# Technical Analysis

Quan Chen, Reinaldo Perez, Hoormazd Rezaei, Haseeb Khan

June 11, 2018

**Abstract**

We build an automatic approach to technical pattern extraction by smoothing and kernel regression in this paper. We use this method to most liquid 100 U.S. stocks to gauge the effectiveness of technical analysis. Technical indicators can develop profitable strategy given the comparison between the unconditional empirical distribution of daily stock returns and the conditional distribution—conditioned on specific technical indicators. We also develop deep learning models in order to extract appropriately the patterns within the data and tune for them appropriately.

## 1 Introduction

Technical Analysis is defined as an analysis methodology for forecasting the direction of prices through the study of past market data, primarily price and volume. Technical Analysis was the tool of choice and evolved a fairly advanced methodology before quantitative and algebraic techniques became the primary mechanism behind technical decision making [Nee99]. To some extent technical analysis is known as unreliable. However, several paper suggest that technical analysis is effective in obtaining valuable information from market prices despite its methods[NP84] [PW88]. Technical analysts assume that the past prices can be used to forecast future returns.[FMT99].

## 2 Description of the Strategy

We aim to apply technical analysis to identify patterns in the time series of prices. By searching for nonlinear patterns from noisy data, we recognize that some price movements are important while others are merely white noise. We can identify signals by our eyes quickly and accurately while computer could not until recently. However, smoothing estimators are well designed for this task because they can identify nonlinear patterns by averaging out the noise. Therefore, we propose using these estimators to replicate and assist human in extracting regularities in historical price series [LMW00].

We provide a brief review of smoothing estimators and describe in detail the specific smoothing estimator we use in our analysis: kernel regression. Our algorithm for automating technical analysis is described in latter section.

### 2.1 Smoothing Estimators and Kernel Regression

We assume that prices $\{P_t\}$ satisfy the following expression [Sim96]:

$$P_t = m(X_t) + \epsilon_t \tag{1}$$

where $m(X_t)$ is an arbitrary fixed but unknown nonlinear function of a state variable $X_t$ and $\epsilon_t$ is white noise.

Above nonlinear relation can be estimated by smoothing, in which observational errors are reduced by averaging out the noise. Besides having some statistical optimality properties, smoothing estimators are motivated by similarity to the way human identifies patterns from noisy data.

Therefore, they are ideal for our purposes. For any arbitrary x, a smoothing estimator of m(x) may be expressed as:

$$m_x = \frac{1}{T}\sum_{t=1}^{T}\omega_t(x)P_t \tag{2}$$

For the kernel regression estimator, the weight function $\omega_t$ is constructed from a probability density function K(x), also called a kernel:

$$K(x) \geq 0, \int K(u)du = 1 \tag{3}$$

By rescaling the kernel with respect to a parameter h, we can change its spread; that is, let

$$K_h(u) = \frac{1}{h}K(u/h), \int K_h(u)du = 1 \tag{4}$$

For this paper, we shall use the most popular choice of kernel, the Gaussian kernel:

$$K_h(x) = \frac{1}{h\sqrt{2\pi}}e^{-x^2/2h^2} \tag{5}$$

By choosing the appropriate h in the above equation, we can optimize the approximation of m(). We use cross-validation as the loss function for h. The cross validation function is given as:

$$CV(h) = \frac{1}{T}\sum_{t=1}^{T}(P_t - m_{h,t})^2 \tag{6}$$

The following figure is an example of stock price time series after applying the smoothing function.

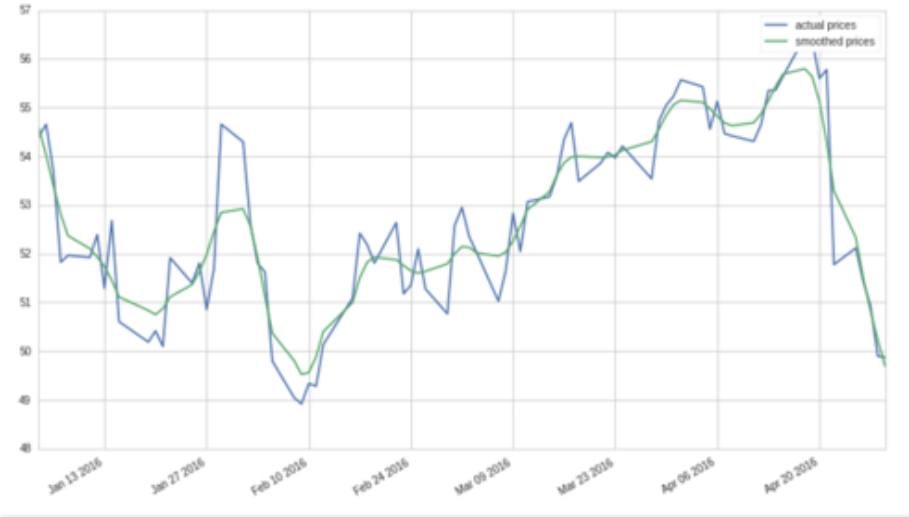


Figure 1: Microsoft (MSFT) smoothing from January 1st 2016 through May 1st 2016, Note: The smoothing estimation is not optimized.

## 2.2 Automating Technical Analysis

Given kernel estimators, we can now build an algorithm for automating the detection of technical patterns. Our algorithm is shown below:

1. Define each technical pattern in terms of its geometric properties, for example, local extrema maxima and minima.
2. Construct a kernel estimator of a given time series of prices so that its extrema can be determined numerically.
3. Analyze kernel estimator for occurrences of each technical pattern.

In Section 2.2.1, we propose definitions of 10 technical patterns based on their extrema. In Section 2.2.2, we describe a specific algorithm to identify technical patterns based on the local extrema of price series using kernel regression estimators

### 2.2.1 Definition of Technical Patterns

We choose eight most poplar technical patterns: Head and Shoulders (HS) Inverse Head and Shoulders (IHS) Broadening Top (BTOP) Broadening Bottom (BBOT) Triangle Top (TTOP) Triangle Bottom (TBOT) Rectangle Top (RTOP) Rectangle Bottom (RBOT) We select these patterns because we want to exemplify smoothing techniques in automating technical analysis.

Consider the systematic component m() of a price history $P_t$and suppose we have identified n local extrema, that is, the local maxima and minima, of Denote by $E_1, E_2, ..., E_n$ the n extrema and $t_1 t_2, ...t_n$ dates on which these extrema occur. Then we have the following definitions.

Definition 1 (Head-and-Shoulders) Head-and-shoulders (HS) and inverted head-and-shoulders (IHS) patterns are characterized by a sequence of five consecutive local extrema$E_1, E_2, ..., E_5$such that

$$HS = \begin{cases} E_1 \textit{ is a maximum} \\ E_3 > E_1, E_3 > E_5 \\ E_1 \textit{ and } E_5 \textit{ are within } 1.5 \textit{ percent of their average} \\ E_2 \textit{ and } E_4 \textit{ are within } 1.5 \textit{ percent of theiraverage}, \end{cases}$$

$$IHS = \begin{cases} E_1 \textit{ is a maximum} \\ E_1 > E_3, E_5 > E_3 \\ E_1 \textit{ and } E_5 \textit{ are within } 1.5 \textit{ percent of their average} \\ E_2 \textit{ and } E_4 \textit{ are within } 1.5 \textit{ percent of theiraverage}, \end{cases}$$

Definition 2 (Broadening) Broadening tops (BTOP) and bottoms (BBOT) are characterized by a sequence of five consecutive local extrema $E_1, E_2, ..., E_5$ such that

$$BTOP = \begin{cases} E_1 \textit{ is a maximum} \\ E_5 > E_3 > E_1 \\ E_2 > E_4 \end{cases}$$

$$BBOT = \begin{cases} E_1 \textit{ is a minimum} \\ E_1 > E_3 > E_5 \\ E_4 > E_2 \end{cases}$$

Definition 3 (Triangle) Triangle tops (TTOP) and bottoms (TBOT) are characterized by a sequence of five consecutive local extrema $E_1, E_2, ..., E_5$ such that

$$TTOP = \begin{cases} E_1 \textit{ is a maximum} \\ E_1 > E_3 > E_5 \\ E_4 > E_2 \end{cases}$$

$$TBOT = \begin{cases} E_1 \textit{ is a minimum} \\ E_5 > E_3 > E_1 \\ E_2 > E_4 \end{cases}$$

Definition 4 (Rectangle) Rectangle tops (RTOP) and bottoms (RBOT) are characterized by a

sequence of five consecutive local extrema $E_1, E_2, ..., E_5$ such that

$$RTOP = \begin{cases} E_1 \text{ is a maximum} \\ \text{Tops are within 0.75 percent of their average} \\ \text{Bottoms are within 0.75 percent of their average} \\ \text{Lowest top } > \text{ highest bottom} \end{cases}$$

$$RBOT = \begin{cases} E_1 \text{ is a minimum} \\ \text{Tops are within 0.75 percent of their average} \\ \text{Bottoms are within 0.75 percent of their average} \\ \text{Lowest top } > \text{ highest bottom} \end{cases}$$

Definition 5 (Double Top and Bottom) Double tops (DTOP) and bottoms (DBOT) are characterized by an initial local extremum $E_1$ and subsequent local extrema $E_a$ and $E_b$ such that

$$E_a = sup\{P^*_{t_k} : t^*_k > T^*_1.\ k = 2, 3, ...n\} \tag{7}$$

$$E_b = inf\{P^*_{t_k} : t^*_k > T^*_1.\ k = 2, 3, ...n\} \tag{8}$$

### 2.2.2 The Identification Algorithm

Our algorithm begins with a sample of prices $P_1, ..., P_T$for which we fit kernel regressions, one for each subsample or window from t to t+l+d-1, where t varies from 1 to T-l-d+1, and l and d are fixed parameters whose purpose is explained below.

We fit kernel regressions to rolling windows of data to narrow our focus to patterns that are within $l + d$ trading days which is the length of the window in our case. If we fit a single kernel regression to the whole dataset, many patterns of different length may be identified, and it is difficult to distinguish signal from noise in this scenario without adding constraints to patterns. Therefore, the length of the window is fixed at $l + d$, but kernel regressions are estimated for patterns in each window. The parameter $d$ assume that there may be a lag between the pattern realization and the time of pattern recognition. We require that the final extremum that completes a pattern occurs on day $t + l + 1$; hence $d$ is the number of days following the completion of a pattern that must pass before the pattern is recognized. In other words, the lag $d$ makes sure that the conditional returns are computed fully out-of-sample and without "look-ahead" bias.

Within each window, we estimate a kernel regression using the prices in that window, hence:

$$m_h(\tau) = \frac{\sum_{s=t}^{t+l+d-1} K_h(\tau - s)P_s}{\sum_{s=t}^{t+l+d-1} K_h(\tau - s)} \qquad t = 1, ..., t - l - d + 1 \tag{9}$$

After we have identified all of the local extrema in the window $[t, t + l + d - 1]$ , we can proceed to extract different technical patterns using the definitions above. Then we repeat the process for the next window $[t + 1, t + l + d]$ and continues until the end of the sample is reached at the window $[T - l - d + 1, T]$.

# 3 Data

We use Quantopian dataset of stock prices. The data is cleaned and the close price is adjusted for corporate actions. Using the platform we are able to select based on liquidity. Selection based on liquidity is to maximize pattern detection.

We collect interest rate data from Federal Reserve Economic Data [1] in order to calculate excess returns.

[1] Federal Reserve Economic Data (FRED)

### 3.1 Training Set

We use data from 100 most liquid stocks from 2002-1-1 through 2010-12-31 as our training data. We compute returns based on waiting and holding period after we have identified technical patterns applying the automated identification algorithm.

### 3.2 Test Set

We choose out-of-sample data from the 100 and 1000 most liquid stocks from 2011-1-1 through 2017-12-31. Using optimized parameters, we compute return distribution for the 4 outperforming patterns.

## 4 Method

### 4.1 Training

Let us define the following parameters:

- Waiting Period, $w$: The amount we wait after pattern recognition before taking a position.
- Holding Period $h$: The amount we hold our position after entering one.

Sharpe ratio is defined to be the mean of returns over standard deviation of them (a proxy for volatility). Here, we use a modified version of Sharpe ratio to pick best performing patterns. Our Sharpe ratio differs from an actual one, by that it only takes into accounts returns after pattern recognition. However, managing a portfolio on patterns, one may have an excess return of zero if none is observed. It is worth mentioning that all analysis is done on excess returns. After pattern recognition, we go long on bullish patterns, and we short the bearish ones.

During the training we take the following steps:

- Firstly, we run the kernel regression on the data in order to remove noise. Smoothing is optimized to minimize cross-validation loss.
- Secondly, we locate extrema on smoothed prices, and consequently on actual prices.
- Next, we detect patterns for each stock and calculate the returns for different combinations of waiting period, $w$, and holding period, $h$. For each stock, we save the best performing combination and its return.
- Finally, we choose best performing combination among all stocks.

### 4.2 Testing

Based on our internal version of Sharpe ratio, we pick patterns with a significantly high Sharpe ratio (more than 2) and test them out-of-sample. If patterns are able to achieve the same performance out-of-sample, we include them in our portfolio construction.

## 5 Results

### 5.1 Parameters

Results of optimal combinations of $w$ and $h$ are provided below.

| Pattern | Waiting Period (days) | Holding Period (days) |
|---|---|---|
| Head and Shoulders | 1 | 1 |
| Broadening Top | 1 | 1 |
| Rectangle Top | 1 | 1 |
| Triangle Top | 2 | 1 |
| Inverse Head and Shoulders | 1 | 1 |
| Broadening Bottom | 1 | 1 |
| Rectangle Bottom | 1 | 1 |
| Traingle Bottom | 2 | 1 |

Figure 2: Optimized waiting and holding periods

## 5.2 Best Performing Patterns

By using the optimized parameters, we compute the return distribution for all 8 patterns. The results show that Broadening Top (BTOP), Inverse Head and Shoulders (IHS), Broadening Bottom (BBOT), Head and Shoulders (HS) are the 4 most profitable patterns.

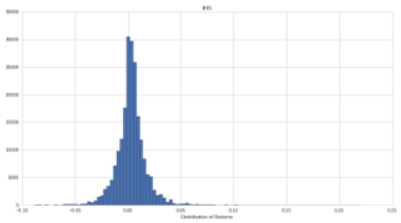

Figure 3: training IHS, Mean: 0.003169 inside SD: 0.013959 SR: 3.6040

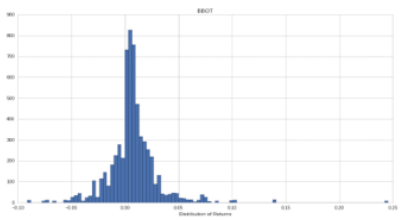

Figure 4: training BBOT, Mean: 0.006803 inside SD: 0.023195 SR: 4.6563

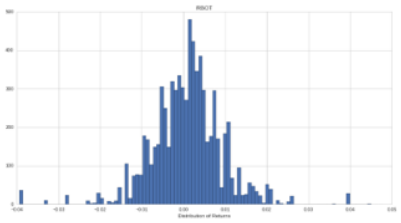

Figure 5: training RBOT, Mean: 0.000224 inside SD: 0.007862 SR: 0.4524

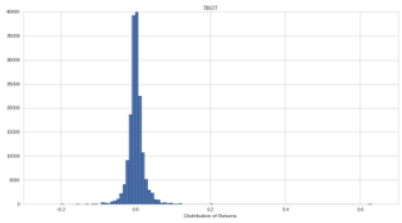

Figure 6: training TBOT, Mean: 0.002390 inside SD: 0.021694 SR:1.7494

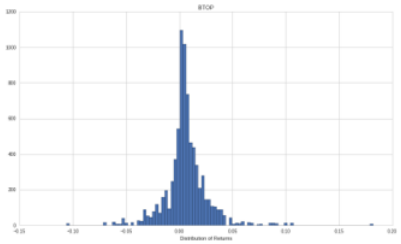

Figure 7: training BTOP, Mean: 0.006174 inside SD: 0.020361 SR: 4.6564

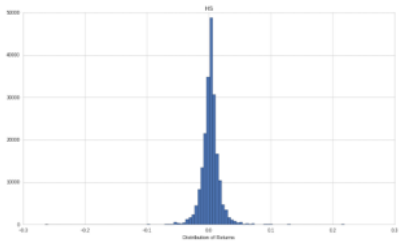

Figure 8: training HS, Mean: 0.002715 inside SD: 0.014904 SR: 2.8912

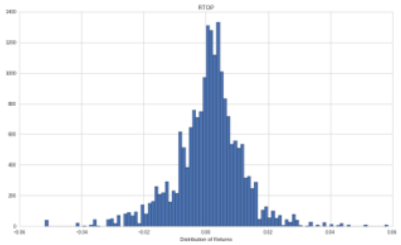

Figure 9: training RTOP, Mean: 0.000930
inside SD: 0.010806 SR: 0.45250

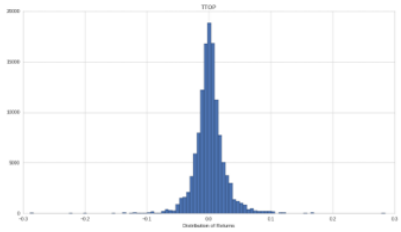

Figure 10: training TTOP, Mean: 0.001341
inside SD: 0.026575 SR: 0.8010

### 5.3 Out-of-sample Data

Testing the 4 best performing patterns on out-of-sample data, we are able to achieve the same performance. Results below strengthen our choice of patterns.

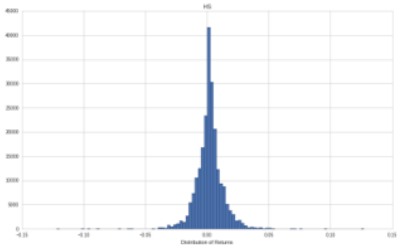

Figure 11: 100 HS, Mean: 0.001825
inside SD: .0.011382 SR: 2.5466

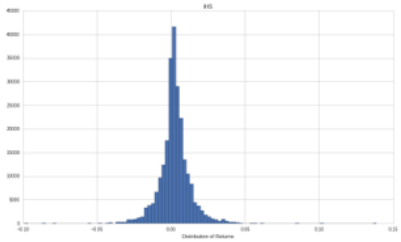

Figure 12: 100 IHS, Mean: 0.002277
inside SD: 0.010785 SR: 3.3525

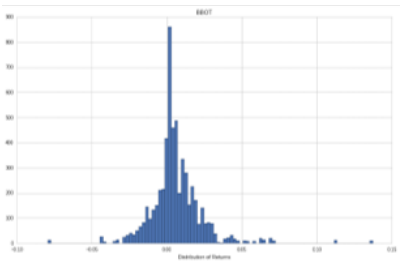

Figure 13: 100 BBOT, Mean: 0.003049
inside SD: 0.013959 SR: 5.5683

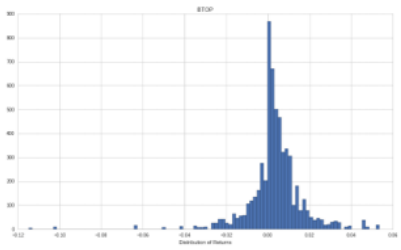

Figure 14: 100 BTOP, Mean: 0.002919 SD: 0.013236 SR: 3.99147

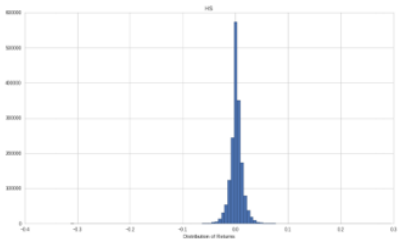

Figure 15: 1000 HS, Mean:0.002086
inside SD: .0.012589 SR: 2.6305

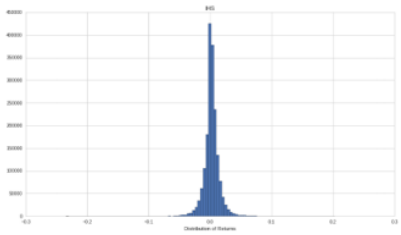

Figure 16: 1000 IHS, Mean: 0.002583
inside SD: .0.012463 SR: 3.2902

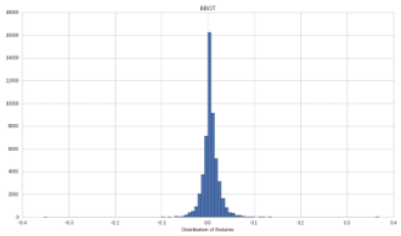

Figure 17: 1000 BBOT, Mean: 0.006314 inside SD: 0.020119 SR: 4.9824

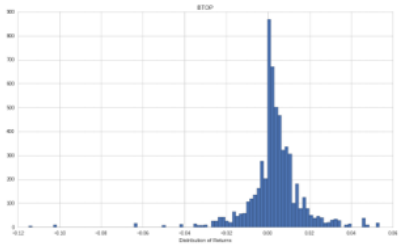

Figure 18: 100 BTOP, Mean: 0.004089 inside SD: 0.015237 SR: 4.2604

### 5.4 Portfolio Construction

As a result of train and test process, we pick the 4 following patterns: Broadening Top (BTOP), Inverse Head and Shoulders (IHS), Broadening Bottom (BBOT), and Head and Shoulders (HS).

We construct our portfolio as described below:

- If patterns are observed at day $t$, we wait according to optimal $w$ corresponding to each observed pattern. We then take position and hold it for optimal $h$. If multiple patterns are observed, we take the naive strategy of taking equal positions in each pattern ($\frac{1}{n}$ for each pattern, for $n$ bullish patterns).
- If no pattern is observed, our excess return for the future days related on day $t$ will equal zero.

Following above strategy, results are as below for the 4 best patterns.

| | |
|---|---|
| BBOT | 4.247 |
| BTOP | 4.3.215 |
| IHS | 3.413 |
| HS | 2.538 |

## 6 Discussion

As expected by the definition of technical patterns, and potentially as a consequence of survivorship bias, it is found that the best holding period for patterns is 1 day. Training over 9 years of data however, we find that only 4 patterns are able to generate high Sharpe ratios. The results hold for out-of-sample data as well. Constructing our portfolio on patterns BBOT, BTOP, IHS, and HS, we were able to achieve real Sharpe ratios of 4.247, 3.215, 3.413, and 2.538, respectively, in the out-of-sample data. These ratios were obtained by calculating the mean of returns for each trading day where at least one observation of a pattern occurred. We also included zero returns for the trading days where no observations occurred. In total, there were 1761 trading days in the out-of-sample data. From these data we derived the Sharpe Ration for each pattern. The high real Sharpe ratios in the out-of-sample data are surprisingly high. We expect outliers to be responsible, however, due to shortage of time we are not able fully study the issue. All being said, it is obvious that the 4 technical patterns, based on which we construct our portfolio, have significant predictive power.

## 7 Conclusion and Future Steps

We believe the following future steps may be beneficial to our study:

- Study how patterns may act in different market environments. For example, it can be analyzed which patterns occur more in volatile markets, and how we can adapt our portfolio construction to way in can take those behaviors into account.
- In our analysis, whenever multiple patterns are observed, we take the naive strategy. More approaches can be studies and compared here.

- Moreover, patterns can be studies among various categories of assets; financial sector and technology sector to name but two.

# 8 Deep Learning

Our paper proposes a deep learning method for reflexive and baseline market forces using a causal aspect dependent stock market price prediction. Our baseline idea is to let an extremely dense neural net extract a plethora of features from a daily priceline index of one stock. Then, after this data is normalized within given price values, the weights in the network are tensorized with respect to the normalized price feature values. In the second phase, we separate short term features from long term influences on stock pricing models, considering Bayesian semiparametric effects on Markov processes. Applying this technique to daily split adjusted company price data, experimental results show that we can achieve 61% correct prediction of the next day's closing price, with 4% MOE under nominal market circumstances for most companies comprising the S&P 500 index.

We modeled the solution using the TensorFlow$^{tm}$ deep learning Python framework based on 24-layer LSTM model. Our results demonstrate how a deep learning model trained on pricing data from daily open, close, high, low, and volume traded information can provide a valuable signal to an investment decision maker. Below are some basic results of our model for 25 companies using data for 3 years.

| **Company** | **LISTING** | **Stock** | **Training / Dev** | **Test** Corr closing prediction next day |
|---|---|---|---|---|
| Apple Corp | NYSE | AAPL | 74.23 % / 69.12% | 63.110% |
| Goldman Sachs | NYSE | GS | 75.19 % / 63.63% | 62.142% |
| JP Morgan | NYSE | JPM | 78.56 % / 68.91% | 62.776% |
| Google | NYSE | GOOGL | 72.62 % / 67.32% | 59.818% |
| Amazon | NYSE | AMZN | 76.55 % / 65.82% | 61.092% |
| Netflix | NYSE | NFLX | 78.15 % / 64.12% | 59.210% |
| Royal Dutch Shell | NYSE | RDS.A | 74.53 % / 65.67% | 58.179% |
| Chevron Corporation | NYSE | CVX | 75.43 % / 69.91% | 59.891% |
| Daimler AG | ETR | DAI | 76.92 % / 71.52% | 61.529% |
| Gazprom | MCX | GAZP | 73.15 % / 71.65% | 62.017% |

Figure 19: Training and Test Results with 10 stocks, using LSTM-24L

## 8.1 Neural Network LSTM Model

In order to predict appropriately the patterns within pricing and volumetric information, there is a need for advanced multivariate methods with a great degree of parametric and hyper-parametric flexibility that can also efficiently extract the underlying source patterns within the data and tune for them appropriately. Deep and Very Deep LSTM models are able to achieve this to a greater degree because of their ability to correctly perform the tensor decomposition necessary in order to identify, classify and correlate chronological data sequences. It must however, be borne in mind that deeper networks are much harder to train and are even harder to fine-tune. In order to train a Deep LSTM with real time 5 factor-pricing, one must trade volume information and incrementally adjust the model daily using the information loss-factor in preprocessing the data with a smoothing kernel - which we discussed earlier -or the data needs to be minimized. Because of this constraint, a smoothing function cannot be applied to the data in a manner similar to our regular technical analysis methodology. Additionally, the In-Network information loss factor has to be kept at a minimum, therefore, attempts to correct over-fitting by incorporating L1 and L2 dropout layers cannot be done. Furthermore, any attempts at incorporating max-pooling layers will lead to a loss in data resolution, and thus could not be used.

Because the task of tuning a very large number of parameters is time consuming, we tried a number of different optimization techniques, the results of which are displayed in the figure below.

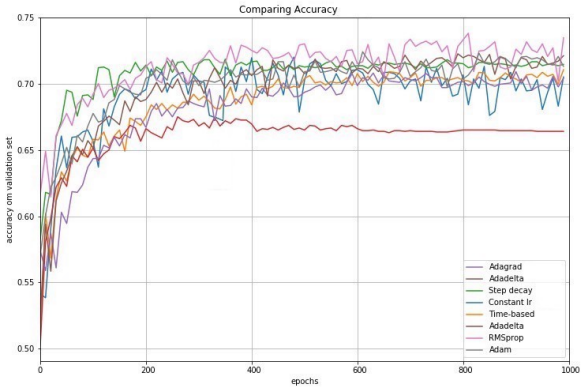


Figure 20: Performances of Different Learning Algorithms

For out activation function after we tried a number of different approaches we proposed and successfully utilized "scaled exponential linear units" (SELUs) to make our Deep LSTM network as self-normalizing. The SELU activation function is given by:

$$SELU(x) = \lambda\tau \begin{cases} x \ if \ x > 0 \\ ae^x - a \ if \ x < 0 \end{cases}$$

Using SELU we were able to dramatically improve the accuracy and precision of our model as well as reduce the degree of over fitting to the training data. The figure below displays the results of the 30 Layer SNN added to the LSTM architecture compared to other setups that we tested when iterated over the same large 20 year datasets. In Looking at the results we observe that a Deep LSTM architecture employing multiple SELU layers provides the best results for our model.

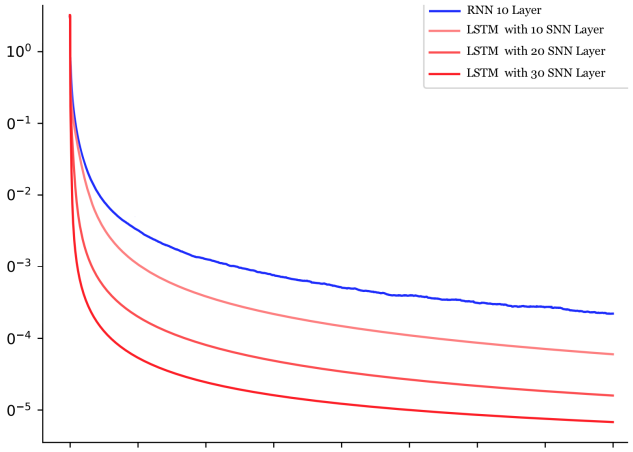


Figure 21: Performance of SNN Layers

We however observed limitations in the network that made it very sensitive to bad data. This essentially meant that the network data has to be cleaned meticulously before any attempts can be made to utilize the potential of the architecture.

### 8.2 Secondary Bayesian Neural Network for Ranging Price Field

Our model served its purpose as a quintessential proof of concept and is only the proverbial peak of the iceberg. It served as a benchmark for proving that it would be possible to structure a Deep Neural Net employing LSTM architecture with multiple SNN layers fully connected to RNN layers in order to serve as hyper-feature extractors, and that such an architecture can yield semi-dependable results in a test environments. Our hope for the future is to add a secondary BNN (Bayesian Neural Network) to the architecture in order to generate a skewed distribution for the next day's predicted price.

## 9 Contribution

Hoormazd, Reinaldo, and Quan worked on replication of Lo's paper, and portfolio construction based on it. Haseeb was responsible for Deep Learning section.

## References


[FMT99] Colin Fyfe, John Paul Marney, and Heather F.e. Tarbert. Technical analysis versus market efficiency - a genetic programming approach. *Applied Financial Economics*, 9(2):183–191, 1999.

[LMW00] Andrew Lo, Harry Mamaysky, and Jiang Wang. Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. 2000.

[Nee99] Christopher J. Neely. Using genetic algorithms to find technical trading rules: A comment on risk adjustment. *SSRN Electronic Journal*, 1999.

[NP84] Salih N. Neftci and Andrew J. Policano. Can chartists outperform the market? market efficiency tests for "technical analysis". *Journal of Futures Markets*, 4(4):465–478, 1984.

[PW88] Stephen W. Pruitt and Richard E. White. The crisma trading system. *The Journal of Portfolio Management*, 14(3):55–58, 1988.

[Sim96] Jeffrey S. Simonoff. Further applications of smoothing. *Springer Series in Statistics Smoothing Methods in Statistics*, page 252–274, 1996.