Parallel prediction of stock volatility

Priscilla JENQ
School of Computer Science
Carnegie Mellon University, PA 15213, USA

and

John JENQ
Computer Science Department
Montclair State University
Montclair, NJ 07043, USA

ABSTRACT

The financial industry is an industry that requires multi-disciplinary expertise. To be a good financial engineer, one should possess skills in math, finance, economics, and coding. Volatility is a measurement of the risk of financial products. A stock will hit new highs and lows over time and if these highs and lows fluctuate wildly, then it is considered a high volatile stock. Such a stock is considered riskier than a stock whose volatility is low. High tech stocks usually have high volatility. Although these stocks are riskier, the returns that they generate for investors can be quite high. Of course, with a riskier stock also comes the chance of losing money and yielding negative returns. In this project, we will use historic stock data to help us forecast volatility. The financial industry usually uses S&P 500 as the indictor of the market. Therefore, S&P 500 would be a benchmark to compute the risk. We will use artificial neural networks as a tool to predict volatilities for a period of time frame that will be set when we configure this neural network. There have been reports that neural networks with different numbers of layers and different numbers of hidden nodes may generate varying results. As a matter of fact, we may be able to find the best configuration of a neural network to compute volatilities. We will implement this system using the parallel approach. The system can be used as a tool for investors to allocating and hedging assets.

Keywords: Artificial Neural Network, Volatility, Parallel Processing

1. INTRODUCTION

The financial industry is one that requires multi-disciplinary expertise. To be a good financial engineers, it requires one to be good in math, finance, economic, and coding skills. The stock market is very volatile and sensitive to various factors, such as politics and war. For example, when there natural disasters, political turmoil, or economic and financial crises occur, financial assets tend to fluctuate very much.

Volatility is a measurement of the risk of financial products. A stock will hit new highs and lows over time and if these highs and lows fluctuate wildly, then it is considered a high volatile stock. Such a stock is considered riskier than a stock whose volatility is low. High tech stocks usually have high volatility and have a so-called higher beta value, i.e., beta value > 1. Although these stocks are riskier, the returns that they generate for investors can be quite high. Of course, with a riskier stock also comes the chance of losing money and yielding negative returns. On the other hand, utilities are more stable in their stock prices so they are considered low volatile, with beta value < 1. A beta value of 0 signifies a security that has no volatility; for example, cash has beta value of 0. It is known that standard deviation alone cannot be used to measure volatility because, as illustrated by the history of the stock market, the market is not normally distributed and in reality, is skewed. Thus, historic stock data may in fact help us to measure volatility. In this report, we will use historic data to help us to compute the volatility. We will use artificial neural network as a tool to predict volatilities for a period of time that will be set when we configure this neural network. There have been reports that neural networks with different layers and different nodes may generate varying results. As a matter of fact, we may be able to find the best configuration of neural network to compute volatilities. We will implement this system using parallel approach. The system can be used as a tool for investors to allocating and hedging assets.

Neural networks are popular in financial and economic computations. For example, Li and Liu used LM BP algorithm to predict the Shanghai stock market [3]. Wang developed an HLP method that gets stock high low points with different frequency and amplitude. The extracted data are then fed into a neural network to

forecast the stock direction and price [8]. Tirados and Jenq used neural networks to predict GDP with ten leading economic indicators as the input [7]. Lin and Feng combined neural networks and pattern matching techniques to analyze and to forecast oil stock price [4]. Zhou and Zhang used financial indicators such as moving averages, volumes, relative strength index, etc. on neural network to predict future stock price [9].

Amornwattana et. al[1] proposed a hybrid artificial neural network (ANN) model for forecasting volatility to do options trading. Hajizadeh et al. [2] proposed a hybrid model with ANN to forecast the volatility of the S&P 500 index. Monfared and Enke [5][6] also proposed a hybrid GJR-GARCH Neural Network model to enhance the performance of volatility forecasting using an adaptive neural network filter for cancelling noise in the data. Youngmin Kima and David Enke discussed using neural networks to forecast volatility for an asset allocation strategy based on the target volatility [10]. Kim et. al[11] proposed a system for early warning of economic crisis.

In this project, we would like to find the local minimum number of hidden nodes in a single hidden layer that would be able to achieve the best prediction in terms of accuracy. We decided to use only a single hidden layer because adding more and more layers would become computationally expensive quickly while only a relatively small amount of performance improvements would be seen.

This proposed problem can benefit greatly from parallelism because most people usually try to find the number of hidden nodes to use through either trial and error or simply by just using the same number of nodes in input layer, both of which can be very ineffective. With parallelism, we were able to speed up the process of finding a local optimal number of hidden nodes for fairly accurate stock price predictions. This paper is organized as the following. We define some terms that were used in our project. Section 3 will describe the development of the system. We discuss the implementation results in sections 4. Section 5 gives the conclusions.

2. TERMINOLGIES

MSE (mean squared error): This value is one of the criteria to stop the training of the neural network. MSE is defined as the sum of the square of errors of outputs divided by the total number of cases involved in the training. The error used here is defined as the difference between the actual (target) value and the predicted value generated from the ANN. Note that there are other methods to compute the MSE, such as using validation data set to stop the training by checking if total number of accuracy is improved and if total accuracy hasn't, then one can stop the training. We did not use this method to stop the training in this project. To stop training, we set a tolerable error and a maximum cycle, and the one that was reached first would stop the training.

Accuracy: Because the predicted price and the actual price must change in the same direction to be considered a good prediction for the stock market (and can at least help us to make the decision to either buy, sell or hold), accuracy can be defined as the number of same directional changes of predictions divided by total number of testing cases. We called this the "hit ratio." Another way to define it is to say that a lower MSE means a higher accuracy because it shows how close the predicted value is from the target value for the values that change in the same direction.

Speedup: This is defined as the ratio of sequential runtime divided by the parallel run time

Efficiency: This is defined as the speed up factor divided by the number of threads used to execute the program.

3. SYSTEM DESIGN AND IMPLEMENTATION

We implement the system using C/C++. There are 16 threads in the system that we used. The program can run multiple threads simultaneously. OpenMP is used to parallelize the program code. Since we'd like to find the local optimal hidden layer nodes, we structure our program so that we can use all threads as much as possible.

The pseudo code of our program is outlined below.

1. Process command line arguments and set corresponding variables of the program
2. Read in data file and put the data cases into data array for future reference.
3. Process the data by normalization so the data values will be converted from arbitrary values to values between -1 and +1
4. Initialize ANNs // the number of ANNs to be trained and tested
5. Parallel do the following using ANN with different hidden layer configuration {
6. While (MSE > tolerable error && cycle < maximum cycle) {
7. For all training data do {
8. Forward-propagation
9. Backward propagation of errors
10. Update weights

```
11. }
12. Compute MSE
13. }
14. // test this ANN
15. Test run current ANN with test data
16. Compute accuracy
17. }
```

The first approach was to parallelize the execution of training of all configured ANNs to run concurrently (line 6). This means that each n node ANN would be assigned to a specific thread run. This approach allowed one to learn the workload of each segment of the program code during different iterations of the program. It was obvious that assigning a different number of hidden nodes to an ANN required different efforts, i.e., for an ANN with small hidden layer nodes, one can accomplish the training and testing quicker than the ANN with higher number of hidden nodes. Since we assigned a dedicated thread to work on an ANN, this approach gave the situation of an unbalanced workload among various threads. This is because a thread working on an ANN with 1 hidden node, for example, will obviously finish faster than a thread working on an ANN with 9 hidden nodes. So even though there was some speedup with static scheduling, we felt that this could be improved.

In order to parallelize even further, consider line 7, which trains the network by going through one data item at a time. Each time, it goes through the forward phase, backward phase and then modifies the weights after finding the gradients using the gradient descent method. This is the so-called stochastic method (also known as online method or incremental method in various internet literatures). So instead of updating the weights one at a time, the weights can be modified as a collection, i.e., compute the weight changes (the gradients of weight change based on the error that was back propagated from the output layer) of all data items and keep these weight changes in a temporary data structure. After processing all data items, the weights can be updated in parallel by adding up all the weight changes for each item together. This method is usually known as *batch method*.

To further improve the performance, we changed the OpenMP scheduling type from static to dynamic and experimented with an increasing counter in the for loop (i.e. iterating from 0 to total_ann) and a decreasing counter in the for loop (iterating from total_ann to 0). We found that using increasing or decreasing counters did not yield a significant difference in results. However, the results shown in the next section proves that reorganizing the program code from static to dynamic affected and improved the performance of the system.

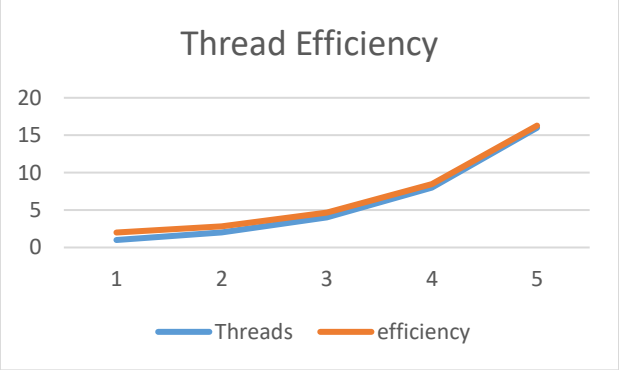
4. EXPERIMENTAL RESULTS

The following table shows the speedup of running the program using different number of threads. The time is the total number of seconds from beginning to the end of running the whole program, including the testing phase. It includes the sequential part and the parallel part of this program.

number of threads	time	speedup
1	271.744	1
2	166.021	1.636805
4	105.95	2.564832
8	72.7911	3.733204
16	59.3806	4.576309

The above table shows the speedup results when using dynamic scheduling of the program code. Although the speed up is not linear, it does show some speed up.

The corresponding efficiency of the results is drawn as the line chart. The results show that efficiency decreases when the number of threads are increased.



We also found that by changing the order of execution of the ANN, the results were affected. The following shows the speedup from running the program with static scheduling.

number of threads	speedup
1	1
2	1.353529353
4	2.746868554
8	4.459963154
16	4.086538969

The results were interesting because it showed that the speedup from 16 threads with static scheduling was worse than the speedup from 16 threads with dynamic scheduling. This is most likely is due to the workload imbalance that occurs with static scheduling. This type of schedule will result in certain threads doing less work than other threads, as running the program with a lower number of hidden nodes will result in lesser work. It is also interesting to note that when using static scheduling, running with 8 threads gave a better speedup than 16 threads. It is possible that the overhead from thread creation far outweighed the benefits of using more threads.

5. CONCLUSIONS AND REMARKS

The financial industry is an industry which requires multiple disciplines to work together. Skills including coding, math, psychology, politics may be necessary to ensure its success.

In this project, we implemented a neural network which can predict volatility of stock using feedforward and backward propagation method to change weights of neural network. OpenMP was used to implement the parallel program. There was speedup observed, although not linear, and efficiencies were also computed. Static and dynamic scheduling methods of OpenMP were implemented and the results from these two schedules were compared.

It was found that increasing the number of hidden layer nodes did not imply better results. It would be interesting to further examine what happens when one adapts more layers in the computation. How many hidden nodes would be required in this case? And how would we decide the number of nodes in each hidden layer?

In addition, further work could be done to figure out what other factor(s) should be fed in into the system so that we would have a better hit ratio and accuracy, with regards to determining the volatility of stock.

6. REFERENCES

1. Sunisa Amornwattana, David Enke, C.H. Dahli, A hybrid option pricing model using a neural network for estimating volatility. International Journal of General Systems, 2007, pp 558-573.
2. E. Hajizadeh, A. Seifi, Fazel M.H. Azrandi, I.B. Turksen. A hybrid modelling approach for forecasting the volatility of S&P 500 index return, Expert Systems with Applications, 2012, 39: 431-436.
3. Feng Li, and Cheng Liu, Application Study of BP Neural Network on Stock Market Prediction, Ninth International Conference on Hybrid Intelligent Systems, pp 174 – 178
4. QianYu Lin, and ShaoRong Feng, Stock market forecasting research based on Neural Network and Pattern Matching, 2010 International Conference on E-Business and E-Government, pp 1940 – 1943
5. Soheil Almasi Monfared, and David Enke, Volatility forecasting using a Hybrid GJR-GARCH neural network model. Procedia Computer Science, 2014, pp 246-253.
6. Soheil Almasi Monfared, and Enke D. Noise cancelling in volatility forecasting using an adaptive neural network model. Procedia Computer Science, 2015, pp 80-84.
7. Edward Tirados and John Jenq, (2009), "Analysis of Leading Economic Indicator Data and Gross Domestic Product Data Using Neural Network Methods", Journal of Systemic, Cybernetics and Informatics, vol 7, no 4, 2009, pp 51-56
8. Lei Wang, and Qiang Wang, (2011) Stock market prediction using artificial neural networks based on HLP, 2011 Third International Conference on Intelligent Human-Machine Systems and Cybernetics, pp 116 -119
9. Yixin Zhou, and Jie Zhang,mStock data analysis based on BP neural network, 2010 Second International Conference on Communication Software and Networks, pp 396 – 399
10. Youngmin Kima and David Enke, Using Neural Networks to Forecast Volatility for an Asset Allocation Strategy Based on the Target Volatility, by Procedia Computer Science 95 (2016) pp 281 – 286
11. Kim TY, Oh KJ, Sohn I, Hwang C. Usefulness of artificial neural networks for early warning system of economic crisis, Expert Systems with Applications, 2004, 26(4), pp 583-590