m =1

```
& 5 & 5 & 0.503945 & 0.508632 & 0.532537 & 0.471448 \\

& 5 & 10 & 0.501211 & 0.501992 & 0.552457 & 0.468792 \\

& 5 & 20 & 0.504179 & 0.507968 & 0.547145 & 0.463479 \\

& 5 & 90 & 0.505273 & 0.508632 & 0.544489 & 0.463479 \\

& 5 & 270 & 0.503828 & 0.507304 & 0.545817 & 0.459495 \\

& 10 & 5 & 0.499253 & 0.496658 & 0.532086 & 0.467914 \\

& 10 & 10 & 0.494571 & 0.494024 & 0.556441 & 0.456839 \\
```

Result m= 5

```
& 5 & 5 & 0.523757 & 0.530708 & 0.618158 & 0.411215 \\
& 5 & 10 & 0.525053 & 0.530040 & 0.588785 & 0.452603 \\
& 5 & 20 & 0.528548 & 0.534713 & 0.579439 & 0.403204 \\
& 5 & 90 & 0.526310 & 0.529372 & 0.592790 & 0.401869 \\
& 5 & 270 & 0.530001 & 0.538051 & 0.598131 & 0.401869 \\
& 10 & 5 & 0.524984 & 0.524194 & 0.598118 & 0.419355 \\
& 10 & 10 & 0.521872 & 0.528037 & 0.588785 & 0.427236 \\
& 10 & 20 & 0.521165 & 0.528037 & 0.590120 & 0.403204 \\
& 10 & 90 & 0.515668 & 0.527370 & 0.596796 & 0.404539 \\
& 10 & 270 & 0.525563 & 0.534045 & 0.599466 & 0.404539 \\
& 20 & 5 & 0.531215 & 0.532698 & 0.588556 & 0.435967 \\
& 20 & 10 & 0.527700 & 0.535183 & 0.587280 & 0.441137 \\
& 20 & 20 & 0.533103 & 0.535381 & 0.610147 & 0.419226 \\
& 20 & 90 & 0.531061 & 0.536716 & 0.612817 & 0.380507 \\
```

m= 20

```
& 5 & 5 & 0.553775 & 0.559946 & 0.658038 & 0.336512 \\
& 5 & 10 & 0.555658 & 0.566757 & 0.668937 & 0.331063 \\
& 5 & 20 & 0.550809 & 0.555177 & 0.686649 & 0.325613 \\
& 5 & 90 & 0.546642 & 0.564033 & 0.663488 & 0.325613 \\
& 5 & 270 & 0.568040 & 0.580381 & 0.653951 & 0.335150 \\
& 10 & 5 & 0.549827 & 0.551440 & 0.663923 & 0.336077 \\
& 10 & 10 & 0.557020 & 0.566757 & 0.688011 & 0.325613 \\
& 10 & 20 & 0.545560 & 0.555858 & 0.688011 & 0.324251 \\
& 10 & 90 & 0.545560 & 0.560627 & 0.664850 & 0.324251 \\
& 10 & 270 & 0.575373 & 0.581063 & 0.713896 & 0.322888 \\
& 20 & 5 & 0.554365 & 0.556328 & 0.671766 & 0.321280 \\
& 20 & 10 & 0.554802 & 0.558011 & 0.680939 & 0.325967 \\
& 20 & 20 & 0.548325 & 0.556540 & 0.697548 & 0.324251 \\
```

M=90

```
It_index is deprecated, pls use .sort_values(by=...)
& 5 & 5 & 0.555679 & 0.570030 & 0.849398 & 0.251506 \\
& 5 & 10 & 0.554571 & 0.545934 & 0.849398 & 0.256024 \\
& 5 & 20 & 0.550053 & 0.551205 & 0.849398 & 0.222892 \\
& 5 & 90 & 0.492470 & 0.491717 & 0.849398 & 0.203313 \\
& 5 & 270 & 0.572821 & 0.602410 & 0.814759 & 0.161145 \\
& 10 & 5 & 0.551415 & 0.562974 & 0.848255 & 0.239757 \\
& 10 & 10 & 0.565556 & 0.574548 & 0.849398 & 0.236446 \\
& 10 & 20 & 0.567328 & 0.588102 & 0.849398 & 0.201807 \\
& 10 & 90 & 0.497387 & 0.520331 & 0.849398 & 0.194277 \\
& 10 & 270 & 0.571536 & 0.597139 & 0.774096 & 0.165663 \\
& 20 & 5 & 0.549760 & 0.566256 & 0.852080 & 0.221880 \\
& 20 & 10 & 0.556845 & 0.563456 & 0.847095 & 0.244648 \\
& 20 & 20 & 0.570030 & 0.571536 & 0.849398 & 0.225904 \\
```

- One notable result is that the SVM model is only tenths of a percentage point better than simple random guessing when it comes to predicting price direction one day ahead (m = 1). This has several important implications. First, it strongly reinforces the Efficient Markets Hypothesis. If a model that incorporates several types of historical data, including some features like momentum that economists have demonstrated are present in stock price data, is not able to do better than a coin flip when it comes to predicting the next day's price direction, then this is very strong evidence that prices follow a random walk. Prices that already reflect available information will change only based on new information, so tomorrow's price direction will be dependent only on new information that arrives tomorrow. A model like ours which analyzes only historical data should not be able to predict the price direction, as all this historical data should already be incorporated into the price. Therefore, our model's difficulty in predicting the next day's stock price supports EMH.

EMH

- Much economic research has been conducted into the Efficient Markets Hypothesis theory, which posits that stock prices already reflect all available information [18] and are therefore unpredictable. According to the EMH, stock prices will only respond to new information and so will follow a random walk. If they only respond to new information, they cannot be predicted. That the stocks follow a random walk is actually a sign of market efficiency, since predictable movement would mean that information was not being reflected by the market prices.

Improvement may be done

- Imputation of the missing data for the first max(n1,n2) days
- More features or more data
- Kernel selection

Stock Market prediction using news headlines

- Sentiment Analysis
- NTLK: NLTK is a leading platform for building Python programs to work with human language data
- A SentimentAnalyzer is a tool to implement and facilitate Sentiment Analysis tasks using NLTK features and classifiers, especially for teaching and demonstrative purposes
- Predict the Dow Jones Industrial Average (DJIA) with a top 25 of news headlines extracted from Reddit

```
Training done.
0.5079365079365079
[ 0.50649351  0.52631579  0.50666667  0.50666667  0.50666667]
Avg = 0.510561859193
Cross validation done.
Done.
```

- For every instance, average sentiment of the 25 news headlines was calculated using NLTK's Sentiment Intensity Analyzer. The compound sentiment value was taken as sentiment score. The average sentiment was used to train a support vector machine (SVM) with a linear kernel using 70% of the dataset as training set. The remaining part of the dataset was thereafter used in order to predict whether the DJIA went up or went down. Five-fold cross-validation resulted in an average accuracy of 51.1% on the test set. Given that 50.8% of the labels in the test set had the value '1' (stock stays the same or increases), the SVM seemed to perform at chance level.

Improvements can be done

- Only use news that are related to the company we want to analyze.
- Combine with the index and stock price features as we did in the first SVM.

A recap of what I have been done

- 1. Get familiar with TensorFlow and Keras.
- 2. Learnt concepts of finance.
- 3. Tested on several LSTM models
  - LSTM Neural Network for Time Series Prediction
  - Deep-Learning-Time-Series-Prediction-using-LSTM-Recurrent-Neural-Networks
  - US Stock Market Prediction by LSTM
- 4. Tested on the supervised learning-SVM models.
- 5. Set up the working environments
- 6. Settle down on the strategy and framework

Deep Learning In Financial Engineering

- 1. What has been done
- 2. What to do this semester
- 3. TRUST-TECH in LSTM
- More SVM
- 4. Paper

What has been done

- 1. Setup
  - Setup working environment
  - Setup AWS IPython Notebook
- 2. Skills
  - Acquired needed skills including Tensorflow, Keras and so on
- 3. Models
  - LSTM Neural Network for Time Series Prediction
  - Deep-Learning-Time-Series-Prediction-using-LSTM-Recurrent-Neural-Networks
  - US Stock Market Prediction by LSTM
  - Wavelet Transform + Stacked Auto-encoder + Long-short term memory
  - Support Vector Machine (textual information)
  - Main framework

What to do this semester

- 1. Clean up the code
- 2. Create uniform interfaces between different weak predictors
- 3. Built commander-soldier main framework
- 4. Evaluate prediction results
- 5. Modify parameters to get desired result
- 6. Finish paper.

TRUST-TECH for LSTM

- Applied TRUST-TECH in RNN (back propagation)
  - Paper :Wang, B. D., Hsiao-Dong Chiang. 2011."ELITE: Ensemble of Optimal, Input-Pruned Neural Networks Using TRUST-TECH."IEEE Transactions on Neural Networks22(1): 96-109.

Web Scraping

- Scrape the historical archives of a web financial blog in order to get for each post the following information: **date**, **keywords**, **text**. The date of the post is fundamental to store the time information; keywords are necessary all the same for a simple reason. The text data we scrape is going to be full of HTML tags. The XPath language (which we'll make extensive use of) has been developed for the specific purpose of easily getting information from a web page and getting rid of the tags in the first place. Nevertheless the process is not always smooth and during the post text selection we may happen to lose some tagged words, which in general are the most meaningful (companies' or people's names). It happens that this words are often the keywords or tags related to the post, thus they are easily reachable with a proper XPath query.
- Save all this information in a JSON file.
- Read the JSON file with Pandas and preprocess the text with NLTK (Natural Language ToolKit) and BeautifulSoup.
- Perform Sentiment Analysis on the clean text data in order to get sentiment scores for each day.
- Generate a final Pandas DataFrame and correlate it with stocks prices to test our hypothesis
- Repeat for multiple blogs with multiple threads

Data cleaning

- read JSON in Pandas DataFrame
- unlist all the entries (XPath queries return Python lists)
- convert dates to datetime
- join **keywords** and **body** columns in one text column and drop the first 2
- The first result must be a Pandas DataFrame with 2 columns: **date** of the post and **text** of the post.
- turn all the text to lowercase
- get rid of all the HTML tags from **text** using BeautifulSoup
- tokenize the text and get rid of stop words using NLTK
- return clean text in form of a list of words

Compute Sentiment Score

- Get a dictionary of Positive and Negative words and then count how many of each occur respectively in each post. Then get a Sentiment Measure out of that.
- Send it to the Support Vector Machine

About Paper

- Focus:
  - Combined Machine learning methods
  - TRUST-TECH Application
  - NLP integration results
  - Benchmark Results

TRUST-TECH Assisted RNN-based Predictor

- **Background**
- In the last semester, we got familiar with TensorFlow and Keras and learnt concepts of finance. Also, tests datasets were run on several LSTM models including LSTM Neural Network for Time Series Prediction, Deep-Learning-Time-Series-Prediction-using-LSTM-Recurrent-Neural-Networks and US Stock Market Prediction by LSTM. Apart from the that, supervised learning SVM models have been also introduced to this project.
- The main innovation for this semester is to introduce the TRUST-TECH into the deep learning neural networks training process, in this context, it will be the backpropagation optimization.The main techniques utilized here is the PSO-guided TRUST-TECH.

Methodology

- **PART 1: Overview of Steps: PSO-guided TRUST-TECH Assisted Predictor**
  - 1. A basic RNN architecture is constructed with specified parameters and coefficients before the proposed method is applied.
  - 2. Construct the PSO-guided TRUST-TECH Assisted Predictor

Methodology

- **PART 2: Construction of the TRUST-TECH Enhanced Ensemble Predictors**
  - The goal here is training RNN Using PSO-Guided TRUST-TECH. Given the input-output pairs(x1,t1),(x2,t2)…to (xn,tn), the training task can be reorganized as an optimization problem with =(i+2)*h+1 dimensions, where i represents the number of input nodes and h represents the number of hidden nodes.

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

Methodology

- We want to minimize the above MSE function, where Yi is a function consisting of input data and weights w of the connected nodes in RNN.
- TRUST-TECH solve an optimization problem by first defining a dynamical system such that the stable equilibrium points(SEPs) in the system have one-to-one correspondence with local optimal solutions of the optimization problem, in which way, we convert the problem to finding multiple stability regions in the defined dynamical system, each of which has a distinct SEP. Then an SEP can be computed with the trajectory method.

The defined dynamical system can be defined as the generalized gradient system

$$\frac{dw}{dt} = -gradient(MSE) = -R(W)^{-1} * \nabla MSE$$

R(w): I for the Back Propagation.

Algorithm

- Initialization: a local optimal weight vector w0, a set W of next-tier SEPs started as W={w0}
- For each search direction
  - Search for exit point along this direction
  - If we along the search direction is found, move forward into the stability region of neighboring SEP for initial point A.
  - Start from the A, using local optimizer to get tier-1 SEP, denoted as w1
  - Update W to include w1.
- Get W of SEP for the generalized gradient system.
- One thing to remember is that the direction selection, as told in class, we are using eigenvectors of the objective Hessian at the SEP.

Potential Problems in RNN

- In RNN, we have Backpropagation Through Time(BPTT), instead of the traditional ANN BP, we also need to incorporate the time variants (still working on that). The Vanish Gradient in RNN may posed potential hazards to the problem as well.

Milestones

- Milestone 1: Finish the prototype in two weeks.
- Milestone 2: Debug in following one week.
- Milestone 3: Report in following one week.

Expected Result

The final model will be of better performance than the basic RNN parameters.
No gradient vanishing problem will pose great threat to this project.