STAT598 Project: Deep Learning for Futures Price Prediction

Min Ren
Purdue University
ren80@purdue.edu

Zhou Shen
Purdue University
shen171@purdue.edu

Abstract

Rapid development of deep learning has revolutionarized the computer vision and NLP field in the past decade. It's very natural to ask if we can apply the deep learning model to help sovle the futures price related prediction task. In this paper, we investigated both of Covolutional Neural Networks (CNN) and Long Short Term Memories (LSTM) Recurrent Networks on three different tasks: mean price prediction, volatility prediction, return prediction. We conducted our experiments on two chinese future contracts: ironore and beanpulp. We successfully found that both of CNN and LSTM may be useful for volatility prediction task and worth further investigation, while these two models might not be good enough in the mean price prediction and return prediction tasks in our experiemts.

1. Introduction

A futures contract is a standardized forward contract, a legal agreement to buy or sell something at a predetermined price and a specified time in the future. The asset transacted is usually a commodity or financial instrument and the transaction is usually done on the trading floor of a futures exchange. Researchers in the area of mathematical finance has built many elegant models of future price by using stochastic analysis, which get great success and contribute to the pricing of various financial derivatives and the flourish of global financial markets. However, like stock price, future price can fluctuate rapidly due to various issues, e.g., the change of demand and supply in the market, politics and economy, and trading activities. The time series of future price contain huge noise and random effect, which makes its prediction still a hard problem. And actually, one with logic and common sense can argue that it is impossible to predict the change of future price with high accuracy, just like uncertainty principle in quantum physics. Fortunately, a lot of techniques in statistics, machine learning and deep learning can provide a good guess about the price movement in the future with acceptable accuracy, which dramatically develops the area of quantitative trading and has been successfully applied by companies on the buy side like hedge funds and trading firms. In this project, we would like to apply the powerful techniques in deep learning to the prediction of future price, and compare its performance with some popular models in time series analysis. It will contribute to the design of high and medium frequency quantitative trading strategies.

1.1. Data

Our raw data in this project consists of prices and volumes of Chinese commodity futures on a minute basis from 2010 to 2015. Type of future contracts covers bean pulp, soybean oil, iron ore, gold, PTA, rubber and steel bar. Each row of the data contains information of the open, high, low, close prices and volume during a minute as well as the delivery date. Note that there exists multiple future contracts for a single commodity with different delivery dates. Usually, the one with the closest delivery date achieves the highest trading volume, which is called the dominant contract. Therefore, the prices and volume of dominant contracts should be identified and organized by time. Our modeling and prediction are based on this reorganized time series data of dominant contracts.

1.2. Existing Methods

Traditional methods apply statistical learning and time series analysis tools to predict price change, including linear regression, SVM, ARIMA models, HMM and so on. These models are comparatively easy to interpret and get decent prediction accuracy rather than random guess.

Nowadays, the dramatical development of computing power and the theoretical contribution from deep learning bring more powerful tools to this area. Various neural networks have seen an explosion of interest over the last few years, and are being successfully applied in areas like finance. Maknickien et al.(2012) [3] gives a good review on these methods. ARIMA based neural network was proposed by Jung et al. (1996) [9] which is capable of predicting up to 6 weeks market trend with acceptable accuracy. Modification and acceleration of Modular Neural Network (MNN) was described in the the article of Schmidt and Bandar (1997) [7]. Fuzzy Boolean Neural Network by Tome and Carvalho (2005) [8] has also been used for prediction of stock market indexes. This model has been proposed as nets capable to learn qualitative rules and reasonably to use these rules. Qiang et al. (2005) presents an improved neural network model titled Amnestic neural network, which simulates human cognitive behavior of forgetting, to solve the problem of cross-temporal data selection. Multi-branch neural networks (MBNN) could have higher representation and generalization ability than conventional NNs by Yamashita and Hirasawa (2005) [10]. The Multi-layer perceptron as well as Radial Basis Function neural network architectures were implemented as classifiers to forecast the closing index price performance by Patel and Marwala (2006) [6]. Exploit of RNN based on genetic learning algorithm (EVOLINO) has a statistically significant number of successful predictions, when parameters of RNN are properly chosen for prediction, but not without risk by investing according to Maknickien et al. (2011) [5].

Authors Rai and Rai (2011) [4] summarized the comparison of different Neural Network types for stock prediction. Despite enormous previous efforts and a wide range of methods applied to this problem, efficient stock market prediction remains a difficult task mainly due to complex and varying in time dependencies between factors affecting the price.

2. Models

In this project, we build two categories of neural networks including CNN and LSTM to test the proper ways of predicting futures market. There are many measures to describe the movement of the futures market, which should also be taken into consideration as comparison.

2.1. Inputs and Outputs

The original input of any models is the time series of five-dimensional futures data on a minute basis, including open, high, low, close prices and volume. We cut consecutive elements of length of a window size as the feed of a neural network; the data in the following minute or several minutes will be the corresponding output.

It is not well known which specific measure will be the most suitable target in this supervised learning problem. One apparent option is the close price in the one following minute or the average of close prices in the following several minutes. This measure will summarize the ultimate change of the short-term market status well. The second option is average of the four prices in the following minutes or their average in the following several minutes. It will be more stable than the single close price and includes the movements within minutes in some sense. The vibration of these measures highly relies on the price level, which causes the time series of output non stationary. Traditional time series models usually require data to be stationary. It is not necessary for neural network models, but we are not sure if this will influence the performance in neural networks.

Some transformation of both inputs and outputs will change the time series data to be stationary. For example, we can model and predict the difference between the high price and the low price within the next minute; the corresponding inputs will be modified to the gap between high and low, the average of four prices for every minute and the volume. This measure will estimate the volatility of the market and offer good reference for further design of trading strategies. Another reasonable transformation is based on the Geometric Brownian Motion assumption of the futures price. Suppose S_t is the futures price at time t and B_t is a standard Brownian motion, then the dynamics of the price under this assumption is

$$d \log S_t = \mu dt + \sigma dB_t$$

Therefore, we can use the $\log(close) - \log(open)$ within a minute as the target to train the network.

2.2. Convolutional Neural Networks (CNN)

Since 2012, one of the most important results in Deep Learning is the use of convolutional neural networks to obtain a remarkable improvement in object recognition for ImageNet. Convolutional Neural Networks are very similar to ordinary Neural Networks: they are made up of neurons that have weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. These then make the forward function more efficient to implement and vastly reduce the amount of parameters in the network.

It is reasonable to attempt CNN in this price prediction task as well. The input time series is a *window-size* $\times$ *dimension* matrix, which can be regarded as a 1D image as a analogy of a regular 2D image like $128 \times 128 \times 3$. The rationale to apply CNN on time series data is to see the sequence in its entirety and capture features from low levels to higher levels which can depict the market abstractly and profoundly. Compared with fully connected neural networks which assigns weights for all elements in each neuron, CNN has less weights and can deal with longer length of inputs during similar training period.

Convolutional Networks are commonly made up of only three layer types: convolutional layer (CONV), pooling layer (POOL) and fully connected layer (FC). We will also explicitly write the RELU activation function as a layer, which applies elementwise non-linearity.

The CONV layer is the core building block of a CNN. The CONV layers parameters consist of a set of learnable

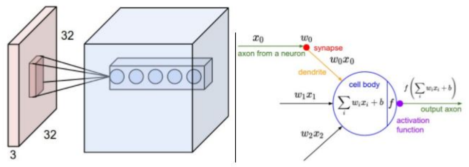

Figure 1. Example volume of neurons in the first Convolutional layer of a 3-D image input

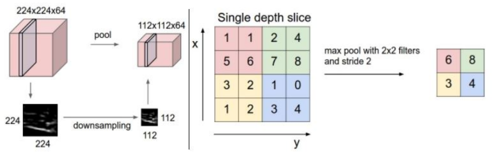

Figure 2. Example of pooling layer downsampling the volume spatially, independently in each depth slice of the input volume

filters. Every filter is small spatially along length, but extends through the full depth of the input volume. For example, a typical filter on a first layer of this ConvNet might have size 5×5 (i.e. 5 consecutive elements in length and 5 because each input time step has 4 prices and a volume). During the forward pass, we slide (more precisely, convolve) each filter across the length of the input time series and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the length of the input time series we will produce a 1-D activation map that gives the responses of that filter at every spatial position. Now, we will have an entire set of filters in each CONV layer (e.g. 10 filters), and each of them will produce a separate 1-D activation map. We will stack these activation maps along the depth dimension and produce the output volume.

It is common to insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2 applied with a stride of 2 downsamples every depth slice in the input by 2 along length, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 2 numbers (little 2 length in some depth slice). The depth dimension remains unchanged.

Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset. The output of the FC will be the predicted value. CS231n in Standard University [1] provides great details about CNN.

2.3. Long Short Term Memory (LSTM) Recurrent Networks

Recurrent neural networks are usually used for sequence prediction. There are typically three sets of parameters: the input to hidden weights (W), the hidden to hidden weights (U), and the hidden to label weight (V). Notice that all the Ws, Us and Vs are shared. The weight sharing property makes our network suitable for variable-sized inputs. Even if Ts grows, the size of our parameters stay the same. We can then minimize our cost function $(yf(x))^2$ to obtain the appropriate weights by computing gradients through backpropagation. Backpropagation through time for recurrent neural networks is usually difficult due to a problem known as vanishing/exploding gradient [16]: the magnitude of the gradient becomes extremely small or extremely large towards the first or the last time steps in the network. This problem makes the training of recurrent neural network challenging, especially when there are a lot of long term dependencies. The reason for this vanishing/exploding gradient problem is due to the use of sigmoidal activation functions in recurrent networks. As the error derivatives are backpropagated backwards, it has to be multiplied by the derivative of the sigmoid (or tanh) function which can saturate quickly. One can imagine that the ReLU activation function can help here as its derivative allows for better gradient flow. However, another problem arises as we use ReLU as the activation function in the forward prop computation.

The most successful attempt to improve the learning of recurrent networks to date is Long Short Term Memory (LSTM) Recurrent Networks. The idea behind LSTM is to modify the architecture of recurrent networks to allow the error derivatives to flow better. At the heart of an LSTM is the concept of memory cells which act as an integrator over time. Suppose that input data at time *t* is x_t and the hidden state at the previous time step is h_{t1}, then the memory cells at time *t* have values:

$$m_t = \alpha \cdot m_{t-1} + \beta \cdot f(x_t, h_{t-1})$$

Just think of m_t as a linearly weighted combination between m_{t-1} and f. A nice property of m_t is that it is computed additively and not associated with any nonlinearity. So if the error derivatives cannot pass through the function f at time step t (to subsequently flow through h_{t1}), it has an opportunity to propagated backward further through m_{t1}. In other words, it allows another path for the error derivatives to flow. To prevent m from exploding, f is often associated with a sigmoid/tanh function. Built on this construct, the hidden state of the network at time t can be computed as following:

$$h_t = \gamma \cdot tanh(m_t)$$

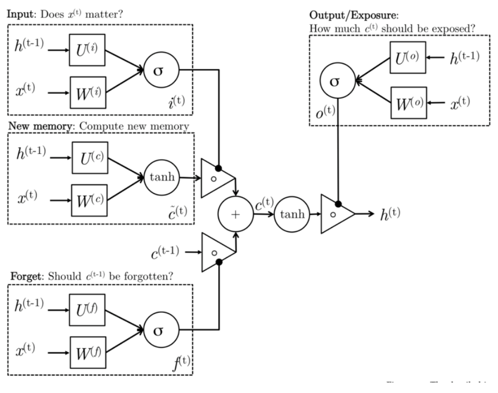

Figure 3. Architecture of LSTM

Note again that both h_t and m_t will be used as inputs to the next time steps, and that means the gradient has more opportunities to flow through different paths.

The variables α, β and γ are often called gates as they modulate the contribution of input, memory and output to the next time step. At a time step t, they are computed as:

$$\begin{aligned}
\alpha(t) =& \sigma(W_{x\alpha}x_t + W_{h\alpha}h_{t1} + W_{m\alpha}m_{t1} + b_\alpha) \\
\beta(t) =& \sigma(W_{x\beta}x_t + W_{h\beta}h_{t1} + W_{m\beta}m_{t1} + b_\beta) \\
\gamma(t) =& \sigma(W_{x\gamma}x_t + W_{h\gamma}h_{t1} + W_{m\gamma}m_{t1} + b_\gamma) \\
f(x_t, h_{t1}) =& tanh(W_{xm}x_t + W_{hm}h_{t1} + b_m)
\end{aligned}$$

Le (2015) [2] summarize LSTM well in a deep learning tutorial.

3. Experiment

3.1. The setup

We conducted the experiments for both of the CNN and the LSTM. The arthitecture of the CNN study we used was CONV, POOL and FC. The hyperparameters of the CNN are summarized in the table below:

Parameters	value
width of sliding window	240
stride of sliding windwo	3
kernel size	4
number of filters	20
batch size	300

Table 1. The hyperparameters for the CNN experiment

For the LSTM study, the network sturture we employed was one LSTM layer and a FC dense layer. The corresponding hyperparameters of the LSTM experiment are summarized in the table below:

Parameter	value
number of recurrent steps	8
width of sliding window	30
stride of sliding window	3
batch size	300

Table 2. The hyperparameters for the LSTM experiment

We emphasize here that we used 30 steps for the width of sliding window of the LSTM, instead of the 240 steps used in the CNN experiment. This discrepancy was due to the different traning speeds of the LSTM and CNN. The training speed of LSTM is way slower than that of CNN. Therefore, we reduced the width of sliding window of LSTM in order to speed up the training given the limited time and available computional power. The hidden state and memory cell is a 8×1 vector in our settings.

We chose to compare the CNN and LSTM experiment under the following three measures or scenarios. For all the study, we randomly selected 90% of the data as the training dataset and used the other 10% as the validation dataset.

1. Measure 1(mean price prediction), Inputs: $open$, $high$, low and $close$ price and the $volume$. Outputs: mean of following three future bars

2. Measure 2(volatility prediction), Inputs: $high - low$, four price average(average of $open$, $high$, low and $close$ price), $volume$. Output: $high - low$ price

3. Measure 3(return prediction), Input: $\log(close) - \log(open)$, Output: $\log(close) - \log(open)$

All the experiment were implemented on Keras/Tensorflow plateform and the results were trained on *HalsteadGPU* cluster with Nvdia Tesla P100 GPU. The maximum number of training epochs we used is 1000. However we stopped the tranining early if we observed the training and validation curves became flat.

3.2. Results

The results of the experiment are summarized in the table (3). We demonstrate our results by two criteria: *val_loss* and R^2. The val_loss is the mean of squared error and the R^2 is the coefficient determination. Both of the criteria were measure on the validation set. We will mainly focus on the R^2 to evaluate the prediction power between different methods.

	CNN		LSTM	
	val_loss	R^2	val_loss	R^2
Measure 1	41	0.997	526138	-23.2
Measure 2	0.3459	**0.2428**	0.3764	**0.15**
Measure 3	1.5611e-06	-0.002	1.6255e-06	-0.0045

Table 3. Performance of each measure on *ironore* future contact by CNN and LSTM. *val_loss* denotes the mean square of error on the validation set. R^2 is the coefficient determination

For the measure 1, the CNN can predict the average price of following three bars very well, with R^2 very close to 1. However our LSTM failed completely. The corresponding R^2 of LSTM falls into far way negative region. We suspect that some of our setup in this case is not appriopriate. Even thought the CNN predicts very well in this case, we should be cautious about this resuts. First, as the price usually does not change dramatically in one minute, thus any modeling method in this case are expected to predict well. Second, since the $val\_loss = 41$, the absolute prediction error is around $\sqrt{41} = 6.4$. After examing the data, the avarage change in one minute is around 2. Thus, in terms of short term prediction, the prediction error is still way larger. This indicates that the results in this case may not be very useful for short term scraping strategy.

For the measure 2, both of the CNN and LSTM have positive signals, which two R^2 fall nicely into the positive region. The CNN have rougly 24% R^2 and the LSTM has smaller but close 15% R^2. This indicates that deep learning models can be potentially futhur developed to help improve the volatility based or related strategy. We emphasize here that this case is different from the CNN in measure 1 case, where high R^2 is expected and may not useful, while The prediction target in measure 2 is "volatility(high-low)", where high R^2 is not expected. In other words, our model may indeed find some interesting signals in the data.

For the measure 3, Both of the R^2 of the CNN and LSTM are struggling around the 0 barrier. This indicates that these two models might not be useful for predicting the return.

Overall, it would be very instereting to further investigate the measure 2 and in general, the prediction of deep learning model for volatility.

‘

	CNN		LSTM	
	val_loss	R^2	val_loss	R^2
Measure 1	897	0.99	10640547	-115
Measure 2	1.9622	**0.3369**	2.7732	**0.2417**
Measure 3	3.5221e-07	-0.0932	3.7542e-07	-0.0538

Table 4. Performance of each measure on *beanpulp*

To validate our findings, we experimented our model on the another futures contract: beanpulp. The beanpulp and ironore is not closely related product. Thus this replication is meaningful. the results are summarized in table (4). Compared to table (3). The pattern is similar. The strong positive stong signal appeared again in the measure 2 case.

4. Conclusion and Future work

We have investigated both of CNN and LSTM on future price, volatility and return prediction. The results on the volatility prediction seem promising and may be worth further investigation. However, our experiments indicate that the CNN and LSTM may not work well or good enough for price prediction and return prediction. Due to limited time and available computational resource, we didn't conducted a comprehensive hyperparameter tunning or network structure exploration. We think that it would be worthwhile to further experiment in these two directions and improvements on our current results are highly likely.

References

[1] Stanford university cs231n: Convolutional neural networks for visual recognition.

[2] Q. V. Le. A tutorial on deep learning. *Google Brain, Google Inc.*, 2015.

[3] N. Maknickiene and A. Maknickas. Application of neural network for forecasting of exchange rates and forex trading. *7-th International Scientific Conference*, 2012.

[4] N. Maknickiene, A. Rutkauskas, and A. Maknickas. Comparison of stock prediction using different neural network types. *International Journal of Advanced Engineering & Application*, 1:157–160, 2011.

[5] N. Maknickiene, A. Rutkauskas, and A. Maknickas. Investigation of financial market prediction by recurrent neural network. *Innovative Technologies for Science, Business and Education Vilnius: Vilnius Business College*, 2:3–8, 2011.

[6] P. Patel and T. Marwala. Forecasting closing price indices using neural networks. *Systems, Man and Cybernetics SMC '06, IEEE International Conference*, pages 2351–2356, 2005.

[7] A. Schmidt and Z. Bandar. A modular neural network architecture with additional generalization abilities for large input vectors. *Third International Conference on Artificial Neural Networks and Genetic Algorithms*, 1997.

[8] J. Tome and J. Carvalho. Market index prediction using fuzzy boolean nets hybrid intelligent systems. *HIS '05, Fifth International Conference*, 2005.

[9] J. Wang and J. Leu. Stock market trend prediction using arima-based neural networks. *IEEE International Conference*, 4:2160–2165, 1996.

[10] T. Yamashita, K. Hirasawa, and J. Hu. Application of multi-branch neural networks to stock market prediction neural networks. *IJCNN '05, Proceedings IEEE International Joint Conference 4*, pages 2544–2548, 2006.