CS230

Deep Learning for Stock Price Forecasting

Kevin Li
Stanford University
kevinqli@stanford.edu

Glenn Yu
Stanford University
glennyu@stanford.edu

Abstract

In this project we explore the task of predicting the movement of stock prices. Given a 60-day input of prices of a stock, the task at hand is to predict whether the price will increase over the next 60 days. We trained an LSTM model, a deep LSTM model and a 1-D CNN model to tackle this task. All the deep learning models did slightly better (59% accuracy) than our baseline linear classification models (58% accuracy), but none did much better than the bias in the dataset alone (55%), suggesting that there is not much signal in predicting price movements using price data alone.

1 Introduction

The goal of this project is to build an automated system to trade stocks. Stock traders examine various patterns in stock market data and use well-known motifs to determine their trade decisions. We believe with a well-trained neural network, we believe we can automate these decision processes and even out-perform humans. In our project, we are interested in making trade decisions to profit from trading on a monthly time scale.

The task at hand is the following: Given the closing prices of the past 60 days of a given stock, predict whether the stock price will go up or go down during the next 60-day period. More formally, given a series of prices $X_1, X_2, \cdots, X_{60}$ for a stock, predict whether $X_{121} > X_{61}$ is true.

2 Related work

Because successful financial models are generally proprietary, it is difficult to find good existing literature with state-of-the-art performance on our specific task. However, there are a few papers that attempt to use deep learning models to tackle some aspect of stock price prediction. One common theme amongst such work is their use of recurrent neural networks (RNNs). For example, Flunkert *et al.* use an autoregressive recurrent network on inputs of purely historical prices (which may be insufficient given a general lack of arbitrage opportunity) to predict stock growth [3]. Bao *et al.* use stacked autoencoders to encode the inputs first before running the encoded outputs through an RNN. This particular study actually used 13 features for each day, including many technical indicators, to predict the next day's closing price [1]. A third study on RNNs more generally by Google shows that a forget bias of 1 in certain RNN architectures, especially long short-term memory (LSTM), is optimal [6]. We draw inspiration from all of these studies in building our RNN models and designing the input feature set, though our task is classification, while the above focused on regression.

In addition to common RNN techniques applied to stock data [1], Borovykh *et al.* used a 1D-Convolutional Neural Network to forecast stock prices [2]. They found that using multiple filters with different windows was able to capture cross-temporal patterns in the input. We also attempt this approach in our research, but again from a classification standpoint and an enhanced feature set.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

3 Dataset and Features

3.1 Dataset

We use daily stock prices from all stocks listed on NASDAQ. We use a dataset from AlphaVantage, an online database for finance data [10]. The dataset provides up to 2-3 years of daily quotes for each stock. We only consider stocks with at least 4 full months of data. For each day of each stock we are given the high, low, open and close prices along with the trading volume for that day.

3.2 Preprocessing

Along with the initial 5 basic features (high, low, open, close, volume), we augment the input with 25 other features from technical indicators that we compute on the basic features. These include features that show trend, momentum, volatility, etc. See the table below for a full list of technical indicators used (some of the indicators produce more than one feature).

MA5/MA10	5/10 day Moving Average
EMA20	20 day Exponential Moving Average
ROC	Price rate of change: shows the speed at which a stock's price is changing.
ATR	Average true range: indicates the degree of price volatility.
BBANDS	Bollinger Bands: provide a relative definition of high and low, which aids in rigorous pattern recognition.
PPSR	Pivot points, supports, and resistances: determine directional movements.
STOK	Stochastic oscillator: a momentum indicator that uses support and resistance levels.
ADX	Average directional index: an objective value for the strength of a trend.
MACD	Moving average convergence divergence: displays trend following characteristics and momentum characteristics.
RSI	Relative strength index: compares the magnitude of recent gains and losses over a specified time period to measure speed and change of price movements of a security.
MFI	Money flow index: measures the inflow and outflow of money into a security over a specific period of time.
CCI	Commodity channel index: helps to find the start and the end of a trend.
KELCH	Keltner channels: volatility-based envelopes set above and below an exponential moving average.

Table 3.2 Technical Indicators Used for Features

For each stock, we get a daily time series, which we split into 60-day windows as training data. We ended up splitting data from over 3000 stocks into 126,564 windows of 60 points each for our input data. For the output labels, we compute a boolean value according to whether the "low" price on the 120th day after the first day (in the input) is higher than the "high" price on the 60th day after the first day. We do choose these prices to factor in the liquidity costs associated with trading - we tend to buy at the higher price and sell at the lower price. We then proceed to split our dataset into training, evaluation, and testing datasets with a 90-5-5 partition.

3.3 Normalization

After obtaining the 60-minute windows of stock prices, we normalize the data before training so that each time window has zero mean and unit variance. Specifically, given a $n = 60$-minute window

$$X_0, X_1, \cdots, X_{n-1},$$

we compute the normalized prices

$$\hat{X}_i = \frac{X_i - \overline{X}}{S_X}$$

where $\overline{X} = \frac{1}{n}\sum_{i=0}^{n-1} X_i$ and $S_X = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(X - \overline{X})^2}$ and use $\{X_i\}$ as the input to the model instead.

For data that is log-normal (such as trading volume), we first log-transform the data

$$X' = \log(X + 1)$$

before feeding the transformed data X' into the normalization procedure.

3.4 Label Distribution

In both the training and validation data sets, the label distribution is roughly the following: 45% increase and 55% no increase. This means that for roughly 45% of the data, the price of the stock increases in the next 60-day period.

4 Methods

4.1 Baseline Models

We experimented with the following classification models to provide linear baselines as reference: ridge, stochastic gradient descent (SGD), and random forest. Models were implemented using scikit-learn [6] with hand-tuning of multiple parameters.

4.2 LSTM Networks

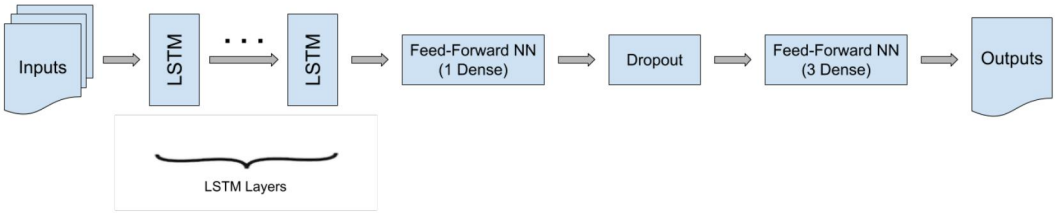

Figure 1: Graphical representation of the LSTM model.

4.2.1 Shallow LSTM

Given the recent success of RNNs in the domain of sequential data, we decided to apply one such network, the LSTM model, to our prediction task [4, 5]. Specifically, we first used the architecture shown in Figure 1 with only a single LSTM layer with 30 hidden units. Hence, the network is a shallow LSTM. The concatenated output of the hidden units is then fed into the next dense layer, followed by dropout, and finally additional dense layers.

4.2.2 Deep LSTM

Next we implemented a deeper, stacked LSTM model, the general outline of which is also depicted in Figure 1, with n LSTM layers where $n > 1$. For our experiments, we tried $n = 2$ and $n = 3$ with each LSTM layer having 30 hidden units as before. The value 30 was chosen because there are 30 total daily features, so it makes intuitive sense that a 30-dimensional encoding should best capture the daily data. The concatenated output of the hidden units of the last LSTM layer is then fed into 4 fully connected layers with dropout after the first.

4.3 1-D CNN

We also decided to try a 1-D convolutional neural network. The reasoning behind using a CNN is because it performs well at finding recurring local patterns with a concise model size, which we think is similar to what traders do when they look at graphs of prices and indicators. We use 9 1-D convolution layers of window sizes 3, 7, 11, 15, 19, 23, 27, 31, 35, convolved in the time axis, along with 1 global max-pool layer (over time) and one fully connected dense layer.

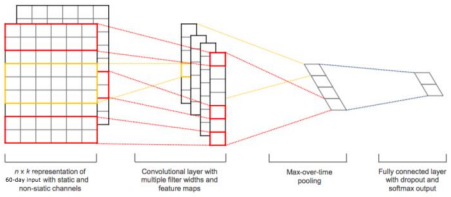

Figure 2: Graphical representation of the 1D-CNN model (Adapted from [7]).

5 Experiments/Results/Discussion

5.1 Model Results

Table 1 shows the baseline and deep learning results after optimization and hyperparameter tuning.

Table 1: Summary of model accuracies.

Model	Train Acc.	Validation Acc.	Test Acc.
Ridge	58.03	57.79	N/A
Random Forest	98.4	55.09	N/A
SGD	54.77	55.06	N/A
Shallow LSTM	63.62	59.10	55.2
Deep LSTM	58.5	57.70	55.3
1-D CNN	56.2	57.70	55.6

5.2 Baseline Analysis

The baseline models generally performed comparably to the deep learning ones. Only random forest overfits the training set because it is an ensemble of many strong classifiers. Interestingly, ridge classifier performs the best, while the other two baselines do no better than predicting the most frequent label. A possible reason is that the input data is noisy and has high bias, so a weaker classifier is actually able to generalize better.

5.3 Deep Learning Analysis

5.3.1 Evaluation and Hyperparameters

Each neural network architecture used sigmoid cross entropy loss and accuracy as its metric. We chose a mini-batch size of 512 over 50 epochs for each in order to make training over a large input set possible, especially for the deep LSTM networks. We found that a learning rate of 1.2e-3 worked well for the LSTM models, while 1e-4 was best for the 1D-CNN model. A dropout rate of 0.2 helped for the shallow LSTM model, but no dropout was actually best for the deep LSTM models since they were unable to fit the training data in the first place. For the 1D-CNN, we found that having fewer dense layers after applying all convolutions worked best, so we reduced the number of layers.

5.3.2 Performance

All three DL models are unable to significantly overfit the training data, which suggests that there may not be enough signal in purely market data to forecast even the price trend of a stock. Despite augmenting the feature set considerably, the technical indicators were unable to distinguish between whether or not a stock is going to increase. However, in our experimentation, the shallow LSTM model showed promise by achieving 75% training accuracy and 59% validation accuracy. As such, we applied l_2 regularization and dropout in order to mitigate the overfitting issue, though the validation accuracy remained roughly similar. This indicates that the inputs are quite difficult for the model to learn and generalize from, which can potentially be solved by training the models for more epochs or changing the feature set.

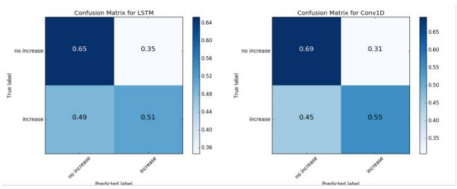

Figure 3: Confusion matrices for the shallow LSTM model (left) and the 1D-CNN model (right). The matrices are shown using results on the validation set to give us perspective on how the model was learning the classification.

We searched through different hyperparameters for each model, including learning rate, batch size, and dropout rate, and report the best validation results in Table 1. As the table shows, the models, even the shallow LSTM, do not really generalize to the test set. Thus, we analyzed the validation confusion matrices for the shallow LSTM and 1D-CNN models shown in Figure 3. We found that the models tend toward predicting no increase for which we hypothesize two possible reasons. The first is that it is easier to predict a stock decrease or stagnation than a stock increase. The second reason is that the models may actually be biased by the unbalanced label distribution (55-45 in favor of no increase).

In general, the LSTM models did seem to perform slightly better than the CNN perhaps because it is able to capture sequential patterns more effectively. Furthermore, we discovered that the shallow LSTM generally achieved higher accuracies than the deeper LSTMs because deep LSTMs have significantly more parameters that need to be learned. As such, they take longer to train and we found that increasing the number of LSTM layers decreased accuracy with the same number of training epochs. The Deep LSTM results reported in Table 1 are for 2 LSTM layers, which achieved slightly better accuracy than the 3-layer LSTM network.

6 Conclusion/Future Work

The main conclusion of this study is that stock price forecasting using solely market data is a difficult task even after simplifying the problem to a binary price trend and applying powerful deep learning models. Nonetheless, this information is useful in guiding future work, specifically in determining relevant feature sets and model architectures. We achieve our best validation results with a shallow LSTM network, though a deeper LSTM may perform better given additional resources for training. We originally attempted to predict the price deviation directly as a regression problem, and this is the approach that standard literature takes, but it may be more helpful to frame the problem as a classification task first, as we do here. Moreover, we have shown that having only historical pricing data or using vanilla deep learning models will not work well, but rather hand-tuned features with custom architectures seem necessary to forecast stock prices effectively.

There are several future directions for this project.

- Enriching our dataset. Certain stocks do not have enough data points and can be subject to small-sample bias. We can consider getting more data points (possibly using different sources) to build a more robust dataset. We can also consider including alternative data sources such as unstructured data from social media and news as helpful indicator features.
- Exploring different time horizons. It is possible that a 60-minute window does not capture all profitable patterns. We can explore a variety of different window sizes (e.g. second-to-millisecond-level data for high frequency trading) to maximize profitability.
- Related tasks. We can look for other kinds of financial data that might have better signal, such as market book transactions (for high frequency trading) or the economic sentiment of the general public (for predicting macro-economic trends).

7 Contributions

Kevin worked primarily on the data pipeline as well as building the 1D-CNN architecture. Glenn worked primarily on building the baselines and LSTM models and integrating the data into the net-

works. We were able to work pretty quickly and in parallel by making our tasks roughly independent. We visualized the data using confusion matrices and analyzed the results together.

Github Repository link

All code written for this project can be found in the following Github repository: https://github.com/kevinqli/deep-learning-trading.

References

[1] Bao, W., Yue J., & Rao, Y. (2017) A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PLoS ONE* **12**(7):e0180944.

[2] Borovykh, A., Bohte, S. & Oosterlee, C.W. (2017) Conditional Time Series Forecasting with Convolutional Neural Networks. *ArXiv e-prints*.

[3] Flunkert, V., Salinas, D. & Gasthaus, J. (2017) DeepAR: Probabilistic Forecasting with Autoregressive Networks. *ArXiv e-prints*.

[4] Hefron, R.G., Borghetti, B.J., Christensen, J.C., & Kabban, C.M.S. (1997) Long Short-Term Memory. *Neural Comput.* **9**(8):1735–1780.

[5] Hochreiter, S. & Schmidhuber, J. (1997) Long Short-Term Memory. *Neural Comput.* **9**(8):1735–1780.

[6] Jozefowicz, R., Zaremba, W. & Sutskever, B. (2015) An Empirical Exploration of Recurrent Network Architectures. *Proceedings of the 32nd International Conference on Machine Learning* **37**.

[7] Lenc, L. & Hercig, T. (2016) Neural Networks for Sentiment Analysis in Czech. *ITAT 2016 Proceedings* **1649**:48-55.

[8] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. & Duchesnay, E. (2011) Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* **12**:2825–2830.

[9] Wang, J., Wang, J., Fang, W., & Niu, H. (2016) Financial Time Series Prediction Using Elman Recurrent Random Neural Networks. *Computational Intelligence and Neuroscience* **2016**:doi:10.1155/2016/4742515.

[10] Alpha Vantage. (2018) www.alphavantage.co.