Implementation of Time-Delay Recursive Neural Network for Classification of Security Price Jumps

Gerald Tan (renhao@stanford.edu), Brandon Peh (bcp39@stanford.edu)
CS 230: Deep Learning

Motivation

- Stock price manipulations by "pump-and-dump" scammers involve systematic trading of stocks with uncertain fundamentals and low daily trading volumes. A recurrent neural network (RNN) trained on a set of time-delayed covariates relating to technical dynamics is shown to be capable of forecasting the occurrence of these activities in the US stock markets.
- We wish to use deep learning to identify stocks which are undergoing the above situation in the universe of US NYSE, AMEX and NASDAQ listed securities.

Data Processing

Our current model uses the daily prices, volumes, earnings per share and outstanding shares of the past 20 days to give a binary prediction of whether the stock price will increase by 50% over the next 5 days.

Since there was no available dataset that we could just simply download, we used a few popular screening metrics to determine which covariates were most likely to be predictive of a pump-and-dump. In total, there were 44 features that are used to predict a pump-and-dump and the features used are prices(20 values for past 20 days), volume(20 values for past 20 days), earnings per share and outstanding shares(including a dummy variable each for missing data). The tables below summarize the features used.

Pump-and-dump screening metrics	***Base fundamental metric***
Market capitalization	Number of outstanding shares, price
Price-to-earning ratio	Price, earnings
Earnings per share	Number of outstanding shares, earnings
Trading volume	Trading volume

Variable		***Data Extraction / Source***
Prices:	$X_{i,t-20}^{[1]}\ X_{i,t-19}^{[1]}\ \cdots\ X_{i,t-1}^{[1]}\ X_{i,t}^{[1]}$	Alpha Vantage (technical data)
Volumes:	$X_{i,t-20}^{[2]}\ X_{i,t-19}^{[2]}\ \cdots\ X_{i,t-1}^{[2]}\ X_{i,t}^{[2]}$	Alpha Vantage (technical data)
EPS:	$X_{i,t}^{[3]}$	Morningstar (fundamental data)
Outstanding Shares:	$X_{i,t}^{[4]}$	Morningstar (fundamental data)

References

- E. Hadavandi, H. Shavandi, A. Ghanbari, "Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting", Knowledge-Based System, Vol. 23, No. 8, 2010, pp. 800-808.
- A. Fan, and M. Palaniswami, "Stock Selection Using Support Vector Machines", Proceedings of the International Joint Conference on Neural Networks, Vol. 3, 2001, pp. 1793-1798.
- K.J. Kim, and W. Lee, "Stock Market Prediction Using Artificial Neural Networks with Optimal Feature Transformation", Neural Computing & Applications, Vol. 13, No. 3, 2004, pp. 255-250.
- K.Y., Shen, "Implementing Value Investing Strategy by Artificial Neural Network", International Journal of Business and Information Technology, Vol. 1, No. 1, 2010, pp. 12-22.

Model

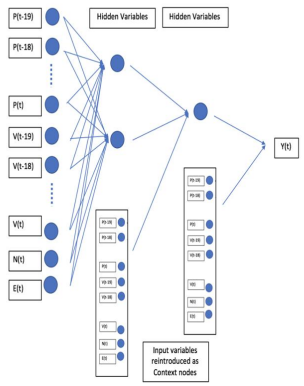


Figure 1: Structure of TD-RNN

We used a four-layer RNN that stores the variables of the preceding layer as context nodes. Past units (either inputs or hidden variables) are able to interact with subsequent units despite already having been fed-forward. The purpose of implementing such a neural network is to allow base variables like price and volume to interact non-linearly with derivative factors such price changes or price-earning ratios to codify more complex technical variations. Figure 1 illustrates how our RNN is structured.

$$h_t^{[1]} = ReLu(W_h^{[1]} X_t + b_h^{[1]})$$

$$h_t^{[2]} = ReLu(W_h^{[2]} h_t^{[1]} + U_h^{[2]} X_t + b_h^{[2]})$$

$$\hat{y}_t = Softmax(W_y h_t^{[2]} + U_y X_t + b_y)$$

$$\hat{y}_t = \mathbb{I}(\hat{y}_t > 0.5)$$

Figure 2: TD-RNN hidden layers

For the hidden layers, we employ ReLu activation function to accelerate learning and avoid vanishing gradient. Since we are solving a binary classification problem, a Softmax classifier is used for the outcome layer to generate an predictor between 0 and 1. Mathematically, the RNN is described as follows in Figure 2 and the loss function that we minimized is shown in Figure 3.

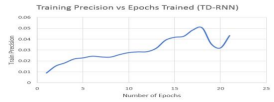
$$\mathcal{L} = -\sum \beta y_t \log(\hat{y}_t) + (1 - y_t) \log(1 - \hat{y}_t)$$

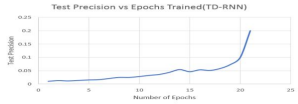
Figure 3: Loss Function

Results

We used a logistic regression model as a baseline model and then used an RNN to improve on the baseline results. The final results that we have obtained are as summarized:

Model	Loss Function	Training Precision	Test Precision
Logistic Regression	Cross Entropy	5.069%	4.000%
Logistic Regression	Weighted Cross Entropy	6.711%	5.769%
TD-RNN	Weighted Cross Entropy	See graph below	See graph below (up to 20%)





Conclusion and Future Work

The results obtained were not surprising. Given a weighted cross entropy loss function, we can make the cost of predicting a false positive higher, thus improving our precision greatly. However, an improvement in precision is not sufficient for the model to be applied to a real security with huge price movements. This is because the model may be able to predict a true positive accurately, but it gives too many false negatives. Therefore, most of the time, the model is unable to identify other true positives that may occur and this is an area that requires further research.

In future, we would hope to complement the classification TD-RNN with a LSTM neural net model which describe the price movements for each security. An investment decision to purchase a security can then be based on an ensemble learning combination of both models - one indicating a likelihood of a price jump and another forecasting future prices.