Enhanced Stock Movement Prediction with Attention LSTM and Adversarial Training

Haotian Zhang
Department of Industrial Engineering and Operation Research
Columbia University
New York, USA
hz2708@columbia.edu

***Abstract*—Attracted by the potential profit by stock trading powered by the latest deep learning models, asset management companies and investment banks are increasing their research grant for artificial intelligence. Market movement forecasting is one of the most attractive practical applications. In this paper, I investigate a enhanced solution to help predict stock movements, which aims to predict whether the price of a stock will be up or down in the near future. The rationality behind adversarial training is that input features, most based on stock price, are essentially stochastic variables and continuously changed with time in nature, which often cause overfitting. The based model is Attentive LSTM model, aiming to model the fact that the data at different time-steps might contribute differently to the representation of the whole sequence. Extensive experiments on the real-world stock data show that (i) ALSTM is more biased compared to LSTM and need regularization. (ii) Adversarial Training could be implemented to increase model's generalization ability and it achieve better performance compared to original models.**

***Index Terms*—attention mechanize, LSTM, adversarial training, stock movement prediction.**

I. Introduction

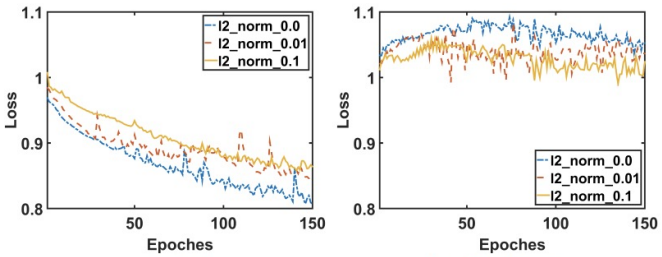
In recent years, several researchers and practitioners have been focusing on the financial market movement prediction. On the one hand, some funds are searching for alternative data sources from financial news/data providers, notably Thomson Reuters and Bloomberg, who have started providing commercial sentiment analysis services. On the other hand, more sophisticated machine learning models for algorithmic trading are purposed.

In this paper, I aim to focus the second part. Specifically, I try to answer the following research question:

1. How does attention mechanism apply to financial time series and is it reasonable?

2. Could adversarial training improve the generalization ability of predicting from the base model?

Before I embark on our investigation, it is necessary for us to clarify the intuition behind this paper. Long Short Term Memory (LSTM) have been applied to market movement classification in early 2010, though technically feasible, Feng [Feng et al., 2021] argues that such methods could suffer from weak generalization due to the highly stochastic property of stock market. Thoroughly explored the L2 regularization, the validation loss shown in Figure 1(b) does not exhibit a decreasing trend.



(a) Train (b) Validation

Fig. 1. LSTM and L2 regularization.

One viable explanation is when dealing with stochastic variable such as stock return, the static input assumption does not hold and such methods fail to generalize well. To address this issue, researchers at least two methods have been purposed: fuzzy theory, adversarial training. This paper would refer to Feng's idea [Feng et al., 2021] by tackling on the technique of adversarial training, which has been commonly used in computer vision tasks [Kurakin et al., 2017]. By adding perturbations on the high-level prediction features of the model, I aim at training the model to make it perform well on both clean features and perturbed features, to see if adversarial learning makes the classifier more robust and generalizable.

I implement the adversarial training proposal based on an Attentive LSTM model [Qin et al., 2017], which is a highly expressive model for sequential data. The attention mechanism aims to model the fact that data at different time-steps could contribute differently to the representation of the whole sequence. For stock representation, status at different time-steps might also contribute differently. For instance, days with maximum and minimum prices in the lag might have higher contributions to the overall representation.

II. Problem Formulation

To answer the questions above, I set a formulation of stock movement prediction task: to learn a prediction function $y^s = f(X^s; \theta)$ with the defination of all symbols used in this paper. I use the list of Symbols as follows:

$\mathbf{x}$	feature vector
$\mathbf{X}$	feature matrix
x, λ	scalars, hyper-parameters
t	time
S	number of stocks

Assuming that I have S=25 stocks, I learn the prediction function by fitting their ground truth labels y = $[y^1, \cdots, y^S] \in R^S$, where $y^s = (1/-1)$ is the ground truth label of stock s in the next time-step. I then formally define the problem as:

Input: A set of training examples $\{(\mathbf{X}^s; y^s)\}$.

Output: A prediction function $f(\mathbf{X}^s; \Theta)$, predicting the movement of stock s in the following time-step.

This paper I use monthly stock-level data of 25 selected US consumer companies with the objective series: (i) current month returns. The performance is measured by several metrics including test accuracy rate, recall to answer the two questions above.

III. Method

A. Attention LSTM

The Attentive LSTM (ALSTM) mainly contains several components: LSTM layer, temporal attention, and prediction layer, as shown in Figure 3.

LSTM layer. Owing to its ability to capture long-term dependency, LSTM has been widely used to process sequential data processing [Qin et al., 2017; Chen et al., 2018a]. The general idea of LSTM is to recurrently project the input sequence into a sequence of hidden representations. At each time-step, the LSTM learns the hidden representation by jointly considering the inputs and previous hidden representation to capture sequential dependency. I formulate it h(t) = LSTM($s_{t-1}; h_{t-1}$) of which the detailed formulation can be referred to [Hochreiter and Schmidhuber, 1997]. To capture the sequential dependencies and temporal patterns in the historical stock features, the LSTM is to learn the long, short term memory by three gates. An LSTM layer is displayed as below.

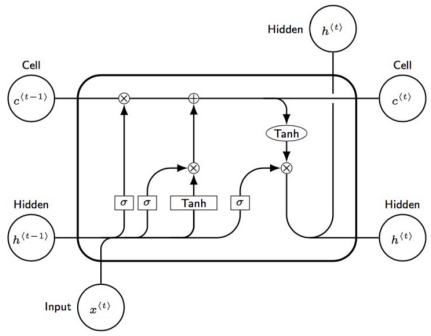


Fig. 2. LSTM Cell.

Attention layer. The attention mechanism has been widely used in LSTM-based solutions for sequential learning problems [Cho et al., 2014; Chen et al., 2018a]. The idea of attention is to compress the hidden representations at different time-steps into an overall representation with adaptive weights. The attention mechanism aims to model the fact that data at different time-steps could contribute differently to the representation of the whole sequence. For stock representation, status at different time-steps might also contribute differently. For instance, days with maximum and minimum prices in the lag might have higher contributions to the overall representation. As such, I use an attention mechanism to aggregate the hidden representations as,

$$\mathbf{a}^s = \sum_{t=1}^{T} a_t^s h_t^s;\ a_t^s = \frac{\exp \hat{a}_t^s}{\sum_{t=1}^{T} \exp \hat{a}_t^s}$$

$$\hat{a}_t^s = \mathbf{u}_a^T tanh(\mathbf{W}_a \mathbf{h}_t^s + \mathbf{b}_a)$$

where $\mathbf{W}_a, \mathbf{h}_t^s, \mathbf{b}_a$ are parameters to learned by the algorithm; and a^s is the aggregated representation that encodes the overall patterns in the sequence.

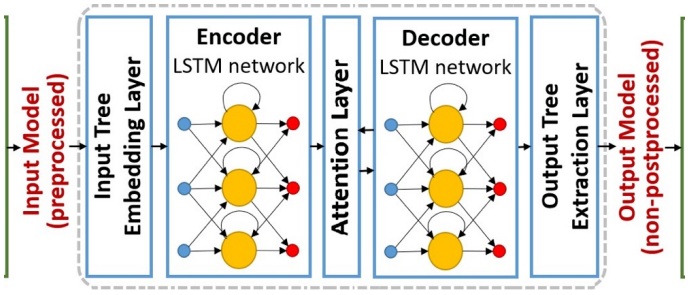


Fig. 3. Attention LSTM Framework.

Here I assume the stock return is a long sequence and I use LSTM as baseline by assuming the historical stock return will help predict the furture. The attention mechanism is to overcome the limitation that allows the network to learn where to pay attention in the input sequence for each item in the output sequence. This increases the computational burden of the model, but results in a more targeted and better-performing model. In addition, the model is also able to show how attention is paid to the input sequence when predicting the output sequence. This can help in understanding and diagnosing exactly what the model is considering and to what degree for specific input-output pairs.

B. Adversarial Training

Recent papers have applied some tricks on the architecture design of Adversarial Training. For example, StockNet uses a Variational Autoencoder (VAE) to encode the stock input so as to capture the stochasticity, and a temporal attention to model the importance of different timesteps [Xu and Cohen, 2018]; The VAE may be a natural way to do so by setting

$$\mathbf{X}^{adv} = \mathbf{X} + \epsilon * sign(\bigtriangledown_x J(\mathbf{X}, y_{true}))$$

Training Algorithm. The pseudocode is listed in Algorithm 1. The overall Adversarial Training algorithm is still naive by intentionally simulate samples by adding small perturbations on static input features. Feng [Feng et al.,

Algorithm 1 Training ALSTM with Adversarial Samples

Input: λ learning rate for network, N batch size, training epochs E, timestep for sequence t

0: **Initialization**

0: intentionally simulate samples by adding small perturbations on static input features $\mathbf{x_t^{adv}} = \mathbf{x_t} + \epsilon_i$ where $\epsilon_i \sim N(0, I)$.

0: Normalize features by training dataset, set up a generator for feeding data.

0: build up an Attention LSTM model, parameters is tuned through hparam.

0: **Training**

0: **for** e = 1, 2, 3,$\cdots E$ do **do**

0: Loading a batch of data

0: Calculate the current prediction based on current model

$$\hat{y_{k,t}} = net(\mathbf{h_{k,t-1}}, \mathbf{k}, \mathbf{x_t})$$

$$\hat{y_{k,t}^{adv}} = net(\mathbf{h_{k,t-1}}, \mathbf{k}, \mathbf{x_{k,t}^{adv}})$$

0: Set the Mean Square Error loss

$$\mathbf{J}(\theta) = \frac{1}{2n}(\sum_{i=i}^{n}(y_i - \hat{y}_i) + \beta \sum_{i=i}^{n}(y_i - \hat{y}_i^{adv}))$$

where β is a parameter to tune, here for simplification, set to 1.

0: Calculate gradient, Update $\theta = \theta - \alpha \bigtriangledown_\theta \mathbf{J}$

2021] has purposed the idea to add perturbation at the last layer to reduce calculation cost and improve performance. The performance is not measured, and could be implemented in further work.

IV. Experiment

A. Experimental Setting

I apply the framework to predict in the US stock market. My methodology is composed of 4 steps. (1) I clean the data by using Standard Scaler standardization and filling NA with median.(2) I introduce the features, (for news sentiment, I use aggregated mean group by RP ENTITY ID and date) whereas I set up the target: monthly return (binary). (3) I define the setup of the several baseline models I employ, namely random forest, XGBoost and GRU. (4) Finally, evaluate by accuracy score and confusion matrix related metrics.

Features This set of experiments are based on 25 selected US consumer companies. The features can be divided into three categories: (i) Sentiment Scores generated from monthly tweets data by NLTK (ii) technical factors (iii) fundamental factors. Most columns are not highly positive correlated, so I didn't do feature selection at this step. But when features increases, PCA could be applied to the cases.

Data The training samples used in the experiments are from 6 consecutive years (72 months), after concatenating all stocks I generate a 2D tensor of 1656 rows. To clean the data, I using Robust Scaler standardization and filling NA with the specific median of the company. I collect features and aggregate the results monthly, whereas I set up the target: (1) monthly return (binary). There are total 1324 rows in the data set, 735 of them are increasing, occupying 55.5% of all samples. This dataset is a little biased. To better predict the binary result, I scale the binary labels (-1/1) to 0-1 and then put into models. Then I classify the sample above 0.5 as a positive sample and vice versa. This achieve the best test accuracy among (i) no scale, predict binary (ii) scale between 0-1 (iii) no scale, predict values then change into binary.

Baselines Besides the model I use, I also compare the results with following base models: (1) ALSTM: Attentive LSTM [Qin et al., 2017], which is optimized with normal training. Similar as LSTM, I also tune U, T, and λ. (2) LSTM: LSTM is a neural network with an LSTM layer and a prediction layer [Nelson et al., 2017]. I tune three hyperparameters, number of hidden units (U), lag size (T), and weight of regularization term (λ). (3) GRU: a simplified version of LSTM. (4) XGBoost: a sample of boosting tree. The parameters are tuned by train-validation-test splitting. (5) Random Forest: a sample of bagging tree . The parameters are also fine tuned by train-validation-test splitting.

Evaluation Metrics I use a metrics to evaluate our networks performance. The metrics were accuracy (2), precision (3), recall (4) and F-measure (5), a harmonic mean between precision and recall. Those metrics are calculated based on positive classes (true positives – tp), negative classes (true negativestn) and inaccurately for both class (false positive- fp, false negative- fn).

Parameter Settings. I implement the Adv-ALSTM with Pytorch and optimize it using the mini-batch Adam[Diederik and Jimmy, 2015] with a batch size of 10 and an initial learning rate of 0.001 with momentum. I search the optimal hyper-parameters of Adv-ALSTM on the validation set. For T, Adv-ALSTM inherits the optimal settings from ALSTM, which are selected via grid-search within the ranges of [2, 3, 4, 5, **10**, 15]. λ, the L2 penalty coefficient on parameters, could be further tuned within [0.001, 0.01, 0.1, 1], respectively. I further tune β and ϵ within [**1**] and [0.1, 0.05, **0.01**, 0.005, 0.001], respectively. I report the mean testing performance when Adv-ALSTM performs best on the validation set over five consequent runs.

B. Experimental Results

Performance Comparison. The metrics were accuracy (2), precision (3), recall (4) and F-measure (5), a harmonic mean between precision and recall. Those metrics are calculated based on positive classes (true positives – tp), negative classes (true negativestn) and inaccurately for both class (false positive- fp, false negative- fn). Table 1 display the relative performance among RandomForest (baseline1), XGBoost (baseline2), GRU (baseline3), ALSTM and ALSTM-Adv. Based on the results at table 1, ALSTM could be further implemented, but some preliminary conclusions could be reached at this step.

TABLE I
TABLE OF PERFORMANCE MEASURE AND COMPARISON

Models	Performance Measurement			
	Test Accuracy	*Precision*	*Recall*	*F-measure*
RandomForest	65.1	66.3	82.5	73.5
XGBoost	61.7	64.9	75.8	69.9
GRU	**68.3**	70.2	80	74.8
ALSTM	56.6	58.4	90.7	71.0
ALSTM-Adv[a]	59.0	59.7	**98.4**	**74.3**

[a]ALSTM by adversarial Training, all numbers are in present

The original plan is to implement the cumulative profit curve of different methods with the portfolio of choosing top 5 stocks and compare it with the equal weight portfolio. Due to time limit, this part would be further implemented. To long the top 5 stocks suggested by the predictor at the market closing of the end of each trading month, and then sell these stocks upon the end of the next trading month. The predictions are based on around 60 features that are calculated before the market closing of that trading day at the end of each month.

Does Attention catch information? The test accuracy rate of the original LSTM is 68.3%, while ALSTN achieves an accuracy rate of 59.0%. This result is not expected. Since the data at different time-steps might contribute differently to the representation of the whole sequence. For example, the closer sequence would have more weights in prediction the value at the next time-step. Also, the features with high variance play an import roles in the prediction of the next following days. Attention mechanization is natural to be applied to stock movements prediction but it's harder to implement compared to LSTM. There are many ways that could go wrong for ALSTM. The main difficulty is to set up a data loader with generator functions and design the network without existing package about ALSTM, though Transformers have been widely applied to Natural Language Processing problems. Besides, the second difficulty is to tune the hyper parameters since for ALSTM it's important to select a time step, by which how long the memory of the sequence to keep. It's a tough question for financial data but here since I use less frequent data (monthly) and I simply set a small number 10. Lastly this work only sets up a baseline model and could be further implemented:

(i) use hparams package in tensorflow to tune the parameters including the number of layers, learning rate, time step.

(ii) more attention could be paid to the data engineering part. I use 58 features and most of them are different measures, so I assume not high correlated. But **SampleReweighing** could be applied in the financial data sets to distinguish between hard examples and confusing examples.

In conclusion, by the baseline method, it already achieves 68% test accuracy rate compared to 55% of random guess. The work could be further improved by fine tuning and the techniques I mentioned above. If time permits, further study could be on displaying attention scores and select a time slot for the detailed study.

Does Adversarial Training improve generalization? The Adversarial training is an alternative technique for training robust models besides fuzzy theory. Here I intentionally simulate samples by adding small perturbations on static input features. The test accuracy of the Adv-ALSTM doesn't improve much compared to the ALSTM. The epsilon rate is picked between [0.1, 0.05, 0.01, 0.005, 0.001] and 0.01 is further picked in term of the validation errors. Overall, the performance is not so well because of the naive design. Several modern techniques could be applied for a more sophisticated design like FSGD and add perturbation to different layers of ALSTM. So far, seldom conclusions can be reached: (i) the small perturbation slightly improve the models' generalization ability on the test set because noteworthy the test accuracy is around 3% higher than the train accuracy.

V. Conclusion

First I deduce that neural network solutions for stock movement prediction could suffer from weak generalization ability since they lack the ability to deal with the stochasticity of stock features. To address this, I proposed an Adversarial Attentive LSTM solution, which leverages adversarial training to simulate the stochasticity during model training. I conducted elementary experiments on validating the effectiveness of the proposed solution, signifying the importance of accounting for the stochasticity of stock prices in stock movement prediction. Morever, the results showed that adversarial training enhances the generalization of the prediction model. In future, following directions could be tried: 1) Other architecture for adversarial training, including deducing the computation cost or improving prediction robustness. 2) Other baseline models besides Attention-LSTM could be tried. 3) Other assets such as bitcoins, commundities could be studied.

This paper is greatly inspired by Enhancing Stock Movement Prediction with Adversarial Training [Feng et al., 2021].

Acknowledgment

Author Haotian Zhang acknowledges great support from B9654 Artificial Intelligence course. Specially thank for Prof. Lentzas's well design lecture notes, assignments, teaching, thank Uliana for strong support both in holding office hours and answering my questions. All the best for your future endeavors!

References

[1] [Musgrave, 1997] G. Musgrave. A random walk down wall street. Business Economics, 32(2):74–76, 1997.

[2] [Qin et al., 2017] Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In IJCAI, pages 2627–2633, 2017.

[3] [Feng, 2021] F. Feng, H. Chen, X. He, J. Ding, M. Sun, T. Chua. Enhancing Stock Movement Prediction with Adversarial Training. In IJCAI, pages 5843–5849, 2021.