Deep Learning and the Cross-section of Stock Returns: Neural Networks Combining Price and Fundamental Information

Bo Zhou[1]

This version: March 1st , 2019

Abstract

Traditional empirical finance research uses hand-engineering and trial-and-error to look for the anomalies in the cross-section of stock returns. In this paper, we take advantage of deep learning and utilize both the price and fundamental information to separate stocks' winners from losers. For the first model, we used a 2-layer Long Short-Term Memory (LSTM) neural network with past 80 days' return information as inputs to predict the next day's return and find a before-trading-cost monthly return of 29.58% with a t-statistic of 26.81. The return of the stocks shows a strong short-term reversal pattern. For the second model, we design a novel 2-layer LSTM and Multi-layer Perceptron (MLP) hybrid neural network and utilize the monthly return data and annual accounting data to predict the returns in the next month. We achieve a monthly return of 2.37% with a t-statistic of 8.97 before trading cost from 1993 through 2017. We use TAQ intraday data to explicitly estimate the trading cost and find that profits of the daily trading strategy in the first model turn negative. However, the trading strategy utilizing both the price and fundamental information in the second model keeps significantly positive with a monthly return of 1.57% and t-statistic of 6.03 after the trading cost. We show that the 2-layer LSTM and MLP hybrid model performs better than the MLP-only and the hand-engineering momentum and short-term reversal double sort trading strategy.

Keywords: Deep Learning, Machine Learning, Recurrent Neural Network(RNN), Long Short-Term Memory(LSTM), Deep Neural network (DNN), Multi-layer Perceptron (MLP), Momentum, Short-Term Reversal, Random forest, XGBoost, Anomalies, Trading Strategy, Industry portfolio, Trading cost, Transaction costs, Proportional Effective Spread (PES)

JEL classification: G10; G12

[1] email: zhoubo97@gmail.com, Address: Rutgers Business School, 1 Washington Park, Newark, NJ 07102, USA

1 Introduction

Artificial Intelligence (AI) is fundamentally changing and will continue to change the human society. Just like people start using electricity to replace steam engine more than a century ago, people are using AI to transform almost every aspect of our life: transportation, healthcare, manufacturing, communication, entertainment, and finance. Deep learning is one of the most successful algorithms in AI and machine learning. Why does deep learning work? If we have an image that consists of 1 million pixels and we need to detect a car in the image, the traditional technique uses hand-engineering to design features like wheels and car bodies and develop an algorithm to recognize the features. However, the result is usually not ideal. On the contrary, deep learning can utilize vast pool of data and employ a general purpose algorithm such as Convolutional Neural Networks (CNN) to automatically learn the features from the data and achieve results that far exceed the traditional techniques. The key difference is that the first approach is learned by a human from selected data, while the second approach is learned by a machine with the entire data pool. In the second approach, the machine learns the features by themselves and the learned features are usually better than the human designs. Likewise, for decades, financial economists used hand-engineering to search for patterns in the price and fundamental information of stocks and found a zoo of anomalies Cochrane (2011). Because the anomalies are designed by a human, they usually have good explanatory value but may be far from efficient. For example, the well-known momentum effect (Jegadeesh and Titman (1993)) uses the summation of the returns from month(t-12) to month(t-2) prior to the portfolio formation to predict the next few months' returns. Although the summation of returns is a good proxy for stocks' past year performance, it ignores the detail time-series structure within the 11 months before the portfolio formation. Novy-Marx (2012) found that, for US stocks, it is mostly month(t-12) to month (t-7)'s returns that determine the month(t)'s return. Deep learning algorithms can give us a design that could be better than a linear combinantion of the past returns with equal weights. Moreover, given this zoo of anomalies, a natural idea is to combine the different anomalies and generate a score to separate winners from losers. For example, Piotroski (2000) uses nine accounting signals to generate a F-score and find the F-score is strongly related to the cross-section of stocks' returns. Asness, Frazzini and Pedersen (2017) use 16 accounting signals that relate to firm's profitability, growth and safety to build a "Quality Minus Junk" factor and find that the "quality" stocks' outperform the "junk" stocks. In these practices, the researchers use equal weight on these signals, which is probably not most efficient. Again deep learning could do a better job in these tasks.

Deep learning is one of the machine learning algorithms. It generally refer to neural networks with many layers. The procedure of deep learning is not complex at all. It has many similarities to the linear regression. Actually we can use deep learning method to do linear regression if we set a specific objective function and a one-layer neural network structure. Table 1 illustrates a simple comparison between deep learning and linear regression. They both generate a function $\hat{y} = \mathrm{f(x)}$ of the inputs or independent variables x and predict a fitted value $\hat{y}$. The function $\hat{y} = \mathrm{f(x)}$ relies on many weights, which can be seen as a set of "knobs" (Lecun, Bengio and Hinton(2015)). They both adjust the knobs to minimize the objective function. The objective function simply measures the average distance between the dependent variable y and predicted $\hat{y}$ of all training samples.

	Deep learning	Linear Regression
Objective function	Minimize the Cost function, which is equivalent to measure the distance between the real output y and predict output $\hat{y}$. The sum of squared error $\sum_{j=1}^{m}\left(y_j - \hat{y}_j\right)^2$could serve as a cost function.	Minimize $\sum_{j=1}^{m}\left(y_j - \hat{y}_j\right)^2$, where m is number of data points. y_j and $\hat{y}_j$ is jth data point of the dependent variable and its predicted value.
Training Data	m data points, each point consist of input x, and output y, both x and y could be multi-dimensional	m data points, each point consist of independent variable x, and dependent variable y, both x and y could be multi-dimensional
The outcome of learning or regression	A set of weights. Using the weights we can generate a *nonlinear* function $\hat{y} =$ f(x)	A set of weights. Using the weights we can generate a *linear* function $\hat{y} =$ f(x)
Method to minimize the cost function	Numerical	Usually analytical
Number of weights	If y is 1-dimensional and x is n-dimensional, the number of weights could be much larger than n+1, usually in the order of O(n^2)	If y is 1-dimensional and x is n-dimensional, the number of weights is n+1

Table 1, a simple comparison between deep learning and linear regression.

An important difference between the deep learning and the linear regression is the nonlinearity of the fitted function $\hat{y} =$ f(x) in deep learning. For example, if we have either red or blue dots in a plane and it forms a pattern. Given enough training examples, we need to use a dot's 2-dimensional coordinate, x and y, to predict its color. Or equivalently, we want to draw a border to separate the blue and red dots.

A possible result of deep learning and linear regression is illustrated in Figure 1. Deep learning can learn a much complex pattern while linear regression (such as logistic regression) can only draw a straight line. This is because the prediction in linear regression is only a linear combination of 2 independent variables: the coordinates x and y. In contrast, it has been proved that deep neural network can simulate any function of the input variables.

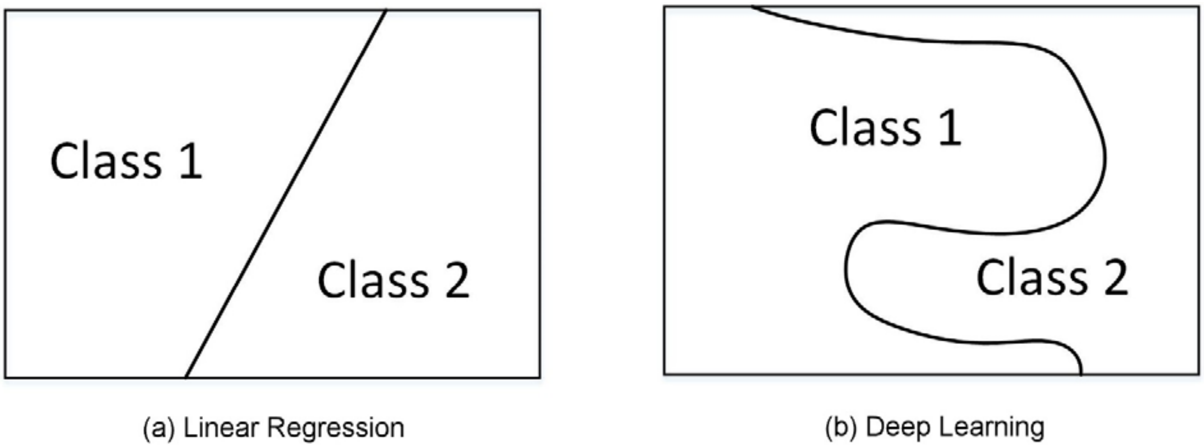


Figure 1, the comparison between deep learning and linear regression. Suppose we have either red or blue dots in a plane and the dots form a pattern. Given enough training example of the red and blue dots, we need to use a dot's 2-dimensional coordinate to predict its color. Deep learning can learn a much complex pattern while linear regression can only draw a straight line as the boundary of red and blue dots.

Generally, there are two kinds of prediction problem in finance literature: cross-sectional and time-series. In deep learning, we use Feed-Forward Deep Neural Network (FF-DNN) or *multilayer perceptron* (MLP) to perform the cross-sectional prediction and the Recurrent Neural Network (RNN) to do the time-series prediction. Especially, an RNN structure called Long Short-Term Memory (LSTM) proposed by Hochreiter and Schmidhuber (1997) is the *de facto* standard model for RNN. LSTM model solves the vanishing or exploding gradient problem of RNN, so it can deal with long-term dependencies in the time-series. For cross-sectional prediction, we feed a set of lagged firm characteristics to the neural network and let it learn from the training data and predict the next period's return. For time-series prediction, we feed the past n days (or months) returns sequentially to the LSTM neural network and let it predict the return of next day (or next month). It is straightforward to apply the MLP and LSTM neural network model to the financial datasets and previous works such as Takeuchi and Lee (2013), Krauss, Do and Huck (2016) and Fischer, Krauss (2017) found more than 45% annualized profit in the US stocks. However, they either did not consider the trading costs at all (Takeuchi and Lee (2013)) or fix the round-trip trading cost at 0.1% level for all the stocks in all years (Krauss, Do and Huck (2016) and Fischer, Krauss (2017)).

Although Krauss, Do and Huck (2016) and Fischer, Krauss (2017) considered only the S&P 500 stocks, the 0.1% round-trip trading cost is still too low, especially in the year of 1990s and early 2000s. This could lead them to over-estimate the after-trading-cost profit of their strategy.

In this paper, we use two models and perform three tests. In the first test, we feed the past 80 days return to a 2-layer LSTM neural network and predict the next day's return. The neural network will give each stock probabilities to outperform (or underperform) the next day. We long (short) the 10% stocks with the highest probability to outperform (underperform) in the next day and find a 29.58% monthly equal-weighted return and a 19.34% monthly value-weighted return before the trading costs. The neural networks learned a strategy that shows a strong short-term reversal pattern and a modest momentum pattern (see detail in Figure 7). We then use the TAQ intraday trade and quote data to explicitly estimate the Proportional Effective Spread (PES) for each stock in each month and use it as the proxy for the trading cost. The 2-layer LSTM strategy's equal-weighted profits turn negative after the trading cost. The value-weighted profits become much smaller but are still positive for 5,10 and 21 days holding periods. The first test shows that, with the help of deep learning, we can achieve a monthly return of 29.58%, which can be translated into 2141% annually. However, this profit is only on paper. The reason why we can achieve so high a profit is that many of the returns in CRSP database are inferred from the mid-point of the bid and ask quotes rather than the real transactions. The bid-ask spread could be huge before the year of early 2000s, especially for stocks with small size and high idiosyncratic volatilities. If a stock's bid price is $9 and the ask price $11, the mid-point is 10. The neural network, for example, suggests the stock's mid-point probably go to $10.5 in the next day. We suppose this did happen. This was a 5% daily profit on paper. But I could only buy the stock at the price of $11 today at the ask price, which would translate into a loss tomorrow. From the first test, we can see that, although we can data-mining profits as high as 2141% per year, without considering the trading cost and market friction, this kind of results probably have less value.

In the second test, we feed the past 80 day's returns of 17 industries to the 2-layer LSTM neural network, which is same as the neural network in the first test, and let it predict the next day's return. We follow (Fama and French 1997) to form the 17 industry portfolios and long and short three entire industries according to their probabilities to outperform and underperform. This time, the neural network learned a momentum strategy without a short-term reversal pattern, which is consistent with Moskowitz and Grinblatt (1999)'s findings. In the short-term reversal pattern in the first test (as shown in Figure 7), the winner stock's return has a sharp decline in the few days before the portfolio formation. However, in the second test, the winner industry's return has a sharp increase in the few days leading to the portfolio formation (see detail in Figure 9).

Previous literature either use the LSTM neural network to deal with the time-series data or use the MLP neural network to make a cross-sectional prediction. To our knowledge, there has not been a work that combines the LSTM and MLP neural network. In the third test, we design a novel 2-layer LSTM and MLP hybrid neural network structure (see detail in Figure 4 of Section 4.2) that combine the cross-sectional anomalies and the time-series price information. The strategy achieves a monthly equal-weighted return of 2.49% with a t-statistic of 12.63 and a monthly value-weighted return of 1.56% with a t-statistic of 5.00 from 1981 through 2017, before the trading costs. We compare this trading strategy with a MLP-only model and a momentum and short-term reversal 5x5 double sort trading strategy. Although they have similar returns before the trading cost, the LSTM-MLP hybrid model has a much lower turnover ratio. As a result, the return of the LSTM-MLP hybrid model is still very significant after considering the trading cost, with a monthly equal-weighted return of 1.57 and a t-statistic of 6.03 from 1993 through 2007. In contrast, the profit of the MLP-only model becomes much lower and the profit of the 5x5 double sort strategy turns insignificant after considering the trading cost. We also combine the LSTM-MLP hybrid model with other machine learning algorithms such as random forest and XGBoost. The ensemble learning model that combines the three model achieves an even higher before-trading-cost return. Lastly, we perform an regression between the monthly return time-series of different models. We show that, although the trading strategy learned by the LSTM-MLP hybrid model shows a pattern that is similar to a momentum and short-term reversal combined trading strategy, it trades very different stocks from the human engineered momentum and short-term reversal 5x5 double sort strategy.

Our work contributes to the current literature in four aspects. Firstly, we apply the 2 layer LSTM neural network to entire cross-section of the US stocks (only excluding stocks with price less than $5 and size less than 1% percentile). We show that, with the deep learning algorithm, the profits of daily trading strategy could be as high as 2141% annually. However, the profits turn negative or become very small after considering the trading cost. Secondly, we are also the first work that use TAQ database to accurately estimate the trading costs for the deep learning trading strategy and find the accurate after-trading-cost profits. Thirdly, we apply the 2-layer LSTM neural network to 17 industry portfolios and the neural network automatically learn an industry momentum pattern without a short-term reversal effect. Lastly, the most important contribution of this paper is that we designed a novel neural network structure that combines the LSTM and MLP neural network, which achieves a significant profit after considering the trading cost. The hybrid neural network structure gives a solution on combining the cross-sectional and time-series information and preserve the “sequential” information in the time-series.

The remaining of the paper is organized as follows. Section 2 reviews the related literature. Section 3 gives a detailed explanation of the methodology. Section 4 describes the data and the design of the neural networks in the three tests. Section 5 shows the results, in which section 5.1 to 5.3 shows the results of the three tests respectively. Section 6 concludes. In Appendix A, we also provide an in-depth review of the basic neural network structure. We derive the formula for gradient descent and back-propagation and explain the varnishing and exploding problem in RNN. It could serve as introductory material for deep learning and neural networks. In all the following sections, for the notation such us day(t-80), or month(t-12), t always refers to the portfolio formation day or month.

2 Related Literature

The related literature falls into two categories: the financial anomalies literature and the "deep learning in finance" literature.

There is a rich literature in the first category. For example, Jacob (2015) reviewed 100 anomalies and found that the strength of many anomalies is strongly related to the time-varying market sentiment. Harvey, Liu and Zhu (2016) reviewed 313 factors and proposed a new hurdle of t-statistic 3.0 for a newly found anomaly. Also, as we mentioned in the introduction, Piotroski (2000) and Asness, Frazzini and Pedersen (2017) employ various accounting signals and combine them into a quality score to predict stocks' returns.

A recent work by Light, Maslov and Rytchkov (2017) use a technique called Partial Least Square (PLS) to combine 26 different anomalies. For the first step, they run a cross-sectional regression of the return R_t on each of the lagged normalized firm characteristics X_{t-1}^{a}. If there are 3000 stocks in year t, the regression will have 3000 data points and R_t and X_{t-1}^{a} are both 3000 dimensional (We use 3000 as an exemplary number in this paragraph). There are 26 anomalies and they run 26 such regressions and obtain 26 slopes λ_t^a, a=1,2,3….26, in each year. Regression on a normalized vector in the 3000-dimensional space is equavalent to doing a projection of the vector R_t on the normalized vector. In the cross-sectional 3000 dimensional space, the 26 factors (each factor is 3000 dimensional) forms a 26-dimensional subspace and the 26 slopes λ_t^a of R_t, a=1,2,3….26, are equivalent to the coordinates of the vector R_t in the 26-dimensional subspace. Now we have a 3000 dimensional space and a 26 dimensional subspace. For year t+1, let us focus on the 26 dimensional subspace. For each stock i at year t+1, we have 26 coordinates of the stock in the 26 dimensional subspace, which is the firm characteristics at year t. In order to predict R_{t+1}, we need to know its coordinate in the 26-dimensional sub-space, λ_{t+1}^a. The authors assume that $\lambda_{t+1}^a \approx \lambda_t^a$. For the second step, they run a regession of each stocks' 26 characteristics on the 26 λ_t^a, a=1,2,3….26. The regession has only 26 data points and they run 3000 such regressions. Effectively, they are doing a projection of each stock's vector (represented by the 26 coordinates for each stock i) to the direction of R_t (represented by the 26 coordinates of λ_t^a, a=1,2,3….26), within the 26-dimensional subspace. The stocks with largest projection (the slope of the regression) on the direction of R_t will have the highest predicted return at time t+1. The PLS approach gives the 26 anomalies different weights but the it is still a linear combination. For some of the 26 anmalies, there are too many missing values, which makes the training sample size a lot smaller. In this paper, we employ 15 of the 26 anomalies that have the fewest missing values.

In the second category of the "deep learning in finance" literature, the paper available is limited, especially in finance journals. Of course, there is a vibrant literature in general purpose deep learning and neural networks. Lecun, Bengio and Hinton (2015)'s paper published in *Nature* gives an excellent review of the basic concept and application of deep learning. Heaton, Polson and Witte (2016) provides a brief overview of the application of deep learning in finance. The papers that most related to our work are Takeuchi and Lee (2013), Krauss, Do and Huck (2016) and Fischer, Krauss (2017).

Takeuchi and Lee (2013) is probably the first paper that applies deep learning techniques in predicting the stock returns. They obtain an annualized profit of 45.93% in the 1990-2009 period before the trading cost. Essentially they treat the time-series price information as parallel firm characteristics. They feed 33 variables to an MLP-like neural network and predict next month's return. The 33 variables consist of 12 monthly return from month(t-2) to month(t-13), 20 daily returns in the month(t-1) and a dummy variable indicating whether the portfolio formation month is January. Since they feed the 20 daily return in the month(t-1) to the neural network, their strategy is probably a momentum and short-term reversal combined strategy. The difference between their neural network model and a classic MLP is that they follow Hinton and Salakhutdinov (2006) and use a Restricted Boltzmann Machine (RBM) in each layer. RBM is an auto-encoder that used in unsupervised training. They use the RBM to pre-train each layer and used the obtained weights as the initial values for the MLP training.

Krauss, Do and Huck (2016) apply three deep learning techniques: deep neural network (DNN) which is equivalent to MLP, gradient boosted trees (GBT) and random forest (RAF). They feed the past 250 days return to the three neural network and predict the next day's return. When longing (shorting) the top (bottom) 2% of the S&P500 stocks, they achieve a daily return of 0.45% before the trading cost. As we mentioned in the introduction, they use a constant round trip trading cost of 0.1% for all stocks in all years. Applying a trading cost of 0.1% to both the long and short portfolio, the after-trading-cost profit is 0.45%-0.2%=0.25%/day. Fischer, Krauss (2017) perform a similar procedure to the S&P500 stock universe with LSTM neural network. They achieve a daily profit of 0.46% before the trading cost. Again, a constant round-trip trading cost of 0.1% is used and they estimate a 0.46%-0.2%=0.26% daily profit after the trading cost.

3 Methodology: An Overview of How We Train the Neural networks and How We Trade With the Strategy

We will use the settings of the first test as an example to give a detailed overview of how we train the neural network and how we trade. In the first test, we use the past 80 days' daily return as inputs to predict the next day's return. After the training, we will obtain a set of weights and generate a predicted $\hat{y}$, which is a vector consisting of a stock's probability to outperform and underperform in the next day. According to the two probalilities, we will long (short) the 10% of the stocks with the highest probability to outperform (underperform).

In the case of linear regression, we will have 80 slopes and one intercept in the prediction function $\hat{y} = \mathrm{f(x)}$ if we have 80 inputs. However, in the setting of the first test, we will have 30855 weights. The 30855 weights are not only applied on the 80 inputs but also on the intermediate hidden units, which is equivalent to latent varibles in econometrics. If we have 5 quintiles for the next day's return, $\hat{y}$ would be a probability vector like [0.1, 0.1, 0.1, 0.2, 0.5], which indicates that we will have 10% probability to fall into the bottom 20% quintile and 50% probability to fall into the top 20% quintile. If the real next day's return is in the top 20%, the real y would be [0, 0, 0, 0, 1]. We use the a objective function like $\mathcal{L}(Y, \hat{Y}) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{5} y_i^{(j)} \log(\hat{y}_i^{(j)})$ to measure the distance between y and $\hat{y}$ across all training samples, where m is the number of training samples and $y_i^{(j)}$ and $\hat{y}_i^{(j)}$ are the i th dimenson of the y and $\hat{y}$ probability vector of the training sample j. The 5 in the sumation indicates we have 5 dimensions in the output y and predicted vector $\hat{y}$. Where does this objective function come from? Recall that the probability mass function of bernuolli distribution is $P(y|\hat{y}) = \hat{y}^y\,(1-\hat{y})^y$. Since $\sum_{i=1}^{5}\hat{y}_i = 1$, $P(y|\hat{y}) = \prod_{i=1}^{5}\hat{y}_i^{y_i}$ can serve as the probability mass function of the "multinuolli" distribution for one data point if we have 5 outcomes. To minimize $-\sum_{i=1}^{5}\log(\hat{y}_i^{y_i})$ is equivalent to maximize $\sum_{i=1}^{5}\log(\hat{y}_i^{y_i})$ or $\prod_{i=1}^{5}\hat{y}_i^{y_i}$. So minimizing the distance between y and $\hat{y}$ using this objective funcion is equivalent to a maximum likelihood problem.

We use a rolling window to convert the raw CRSP data into the training data. Specifically, if we have 81 consecutive days' return of a stock, we will have one sample. The first 80 days' return is the input vector x and the 81st day's return is the output y. If we have 82 consecutive days of return, we have two samples. The second sample is $(\text{return}_2, \text{return}_3, \ldots, \text{return}_{81}|\ \text{return}_{82})$. To avoid the look-ahead bias and keep the weights "fresh" (trained from recent data), we employ a learning and predicting procedure as follows. Before each month(t), we use the previous 36 months' data to train the weights of the neural network. The weights are shared by all stocks and will be fixed in the portfolio formation month(t). In each trading day of month(t), we use the shared weights and each stock's 80 inputs to predict the probability vector $\hat{y}$. In the rolling 3-year training window, we have roughly 750 trading days and if we have 3000 stocks, we will have 2,250,000 sample points to train the weights in each month. In the beginning of the month(t+1) , we roll the 3-year training window for one month and train the weight again and use the new weights in month(t+1).

The training or gradient descent process can be conceptually illustrated in Figure 2. The cost function $\mathcal{L}(Y,\hat{Y}) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{5} y_i^{(j)}\log(\hat{y}_i^{(j)})$ is a function of the 2,250,000 data points and the 30855 weights. We will adjust the weights to minimize the cost function. We start with a set of random weights and calculate the derivative of the weights, $\partial\mathcal{L}(Y,\hat{Y})/\partial w$, at these random weight values. The values of the derivatives will give us a direction and a speed to adjust the weights. For example, at the the left side of Figure 2, the negative derivative tell us to increase the weights at a certain speed. In each step, we change the weights a little bit and calculate the derivatives again. We can think of the cost function as a hill. In real case, the multi-dimensional derivatives will point to the fastest direction to go down the hill. We will follow this direction and finally reach the lowest point. In practice, millions of training data points will exceed the limit of computer memory, so we cannot calculate the the derivative at once. Instead, we will split the 2,250,000 data points into 225 mini-batches. Each mini-batch consists of 10000 data points. As a result, we split the one big step of gradient descent into 225 small steps. The 10000 data points in one mini-batch is a sample of the 2,250,000 data points. So the direction of grdient descent in one mini-batch may have small perturbation and may not precisely follow the fastest direction down the hill. However, statistical rule guarantees that the small perturbation will cancel out each other and the 225 small steps will very precisely follow the fastest direction.

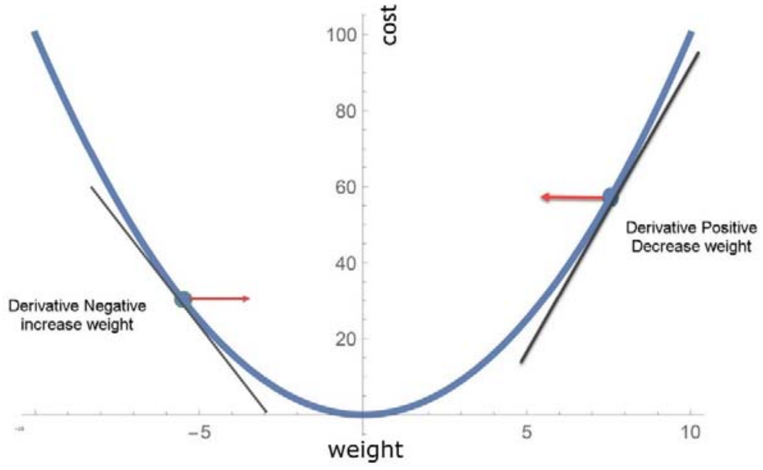


Figure 2, the illustration of the gradient descent in training a neural network.

We also need to point out that the training of a neural network is a trial and error process. We need to adjust the hyperparameters such as the number of hidden units, the number of layers, the number of training epochs and the learning rates etc., to let the cost function go down the hill faster. If there are 10000 combinations of the hyperparameters, our result is only one of them. Finetuning the hyperparameter is a necessary process in any neural network training. If a set of optimal hyperparameters is found, adjusting them a little will not affect the result very much. Of course, a better approach is to try many combinations of the hyperparameters and see the distribution of the results. But this is not feasible given today's computation capacity. With the fastest GPU powered computer available to us, the training process with one set of hyperparameters will cost us more than 10 hours. This also means that we probably have not reached the global optima. Our result is only a sample of the sub-optimal results and still have room to improve.

4 The Description of Data and The Design of the Neural Networks in The Three Tests

4.1 Data and Software

The firm-level data in this research comes from four databases. The first and second of these is the Center for Research in Security Prices (CRSP) daily and monthly database. For CRSP data, we only include the stocks with share code (SHRCD) 10 and 11 and exchange code (EXCHCD) 1, 2 and 3. These stocks include US companies trading in NYSE, AMEX and NASDAQ and exclude the ADR and REIT stocks. We DO NOT exclude the financial firms with standard industrial classification (SIC) code between 6000 and 6999. The Data covers the period from 1968 through 2017.

The third database is COMPUSTAT annual report database. We use the modern accounting data from 1970 to 2017. We follow Fama and French (1993) and Novy-Marx (2013) in calculating the book value of a firm. Namely, we first calculate shareholder's equity plus the deferred taxes and investment tax credits and then minus the value preferred stocks. The shareholder's equity is obtained from (1) Compustat data item SEQ, if available, or else (2) common equity (we use CEQ if available, or book value per share BKVLPS times common shares outstanding (CSHO) for the common equity) plus the value of the preferred stocks, if available, or else (3) the total asset (AT) minus the total liability (LT). The deferred taxes and investment tax credits come from Compustat TXDITC if available, or else TXDB plus ITCB. The value of preferred stock is redemption value (PSTKR) if available, or else liquidating value (PSTKRL), or else carrying value (PSTK). If the values of TXDITC, TXDB, ITCB and PSTKR, PSTKRL, PSTK are all missing, they are set to be zero. We require the firm to have at least two year's history in Compustat to be included in our sample.

The fourth database is the TAQ intraday trade and quote database from 1993 to 2017. We use TAQ database to estimate the trading cost of the deep learning strategy. Because the intraday database is very large, we only use the data of the last trading day of March and September in each year to estimate the trading cost.

In the first and second test, we require the ending price at day(t-1) be above $5 and the size at day(t-1) be above 1% percentile of all the stocks, where day(t) is the portfolio formation day. In the third test of 2-layer LSTM and MLP hybrid neural network, we require the ending price at month(t-1) be above $5 and the size at the end of month(t-1) be above 1% percentile of all the stocks. These requirements exclude approximately 25% of the stocks from the sample universe.

It is possible that a stock's observation in a certain trading day is missing. We call it a "hole" in a stock's return sequence. For the first test, we require that there is at most 5 "holes" in the 80-returns input sequence. This requirement does not affect the sample size very much. From 1968 to

2017, we have 64612395 data points. After applying this “at most 5 holes” requirement, the sample drops to 61220503 data points. If we apply the “no hole” requirement, the sample drops to 60192060 data points. For the third tests, we require that there is at most 1 hole in the 12 monthly return and volume sequence.

For the return sequence and the firm characteristics inputs, we use a z-score for each signal instead of the raw values. The z-score is calculated as

$$z\_variable_{it} = \frac{variable_{it} - cross\ sectional\ mean(variable_t\ at\ day_t\ or\ month_t)}{stdev(Variable_t\ at\ day_t\ or\ month_t)} \tag{10}$$

For MLP and LSTM neural networks, we use the scripting language Python and the deep learning framework Keras with Tensorflow backend. For random forest classifier, we use the machine learning library Scikit-Learn. For XGBoost, we use the python application programming interface of XGBoost provided by its official website. We employ several P3 instances with GPU in the Amazon Web Service (AWS) cloud in training the neural networks.

4.2 The Design of the Neural Networks in The Three Tests

A detailed overview of the basics of the RNN and LSTM neural network can be found in Appendix A. The description of the neural network design in this section is based on the terminologies and formula in Appendix A. For the first and second test, we employ a 2-layer LSTM model, which is illustrated in Figure 3.

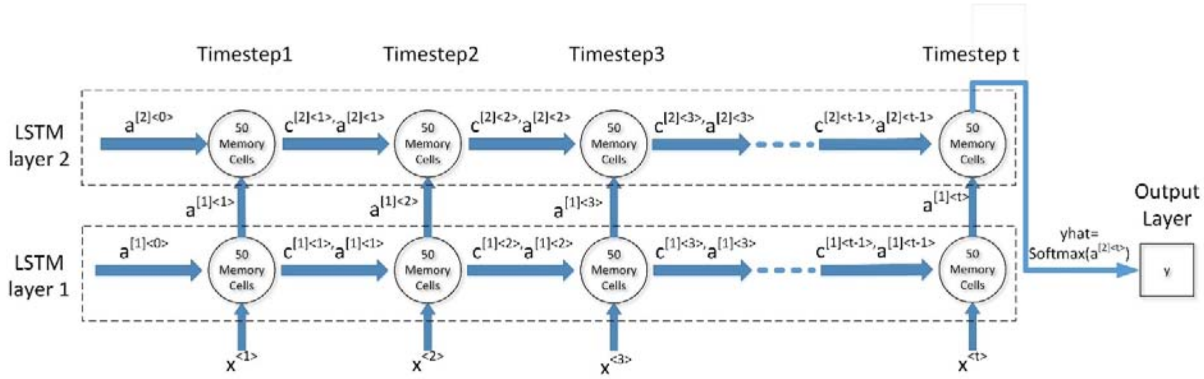


Figure3, the 2-layer LSTM neural network in the first and second test.

In Figure 3, for example, the notation $a^{[1]<2>}$ represents the output of the LSTM layer [1], timestep <2>, in which the square bracket "[]" represents the layers and the angle bracket "<>" represents the timesteps. In each timestep of the first LSTM layer, $a^{[1]<i>}$ is not only fed to the next timestep but also fed to the same timestep of the second LSTM layer. We have 80 timesteps in both layer 1 and layer 2. In each timestep, we have 50 memory cells. As a result, $a^{[1]<i>}$ and $a^{[2]<i>}$ are both 50-dimensional vector. The input of the first layer, $x^{<1>}$ $to\ x^{<t>}$, is simply the z-score of the returns of past 80 days, in which each timestep only receives a scaler $x^{<i>}$. The input of each timestep of the second layer is a 50-dimensional vecto/r. For example, the 50 dimensional vector $a^{[1]<80>}$ is the input of timestep 80 of layer 2. The output of the last timestep of the second layer is fed to the output layer and generate a 5-dimensional predicted $\hat{y}$ through a softmax function.

Following Fischer and Krauss (2017), we can calculate why there are 30855 weights in the neural network of the first test with the parameter settings we have described above. Again please refer to the introduction of LSTM neural network in Appendix A if you have questions in the following calculations.

For an LSTM layer, the number of weights is

$$\text{number of weights of a LSTM layer} = 4(h(i+h)+h) \tag{11}$$

where, h is the number of the hidden units. i is the dimension of the input in each timestep. In our case of the first LSTM layer, h=50 and i=1 because we feed one return at each timestep in the first layer and we have 50 hidden units. In LSTM, we have three gates and we need to calculate the candidate memory cell value $\tilde{c}_t$. These 4 vectors are all h dimensional. The formula to calculate them is exactly the same. For example, $\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c)$, in which, $\tilde{c}_t$ is h dimensional and $[a_{t-1}, x_t]$ is h+i dimensional. So W_c must be h*(i+h) dimensional. b_c is h dimensional. For each gate, we thus have $h(i+h)+h$ weights. We have 4 of these equations so we have $4(h(i+h)+h)$ weights for one LSTM layer. For the first LSTM layer we have 4*(50*51+50)=10400 weights. For the second LSTM layer, h=50 and i=50 because the input of each timestep of the second LSTM layer is the 50-dimensional output $a^{[1]<i>}$ of the same timestep of the first LSTM layer. So we have 4*(50*100+50)=20200 weights. In the output layer, we have 5 imbalanced softmax units. Each units have 50 inputs from the last timestep of second LSTM layer and a bias. So it is 5*(50+1)=255 weights. Totally, we have 10400+20200+255=30855 weights in the 2-layer LSTM neural network of the first and second test.

For the third test, we want to combine the time-series price information and cross-sectional accounting information. We design a hybrid neural network model that utilizes the features of both the LSTM and MLP neural network. The structure of the model is illustrated in Figure 4. The 2-layer LSTM in the upper right corner of Figure 4 is very similar to the 2-layer LSTM model in the first and second test. The input x for the LSTM layer 1 is the z-score of the returns of the past 12 months. So the LSTM layer 1 and 2 both have 12 timesteps.

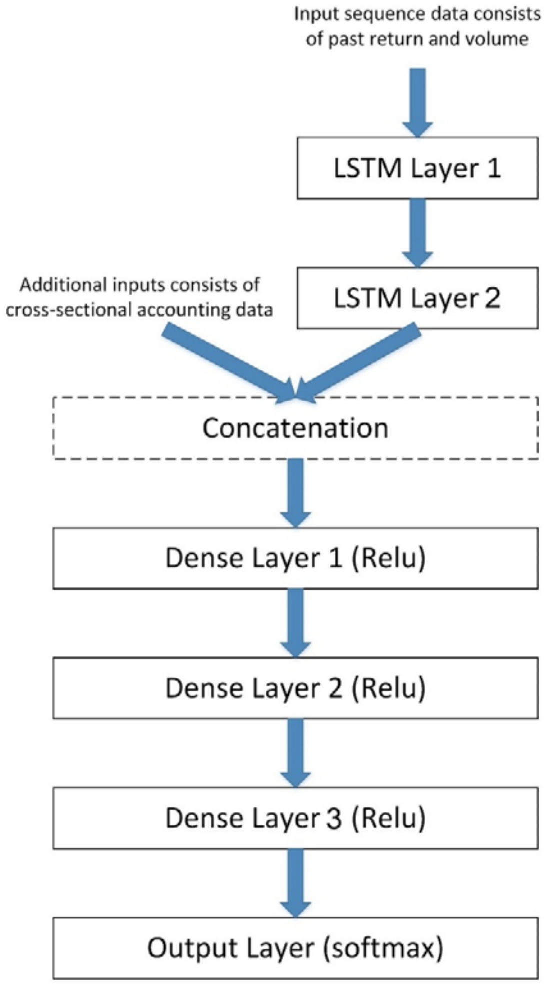


Figure 4, two-layer LSTM and MLP hybrid neural network structure used in the third test (section 5.3). The input x for the first LSTM layer on the top right corner is the z-score of the returns of the past 12 months. At the output of the second LSTM layer, a 30-dimensional vector consisting of the past price information will concatenate with a 15-dimensional vector consisting of the latest available accounting signals of a stock. The combined 45-dimensional vector will be fed to a 4-layer (128-64-32-5) MLP and generate a predicted $\hat{y}$ for the next month.

At timestep 12 of the second LSTM layer, a 30-dimensional vector $a^{[2]<12>}$ is fed to a concatenation layer. At this layer. The 30 dimensional $a^{[2]<12>}$ will concatenate with a 15-dimensional vector that consists of 15 lastest available accounting signals. The combined 45-dimensional vector will be fed to a 4-layer MLP and generate a 5-dimensional predicted output $\hat{y}$ for the next month. The 4-layer MLP has a structure of 128-64-32-5.

Here is a question: what if we directly concatenate the 12 returns to the 15 firm characteristics and feed them parallelly to an MLP-only neural network? What difference does our new neural network structure make? The key is that the 30-dimensional vector output by the second LSTM layer keeps the sequential information of the returns. This sequential information is valuable in improving the return prediction. In section 5.3.3, we will compare the LSTM-MLP hybrid model and the MLP-only model and show the advantage of the hybrid model.

To exclude the seasonal effect, we use the annual financial report for firm characteristics. To make sure that the accounting data is available at time t and avoid the look-ahead bias, we require a 6 month gap between the COMPUSTAT observation time DATADATE and the portfolio formation time t. For example, if the output y is in Jan. 2000, the return sequence will be the monthly data from Jan 1999 to Dec 1999. The accounting data will be collected from the latest annual financial report before July. 1st, 1999. The definition of the 15 fundamental signals are from Light, Maslov and Rytchkov (2017)'s 26 anomalies. We choose the 15 signals that have the fewest missing values to maximize the training sample size. The 15 anomalies are defined as follows.

(1) LOGME. Log of market equity at month(t-1). Log(ME(t-1))

(2) BM. Book to market ratio, which is the book equity at least 6 month before portfolio formation time t divided by market equity at month(t-1).

(3) GPOA. The gross profitability. It is defined as

$$\mathrm{GPOA} = \frac{REVT - COGS}{AT}$$

which is (revenue-cost of goods sold)/total assets. Novy-Marx (2013) shows that firms with high gross profitability will earn higher return in next period.

(4) ROE. Similar to GPOA, ROE is traditional measure of the profitability and widely discussed by works such as Fama and French (2006) and Fama and French (2008). We will employ the definition of ROE in Fama and French (2008)

$$ROE = \frac{IB - DVP + TXDI}{BE}$$

in which, IB is income before extraordinary items, DVP is the preferred dividend (if available), TXDI is the deferred tax (if available) and BE is the book equity.

(5) ROA. We define the return on assets as

$$ROA = \frac{IB}{AT}$$

In which, IB is income before extraordinary items and AT is the total assets.

(6) IK. The investment-to-capital ratio. Previous research finds firms that invest too much will end up with lower return in next period. the investment-to-capital ratio is defined as

$$IK = \frac{CAPX}{PPENT}$$

Where the CAPX is the capital expenditure and the PPENT is COMPUSTAT item “property, plant and equipment.”

(7) AG. Asset growth rate, which is defined as

$$AG = \frac{AT_t - AT_{t-1}}{AT_{t-1}}$$

In which, AT_t is the last available total assets and AT_{t-1} is the total assets one year before the AT_t. Previous research showed that firms with higher asset growth rate will have lower return in next period.

(8) IG. Investment growth, which is defined as

$$IG = \frac{CAPX_t - CAPX_{t-1}}{CAPX_{t-1}}$$

in which, $CAPX_t$ is the last available capital investment and $CAPX_{t-1}$ is the capital investment one year before $CAPX_t$. This is another anomaly of investment. Similar to the anomaly in (6), firms with higher investment growth will have lower return.

(9) IA. Investment to asset ratio, which is defined as,

$$IA = \frac{INVT_t - INVT_{t-1} + PPEGT_t - PPEGT_{t-1}}{AT_{t-1}}$$

in which, the INVT is inventory and PPEGT is gross property, plant and equipment. AT is total asset.

(10) ACC. Accruals. The anomaly of accruals is documented by Sloan (1996), which shows that firms with higher accruals will have lower return in next period. the accruals is defined as,

$$\begin{aligned} ACC = [\big((ACT_t - ACT_{t-1}) - (CHE_t - CHE_{t-1})\big) \\ - \big((LCT_t - LCT_{t-1}) - (DLC_t - DLC_{t-1}) - (TXP_t - TXP_{t-1})\big) \\ - DP_t]/AT_{t-1} \end{aligned}$$

In which, ACT is total current assets, CHE is cash and short-term investments, LCT is total current liabilities, DLC is debt in current liabilities, TXP is income tax payable, DP is depreciation and AT is the total assets.

(11) NOA. Net operating assets. Previous research such as Hirshleifer et at. (2004) shows that higher net operating assets will lead to lower return. NOA is defined as

$$NOA = \frac{Operating\ asset - Operating\ liability}{AT_{t-1}}$$

In which,

$$\text{Operating asset} = \text{AT} - \text{CHE}$$

$$\text{operating liabilities} = \text{AT} - \text{DLC} - \text{DLTT} - \text{MIB} - \text{PSTK} - \text{CEQ}$$

where, AT is the total assets. CHE is cash and short-term investments. DLC is debt in current liabilities. DLTT is total long-term debt. MIB is minority interest, PSTK is preferred stock and CEQ is the common equity.

(12) NS. Net Stock issues. Fama and French (2008) shows that firms that issue more shares tend to underperform in next period. NS is defined as

$$NS = \log(\frac{SASO_t}{SASO_{t-1}})$$

where SASO is the split-adjusted shares outstanding SASO=CSHO*AJEX, CSHO is share outstanding and AJEX is cumulative adjustment factor.

(13) EP. Earning to price ratio.

$$EP = \frac{IB}{ME}$$

where IB is income before extraordinary items and ME is market equity.

(14) CP. Cash flow to price ratio.

$$CP = \frac{IB + EDP + TXDI}{ME}$$

where IB is income before extraordinary items, ME is market equity, EDP is the equity's share of depreciation and TXDI is the deferred tax (if available).

(15) LV. Leverage ratio.

$$LV = \frac{AT - CEQ}{ME}$$

where AT is the total assets, CEQ is the common equity and ME is market equity.

5 Results

5.1 The Result Of The 2-Layer LSTM Model With Individual Stocks' Previous 80 Days's Return As Inputs

5.1.1 The Before-Trading-Cost Results Of The 2-Layer LSTM Model With Individual Stocks' Previous 80 Days's Return As Inputs

For the first test, we employ the past 80 days' return as input and use them to predict the next day's return. We implement a 2-layer LSTM model as described in section 4.2. In the beginning of each month, we use the data of the past three years to train the neural network and obtain a set of weights. We fix the weights for one month and predict a stock's probability to outperform or underperform in the next day for each stock in each day. We long (short) the 10% stocks with the highest probability to outperform (underperform) in each day.

In the 1970s there are on average 2000 firms in each trading day. The number of firms peaks in 1996-2000 periods to about 5000 firms and drops steadily after that. We use roughly 750 days' data as the training dataset. So in the 1970s, we have roughly 1,500,000 data points, and in 1996-2000, we have roughly 3,000,000 data points in the training samples. Table 2 shows the returns and the Fama French 3-factor α of the long-short portfolio. The Panel A and B report the results of the equal-weighted portfolio and Panel C and D reports the results of the value-weighted portfolio.

[Insert Table 2 Here]

We follow Jeehadesh and Titman (1993) for the definition of holding period. For example, if the holding period is 5 days and the dollar value of the long-short portfolio is $1, from the first to fifth day, we will long and short stocks with worth of $1/5 and hold them for 5 days. Starting from the fifth day, our long-short portfolio will both have the value $1 and keep the value ever since. Starting from the sixth day, We will unroll the long-short portfolio that we form in the t-5 day and long and short another $1/5. The purpose of the longer holding period is to limit the trading costs. For 5-days holding period, the transaction cost is roughly 1/5 of that of 1-day holding period. For all the holding period, we use the portfolio formation day(t)'s one-day return as the target output y in the training process.

We can see from Panel A of Table 2 that the profit is highest in 1991-2000 periods. The the monthly profits for longer holding periods are much smaller than that of the 1-day holding period. We can see a monthly equal-weighed return of 29.58% with a t-statistic of 26.81 for 1-day holding period in the period from 1971 through 2017, which can be translated into an annualized 2141% profit. It demonstrates how deep learning can successfully pick up return patterns and what level of the profit we can "data mining" from the existing datasets. However, the profit is before the trading cost. Later, we will see the equal-weighted return turn negative after the trading cost. A lot of empirical finance research did not consider the trading cost in their work. Our result is also a caution that no matter how robust the result is, without carefully considering the trading cost, the result probably will have less value.

From Panel B of Table 2, we can see that the returns of the 2-layer LSTM trading strategy on average have low factor loadings except for a -0.672 SMB factor loading for 1-day holding period in the first column. Because the profit of the strategy is too large and there is a positive market beta, the negative SMB beta does not make much difference between the α and the raw return.

From Panel C of Table 2, we can see that the value-weighted return for the entire sample period drops to 19.19%, which indicates that micro and small stocks with higher transaction costs contribute more to the equal-weighted return. The time-series of before-trading-cost monthly equal-weighted return of 2-layer LSTM neural network for 1-day and 5-day holding period is shown in Figure 5. We can see that before 2010 the monthly equal-weighted returns for 1-day holding period are steadily above the 10% level. The returns are particularly high from the late 1980s to the year of 2000. During the market turmoil in the year of 2000 and 2008, we can see spikes in the returns, which indicate the larger market friction and limit to arbitrage in these periods may result in higher before-trading-cost profits.

[Insert Figure 5 Here]

To investigate the effect of stock size on the profit of the 2-layer LSTM trading strategy, we split the stocks into three subgroups according to their size at the end of day(t-1). The breakpoints are the 20% and 50% size breakpoints of NYSE stocks in each day. We define the three size group as micro, small and big stocks. The three size groups form roughly 43%, 28% and 29% of all the stocks. The equal-weighted returns and its corresponding Fama French 3-factor α of the three size subgroup are reported in Table 3 and the time series of 1-day holding period return of the three size subgroups are shown in Figure 6. We use all the stocks in the training process to adjust the weights and use the same set of weights for all three size subgroups. Table 3 shows us that the micro stocks contribute most to the before-trading-cost profit of the 2-layer LSTM trading strategy in all holding periods, in which the 1-day holding period has a profit of 48.09% for the entire sample period. However, we can see that the big stocks' profits are also very significant in all holding period. The differences of the profits among the three size sub-groups are clearly shown in Figure 6.

[Insert Table 3 and Figure 6 Here]

To investigate the source of the profit and the return pattern that the neural network learned, we draw a picture of the average z-scores of the daily returns before and after the portfolio formation day(t). We can see from Figure 7 that there is a strong short-term reversal pattern before and after the portfolio formation day. The long portfolio represented by the solid orange line has a spike of negative returns in the period from day(t-14) through day(t-1). However, the same long portfolio has a slightly higher return than the short portfolio from day(t-80) to day(t-15). We can see that the 2-layer LSTM neural network picks up a momentum and short-term reversal combined trading strategy.

[Insert Figure 7 Here]

5.1.2 The After-Trading-Cost Profits of 2-Layer LSTM Model With 80 Days's Returns Of Individual Stocks As Inputs

Up to now, all the profits we have reported in the first test are before the trading costs. We see in Figure 6 and Table 3 that micro stocks contribute most to the profit of the 2-layer LSTM trading strategy, but the micro stocks also have a higher trading cost because of a higher bid-ask spread. In this section, we will explicitly explore the effect the trading costs on the profits.

We estimate the trading costs directly from the TAQ trade and quote dataset. The TAQ data is available from 1993 through 2017. The pros of this approach to estimate the trading cost is that it is much more accurate than the traditional method using Roll (1984)'s bid-ask bounce model. The cons are (1) The trading cost data is only available after 1993. (2) The amount of data is huge. (3) because of (2), in each year, we can only use a few days' data as a proxy for all the days. For every year, we download all the TAQ trade and quote data for two days: the last trading day of March and September. Although there is only 2 days of data in each year, the dataset from 1993 to 2017 is already 318GB. We combine the TAQ trade and quote dataset and use each trade's price and its last available bid-ask quote to estimate the effective spread. The effective spread is two times the difference between the realized price and the mid-point of bid-ask quote. Previous literature shows that the effective spreads are usually smaller than the quoted spread and are better estimates of the real trading costs. Because it is two times the difference, it represents a round-trip trading cost. The Proportional Effective Spread (PES) is the effective spread divided by the realized price.

$$\text{Proportional Effective Spread (PES)}_{it} = \frac{2 * |Price_{it} - Bid\ Ask\ Midpoint_{it}|}{Price_{it}} \qquad (12)$$

In each March and September, we (1) estimate the PES for each trade. (2) average the PES in each day for each stocks. Then we have a precise PES estimate for each stock in each March and September. (3) To avoid the look-ahead bias, in each year, we use the March data as proxy for the 6 months from April through September and use the September data as proxy for the 6 months from October through next March. Then, we have the PES for every stock in every month from 1993 to 2017. However, this approach still leaves a large portion of the stocks' trading cost missing.

To solve this problem, we (1) use a stock's explicit PES, if it is not missing. (2) estimate a proxy PES with a stock's size and idiosyncratic volatility (IVOL), if the stock's PES is missing in a month. In each March and September, we rank all the stocks in our sample independently into 10 size and 10 idiosyncratic volatility deciles. We estimate the average PES for each of the 10x10 portfolios and use it as a proxy for the PES for each portfolio. The idiosyncratic volatility (IVOL) is defined as the standard deviation of the errors in the regression of a stock's past year daily returns (approximately 250 returns) on the daily Fama French 3-factors. We use the size and idiosyncratic volatility at the end of month(t-1) in the ranking, where month(t) is the portfolio formation month. A minimum of 60 non-missing daily return observations in the past year is required in the regression. When the PES is known, we will also estimate a turnover ratio from our trading strategy in each month and calculate a trading cost for each stock. The average turnover ratio from 1993 to 2017 for the 2-layer LSTM daily trading strategy is about 55.1%.

[Insert Table 4 and Figure 8 Here]

Table 4 provides information about the average PES for each size and idiosyncratic volatility groups for each year from 1993 to 2017. The results in Table 4 is the average of March and September's PES in a year. Table 4 shows that the PES decreases (increase) monotonically against the size (idiosyncratic volatility) ranks for all the years. We can see that there is a big PES gap between the smallest and the second smallest size group. The PES is much higher in the 1990s and early 2000s than in recent years. The PES generally declines with time. However, the PES of 2000, 2008 and 2009 is particularly high compared to their adjacent years, indicating that the trading costs are higher in the period of market turmoil. Figure 8 shows us the average PES for all years for the 10x10 portfolios.

The PES is already a roundtrip trading cost, but the trading costs affect both the long and short portfolios. For example, we long stock A and short stock B in April 1993 and their PES is missing in the TAQ data. Both stock A and B fall into the smallest size decile and the largest IVOL decile, so their PES should be 3.67% (not shown in Table 4 and Figure 8, because the Table and the Figure only show the averages.). The turnover ratio for April 1993 is 65.2%. The profit for forming and liquidating the long-short portfolio will be decreased by 2*3.67%*65.2%=4.79%. If we trade 21 times in a month, we will need a monthly profit of 167% to compensate for the trading costs and will probably lose money in this long-short portfolio for April 1993.

[Insert Table 5 Here]

Table 5 shows the before and after trading costs returns of the 2-layer LSTM trading strategy. Because the trading costs derived from the TAQ database are only available from 1993 to 2017, the before-trading-cost profits for all years are slightly different from the results reported in Table 2, in which the sample covers from 1971 to 2017. Panel A and Panel B of Table 5 report the equal-weighted and value-weighted returns respectively. We can see that, although the equal-weighted returns are much higher than the value-weighted returns in all holding periods before trading costs, the equal-weighted returns are lower than the value-weighted returns after trading costs. In all holding periods, the equal-weighted returns turn negative after the trading cost. However, the value-weighted after-trading-cost profits are still positive and significant for 5, 10 and 21-day holding period.

5.2 The Result Of The 2-Layer LSTM Model With 17 Industry Portfolios' Previous 80 Days's Return As Inputs

Moskowitz and Grinblatt (1999) designed an industry momentum trading strategy. They long the past winning industries and short the past losing industries and find that the industry momentum can explain a significant portion of the momentum effects. In individual stock momentum strategy, we usually sum up the return from month (t-12) to month(t-2) and skip the return in month(t-1) in forming the signal for the long-short portfolio. This is because, for month(t-1), there is a short-term reversal effect. Moskowitz and Grinblatt (1999) find that, unlike individual stock momentum, the industry momentum does not have a short-term reversal effect.

It is interesting to see what return pattern the 2-Layer LSTM learns when we feed the industry portfolio's returns into the neural network. In the second test, we follow Fama and French (1997) and split the stocks into 17 industries. If we treat each industry as a stock, the second test is almost same as the first test except that we have only 17 stocks to choose from. The 17 industries are (1) Food (2) Mining (3) Oil (4) Textile and Apparel (5) Consumer Durables (6) Chemicals (7) Consumables (8) Construction (9) Steel (10) Fabricated Products (11) Machinery (12) Automobiles (13) Transportation (14) Utilities (15) Retail (16) Financials (17) Other. Same as the first test, we also use the past 80 days returns as inputs and predict an industry's probability to outperform and underperform in the next day. According to the probability, we long (short) 3 industies with the highest probability to outperform (underperform). The results are shown in Table 6. The definition of holding period is same as the first test. we can see that for all holding period the equal-weighted monthly returns before the trading cost are very significant.

[Insert Table 6 and Figure 9 Here]

We then plot the z-score of the returns for the long and short industry portfolio in Figure 9. Recall that in Figure 7, when using the individual stock's returns as inputs, the 2-Layer LSTM neural network learns a strong short-term reversal pattern. Figure 9 shows a fundamental difference from Figure 7. Firstly, like Figure 7, Figure 9 has two return peaks in day(t-1). However, in Figure 7, we short the day(t-1)'s winner and long the day(t-1)'s loser. In Figure 9, we long day(t-1)'s winner represented by the orange dotted line and short the day(t-1)'s loser represented by the blue solid line. Figure 9 is a momentum effect without short-term reversal, while Figure 7 is a momentum effect with short-term reversal. The results in Figure 9 confirms Moskowitz and Grinblatt (1999)'s findings.

5.3 Two-layer LSTM and MLP Hybrid Neural Network

5.3.1 The Basic Results of The Before-Trading-Cost Profits of The 2-layer LSTM and MLP Hybrid Model

In the third test, we will try to answer a question: how do we combine the LSTM and MLP neural network and utilize both the price and fundamental information to predict stock's return. As shown in section 4.2, we design a novel 2-layer LSTM and MLP hybrid neural network for this purpose. We use the past 10 year data to train the neural network and use the trained weights to predict the returns in the next month. We long (short) the top 4% of stocks with the highest probability to outperform (underperform). We use top 4% of stocks in this test because we will later compare our results with the often-used 5x5 double sort portfolios in previous literature. The target y in the training is the next month's return, so this is a monthly trading strategy. Because we need 10 years' data to train the neural network and the account data are only widely available after the 1970s, our samples covers the period from 1981 to 2017.

Table 7 reports the before-trading-costs returns and the Fama French 3-factor α of the trading strategy learned by the LSTM-MLP hybrid neural networks. In Panel A of Table 7, we can see that the strategy achieve a before-trading-cost equal-weighted return of 2.49% with a t-statistics of 12.63 for the period from 1981 to 2017, which can be translated into an annual return of 34.33%. The 2-month and 3-month holding period return is 2.05% and 1.69% respectively. The returns decline very slowly after the portfolio formation. The definition of holding period is same as the ones in Jeehadesh and Titman (1993) and the first test. Panel A of Table 7 also report the result for the sub-periods. We can see that the profit is generally higher before the year of 2010. This is reasonable because the deep learning algorithm and the required computation capacity is only widely available after 2010. With more people employing the deep learning algorithm in trading, the profit will be naturally arbitraged away. To keep a competitive advantage, we need to collect and utilize data that others have not used (for example, data from facebook, twitter and google) and develop more advanced algorithm to find pattern in this data. Panel B of Table 7 reports the Fama French 3-factor loadings of the equal-weighted returns. We can see that the factor loadings are generally very low. The results of value-weighted portfolio are reported in Panel C and D. The only significant factor loading is in the first column of Panel D in Table 7. We see a significant negative factor loading of SMB for the value-weighted returns. This result is natural because big stocks contribute more to the value-weighted portfolio's returns.

[Insert Table 7 Here]

Next, we keep the LSTM-MLP hybrid model unchanged but only feed the month(t-12) to month(t-2)'s returns to the 2-layer LSTM neural network. The 2-layer LSTM neural networks thus have 11 timesteps. All other settings keep the same. The result of this modified model is shown in Table 8. We can see from Table 8 that, when the month(t-1)'s return information is omitted, the equal-weighted return for 1-month holding period is 1.87% with a t-statistic of 9.87, which is lower than the 2.49% equal-weighted return in Table 7. The value-weighted return for 1-month holding period is 1.39% with a t-statistics of 4.51, which is also lower than the 1.56% value-weighted return in Table 7. We will see in Figure 10 that these two models learned different return patterns. However, both of the returns in Table 7 and Table 8 are much better than the pure momentum trading strategy documented in Jeehadesh and Titman (1993) (The monthly equal weighted return of momentum effect is about 0.9% from 1993 through 2017).

[Insert Table 8 and Figure 10 Here]

Figure 10 shows the average z-score of the returns and volumes for the long and short portfolio. From Figure 10 (A), we can see that, when we feed all the past 12 month's returns to the first LSTM layer, the neural network learned a momentum and short-term reversal combined strategy. The long portfolio represented by the solid orange line in Figure 10 (A) has higher returns from month(t-12) to month(t-2), but has a negative return spike in month(t-1). We do not see this pattern in Figure 10 (C), where the month(t-1)'s return is not fed to the first LSTM layer. It is also interesting to see the change of trading volume before and after the portfolio formation. In Figure

10 (B) and (D), we use log of volume's z-score because we want the increase and the decrease of the volume to have same impact on the averages. For example, if the volume increase by 100%, the log of z-score of the volume is log(2). If the volume decreased by 50%, the log of the z-score of the volume is $\log(0.5) = -\log(2)$.The z-score of volume is normalized by (divided by) the volume in month(t-12), where t is the portfolio formation month. As a result the z-score of volume for month(t-12) is always 1 and its log is always 0. We can see that, in Figure 10 (B), when it is a momentum and short-term reversal combined strategy, the volume of both the long and short portfolio increases with a faster speed in month(t-1). However, in Figure 10 (D), the volume is flat or even decreases in month(t-1).

5.3.2 The After-Trading-Cost Returns of The Trading Strategy Learned by the 2-layer LSTM and MLP Hybrid Neural Networks

In this section, we explicitly investigate the after-trading-cost profits of the LSTM-MLP hybrid model. We follow the same procedure as in section 5.1.2 in estimating the trading costs. The average turn over ratio of the LSTM-MLP hybrid model from 1993 to 2017 is 43.7%. We do not report Fame French 3 factor α in this table because Table 7 already shows that the before trading cost returns have low factor loading on the FF-3 factors. FF-3 factors are all before trading cost returns. Regressing an after trading cost return on before trading cost factors is not very meaningful. The after trading cost equal-weighted and value-weighted profits from 1993 to 2017 are reported in Panel (A) and (B) of Table 9, respectively. Again, because the TAQ data is only available from 1993 to 2017, the before trading cost profits are slightly different from what we report in Table 7, in which the sample covers from 1981 through 2017. We can see that the after-trading-cost equal-weighted and value-weighted profits are very significant in all one to three months holding period.

From Panel A of Table 9, we can see that the trading strategy learned by the LSTM-MLP hybrid neural networks achieves a monthly return of 1.57%, which can be translated into 20.6% annual return after the trading cost. This is the main finding of this paper. In next few sections, we will compare this result with the returns of other models and show the advantage of the LSTM-MLP hybrid model.

[Insert Table 9 Here]

5.3.3 Comparing the 2-Layer LSTM and MLP Hybrid Neural Networks With MLP-only Neural Networks

The main advantage of the LSTM-MLP Hybrid Neural Networks is that, when we utilize the cross-sectional and time-series information at the same time, we preserve the sequential information of the time-series data in the LSTM part of the model. To show the value of the sequential information, we will compare it with a MLP-only model in this section. In the MLP-only model, we feed the past 12 months return and the 15 firm characteristics signal (totally 27 features) parallely to a 128-64-32-5 neural network. The MLP will treat the returns and firm characteristics equally. All the other settings are same as the LSTM-MLP Hybrid model in section 5.3.1 and 5.3.2. We will long (short) the 4% of the stocks with the highest probability to outperform (underperform) in the next month. The result is reported in Table 10. The sample covers from 1993 to 2017. First, let us look at the before-trading-cost equal-weighted return in Panel (A) of Table 10. The before-trading-cost equal weighted return of the the MLP-only model is 1.91%, which is not a big difference from the return of 2.37% of the LSTM-MLP Hybrid model. However, the after-trading-cost equal-weighted return of the MLP-only model is only 0.49%, which is much lower than the after trading cost return of 1.57% in the LSTM-MLP Hybrid model. In a closer look, we find that the main reason for this is that MLP-only model has an average monthly turnover ratio of 70.9%, which is much higher than the 43.7% turnover ratio of the LSTM-MLP Hybrid model. Why does the MLP-only model have a higher turnover ratio? Here I give a possbile reason. In month t, for a certain stock, a first sequence of 12 returns for month(t-12) to month(t-1) is fed to the LSTM-MLP Hybrid neural networks. In month t+1, a second sequence of 12 return for month(t-11) to month(t) is fed to the same neural network. Because the LSTM neural network deal with the sequential information and the majority part of the first and the second sequence is same, the LSTM-MLP Hybrid neural networks tend to make similar decision in month(t) and month(t+1). In contrary, the MLP-only model treat the 12 returns as parallel and independent inputs. In month(t+1), the MLP-only neural networks will see a completely different permutation of the returns, although the two return sequence is similar. The MLP-only model thus tend to give very different decision in month(t+1), which result in a higher turnover ratio and higher trading cost.

[Insert Table 10 Here]

5.3.4 Comparing The 2-layer LSTM and MLP Hybrid Neural Network With A Momentum And Short-Term Reversal Double Sort Strategy

In the section 5.3.1, we see that the LSTM-MLP hybrid neural network learned a momentum and short-term reversal combined strategy. It is interesting to compare our result to a hand-engineering momentum and short-term reversal double sort trading strategy. In the double sort strategy, in each month, we first rank stocks into 5 quintiles according to the sum of returns from month(t-12) to month(t-2). We further split the stocks in each quintile into 5 groups according to the return in month(t-1). As a result, we equally split the stocks into 5x5 momentum and short-term reversal double sort portfolios. We long the short-term reversal loser in the momentum winner quintile and short the short-term reversal winner in the momentum loser quintile, which are two corners of the 5x5 double sort portfolios. The sample universe is exactly the same as what we use in section 5.3.2, which covers from 1993 to 2017.

[Insert Table 11 and Figure 11 Here]

The results are reported in Table 11. In Table 9, the before-trading-cost equal-weighted returns for 1, 2 and 3 months holding period of the LSTM-MLP hybrid model are 2.37%, 2.02% and 1.73%, respectively. In Table 11, the same returns for the momentum and short-term reversal double sort strategy are 2.08%, 1.50% and 1.03%, respectively. We can see that the return difference become larger when the holding period becomes longer. This indicates that the profits shrink faster after the portfolio formation month in the momentum and short-term reversal double sort model than in the LSTM-MLP hybrid model. In Panel (A) of Table 11, we can also see that the after-trading-cost return of 0.58% is much lower than their corresponding value 1.57% in the LSTM-MLP hybrid model in Table 9. To find out the reason for this, we show the average z-score of returns and volumes before and after the portfolio formation for double sort model in Figure 11. We have three major findings in Figure 11: (1) The profit shrink much faster in Figure 11(A) after month(t) than what we see in Figure 10 (A), which confirms what we have found in Panel (A) of

Table 11. (2) In Figure 11 (B), we can see that the volume of the short portfolio decreases in month(t-1), while in Figure 10 (B) the volume of the short portfolio increase all the way up to month(t). (3) The absolute value of the z-score of the returns in Figure 11 (A) before the portfolio formation month(t) is much higher than that of Figure 10 (A), especially in month(t-1). In Figure 11(A), The peak value of z-score for the short portfolio at month(t-1) is almost 1.8, while in Figure 10 (A), the peak value is only 0.5. The z-scores for month(t-12) to month(t-2) for the double sort model are also higher than those for the LSTM-MLP hybrid model. This indicates that the double sort strategy is trading stocks with extreme previous returns, which results in a 89.9% turnover ratio. (compared to 43.7% turnover ratio for the hybrid model.) This is the reason why the after-trading-cost return for the double sort strategy is much lower than the LSTM-MLP hybrid model. We can conclude that, although the LSTM-MLP hybrid model learned a momentum and short-term reversal combined strategy which is similar to the 5x5 double sort strategy, the hybird model trades very different stocks from the hand-engineering double sort model. The trading strategy learned by the hybrid model trades stocks with less extreme returns but achieves a higher before and after-trading-cost profit.

5.3.5 Ensembe Learning: Combining the 2-Layer LSTM And MLP Hybrid Model With Random Forest And XGBoost Model

Ensemble learning is a very simple idea: we can combine several weak learners and turn them into a strong learner. We can show the idea of ensemble learning in the following concrete example. Suppose we have three predictors of the stock market and they only predict whether the market will go up or down. They all have 60% of chance to be correct. If we let them vote and follow the opinion that get most votes, our chance to be correct improves to $C_3^3\ 0.6^3 + C_3^2\ 0.6^2 * 0.4$=64.8%, where C_n^r is possible combinations of taking r objects from a set of n objects. If we have 101 such predictors, our winning chance improves to 97.9%. However, this requires all the prediction be independent.

In this section, we will employ three predictors: the LSTM-MLP hybrid model, the random forest model and the XGBoost model. The random forest classifier itself is a ensemble learning model, which is an ensemble of decision trees. Each decision tree is trained on a subset of sample and/or features. Although each decision tree may not be a good predictor because of the limited samples and/or features and overfitting, the combination of them will give fairly accurate prediction. XGBoost stands for “Extreme Gradient Boosting”. The idea of Gradient Boosting is as follow. Suppose we need to do a regression and find the function $\mathrm{y} = \mathrm{f(x)}$. We first regress y on x and find a $\hat{y} = f_1(x)$. We then regress the error terms on x and find a $f_2(x)$. We continue to regress the error of errors on x and find a $f_3(x)$ and so on. Each $f_i(x)$ represents a decision tree. The final $f(x)$ will be $\sum_{i=1}^{n} f_i(x)$. The idea of Gradient Boosting is similar to a Taylor expansion.

In the ensemble learning model in this section, we employ a soft voting method. In each month, the three model will give each stock a 5-dimensional probability vector, such as [0.3, 0.2, 0.1, 0.21, 0.19], which indicate the stock has a 30% probability to fall into the lowest return quintile and a 19% probability to fall into the highest return quintile. The summation of the five probabilities is always 1. For each stock in a month, we sum up the three vectors given by the three models. The summation of the five element of the new vector is 3. We divide each element of the vector by 3. With the ensemble learning prediction vector for each stock, we then follow the same procedure as in the LSTM-MLP hybrid. We will long (short) the 4% of the stocks with the highest probability to outperform (underperform) in the next month.

[Insert Table 12 and Figure 12 Here]

The results of the Random Forest, XGBoost and the ensemble learning model are reported in Table 12. In the first column of Panel A and B, we can see that the before-trading-cost returns of random forest and XGBoost model are 1.59% and 2.36%, respectively. The return of XGBoost is almost same as the LSTM-MLP hybrid model. However, the after-trading-cost returns of these two models, which are 0.15% and 1.03% respectively, are much lower than that of the LSTM-MLP hybrid model. The reason of this also lies in the turnover ratio. The average monthly turnover ratio of random forest and XGBoost model are 77.5% and 72.2%, respectively.

Although the before-trading-cost return of the random forest and XGBoost model is lower than the return of LSTM-MLP hybrid model, if we combine these three weak learner, we achieve a before-trading-cost return of 2.53% with a t-statistic of 9.74, which is higher than all three models. The after-trading-cost return is lower than that of the LSTM-MLP hybrid model, which is due to the high turnover ratio in the random forest and XGBoost model.

In Figure 12, we depict the feature importance of the random forest and XGBoost model. Generally, the feature that make the most decrease in the impurity in the tree node splits will have higher feature importance. In each month, the model give each feature a feature importance score. The results in Figure 12 are the average feature importance score across the 300 months from 2013 through 2017. The features are all z-scores. The meaning of the feature names can be found in Section 4.2. We can see that, for XGBoost model, the Cashflow to price ratio (z_cp), the log of firm size (z_logme) and investment-to-capital ratio (z_ik) are the most important features. For random forest model, the log of firm size, the return of month(t-1) and month(t-3) have higher feature importance scores.

5.3.6 Did All The Models Learn Similar Trading Strategies?

In this section, we will explore whether all the models we has explored are trading similar stocks. Up to this section, we have 5 monthly trading models: the LSTM-MLP hybrid model, the MLP-only model, the momentum and short-term reversal 5x5 double sort model, the random forest model and the XGBoost model. We add two models to the regression: the pure momentum strategy and pure short-term reversal strategy. The returns of the pure momentum strategy is the profits when we long the top 20% momentum winner and short the bottom 20% momentum loser. The returns of the pure short-term reversal strategy is the profits when we long the 20% of stocks that have lowest returns in month(t-1) and long the 20% of stocks that have the highest returns in month(t-1). So we totally have seven models.

The regression results are reported in Table 13. First, in column (1), we perform a regression of the double sort model on the pure momentum and the pure short-term reversal model. We can see that the R^2 of the regression is 87.76% and the two slopes have very high t-statistics. This indicates that there is multi-colinerity between the returns of these three models. As a result, in the following regression, we do not include all three of the double sort, momentum and short-reversal model, otherwise the t-statistics of these three model will become very small.

Figure 10 (A) shows that the LSTM-MLP hybrid model learns a momentum and short-term reversal combined strategy. However, when we regress the return of the hybrid model on the returns of 5x5 double sort model in column (2), we find the R^2 is only 6.91%. This indicates that the LSTM-MLP hybrid model learns a trading strategy that is very different from the hand-engineering double sort strategy. When the return of momentum model is added to the independent variable, in column (5), we can see that the R^2 improves to 20.27%. From column (2) to (4), we can conclude that the LSTM-MLP hybrid model shows a stronger momentum pattern than the short-term reversal pattern. Column (6) to (12) show us that the returns of the LSTM-MLP hybrid model have more similarity with the MLP-only, random forest and XGBoost model. When we add all the models except the short-term reversal to the independent variables, the five models together explains 47.76% of the variation of the returns of the LSTM-MLP hybrid model. From the analysis in this section, we can conclude that the LSTM-MLP hybrid model is trading very different stocks from other models.

[Insert Table 13 Here]

6 Conclusion

This paper is a simple practice in applying the deep learning models to the financial datasets. We performs three tests. In the first test, we apply the 2-layer LSTM neural network to the CRSP daily return datasets. We use a stock's past 80 days' returns as inputs and feed them sequentially to the LSTM neural network. The neural network output a 5-dimensional vector which tells us the stock's probability to outperform or underperform. We long (short) the 10% of the stocks with the highest probability to outperform (underperform) and achieve a *monthly* equal-weighted return of 29.58% with a t-statistic of 26.81. The neural network learns a strong short-term reversal pattern. The long (short) portfolio has significant negative (positive) return spikes in the period from day(t-14) through day(t-1) as shown in Figure 7. We download 318GB of data from the TAQ trade and quote database to explicitly estimate the trading cost. After consider the trading cost, the 29.58% monthly return turn negative. The first test teachs us a lesson. There are huge rooms of profit that we can "data-mining" from the financial dataset. The 29.58% monthly profit can be translated into 2141% annual profits. However, without accurately considering the trading cost, the big profits are usually unconvincing.

In the second test, we extend the 2-layer LSTM neural network model to 17 industry portfolios. The neural network learned a momentum return pattern without short-term reversal effect. The return pattern of the long-short portfolio is shown in Figure 9. This result confirms the findings in Moskowitz and Grinblatt (1999).

The most important contribution of this papar comes from the third test. We design a 2-layer LSTM and MLP hybrid neural network. It utilized both the time-series price information and the cross-sectional firm characteristics information. It also preserves the "sequential" information in the time-series. We achieve an equal-weighted return of 2.49 with a t-statistic of 12.63 before the trading costs from 1981 through 2017.

To show the advantage of the LSTM-MLP hybrid model, we compare it with a MLP-only and a hand-engineering momentum and short-term reversal double sort strategy. In the MLP-only model, we feed the return and firm characteristics information parallelly to the neural network. So the sequential information of the returns is omitted. It turns out the MLP-only model has a much higher turnover ratio than the hybrid model, which result in a low after-trading-cost return. In the hand-engineering momentum and short-term reversal double sort strategy, we find that the strategy tend to trade stocks with extreme previous returns. As a result, the turnver ratio is even higher than the MLP-only model. The after-trading-cost return turn insignificant for the double sort model.

We then demonstrate a practice of ensemble learning. We show that, from the three relatively weak learner: the LSTM-MLP hybrid model, the random forest model and XGBoost model, we can combine them and make a strong learner. Lastly, we perform a regression of the returns of the LSTM-MLP hybrid model on other models. We find the other five models altogether can only explain 47.76% of the variation of the returns of the hybrid model. The machine learning models learn very different trading strategies.

In the beginning of this paper, we mentioned the deep learning algorithms can learn features by themselves. A good example of this is the application of convolutional neural network (CNN) to pattern recognition. In the first few layers of the CNN neural network, it first recognizes some features like edges and contours. In the deeper layers, the CNN will combine these edges and contours into object features such as eyes and ears. In the last layer the CNN neural network can detest a person or a cat in the image. In this paper, we are still feeding human engineered features to the neural network. The firm characteristics are combinations of the raw accounting data. In next step, we will try to feed raw account data to the neural network and study what it learns. Another extension of this paper is from the output or target y. In this paper, we train use the quintile ranks of the the raw return as the target y. We find that the trading strategy trade small stocks and incur a higher trading cost. We might misguide the deep learning algorithm because we do not feed the trading cost information to the neural network. In next step, we can incorporate the trading cost into the target y and see the difference in the learning outcomes. Lastly, our results in this paper probably have not reached the optimal level because of limited computation resource in tuning the hyperparameters. With more computation capacity, we are confident that there are still room to improve the profits.

Appendix A:

In this appendix, we will cover three topics:

- **An introduction to the basic neural networks: mlp (multi-layer perceptron), rnn (recurrent neural network)**
- **The vanishing or exploding gradient problem of RNN**
- **How Long Short-Term Memory (LSTM) solved the vanishing or exploding gradient problem**

In this section, we will give a brief overview of MLP (Multi-Layer Perceptron), RNN (Recurrent Neural Network) and LSTM (Long Short-Term Memory). In short, MLP neural network is usually used for parallel inputs. RNN is usually used for time-series inputs. RNN has a serious problem called Vanishing or Exploding Gradient, which make it hard to be used in practice. LSTM is a modified version of RNN and solved the Vanishing or Exploding Gradient problem. Today, LSTM and one of its variant called GRU (Gated Recurrent Unit) is the *de facto* standard RNN model.

A.1 The Forward And Backward-Propagation Of The MLP Neural Network

A MLP (Multi-Layer Perceptron) model is a multi-layer neural network. A example of a classic MLP is illustrated in Figure A.1.

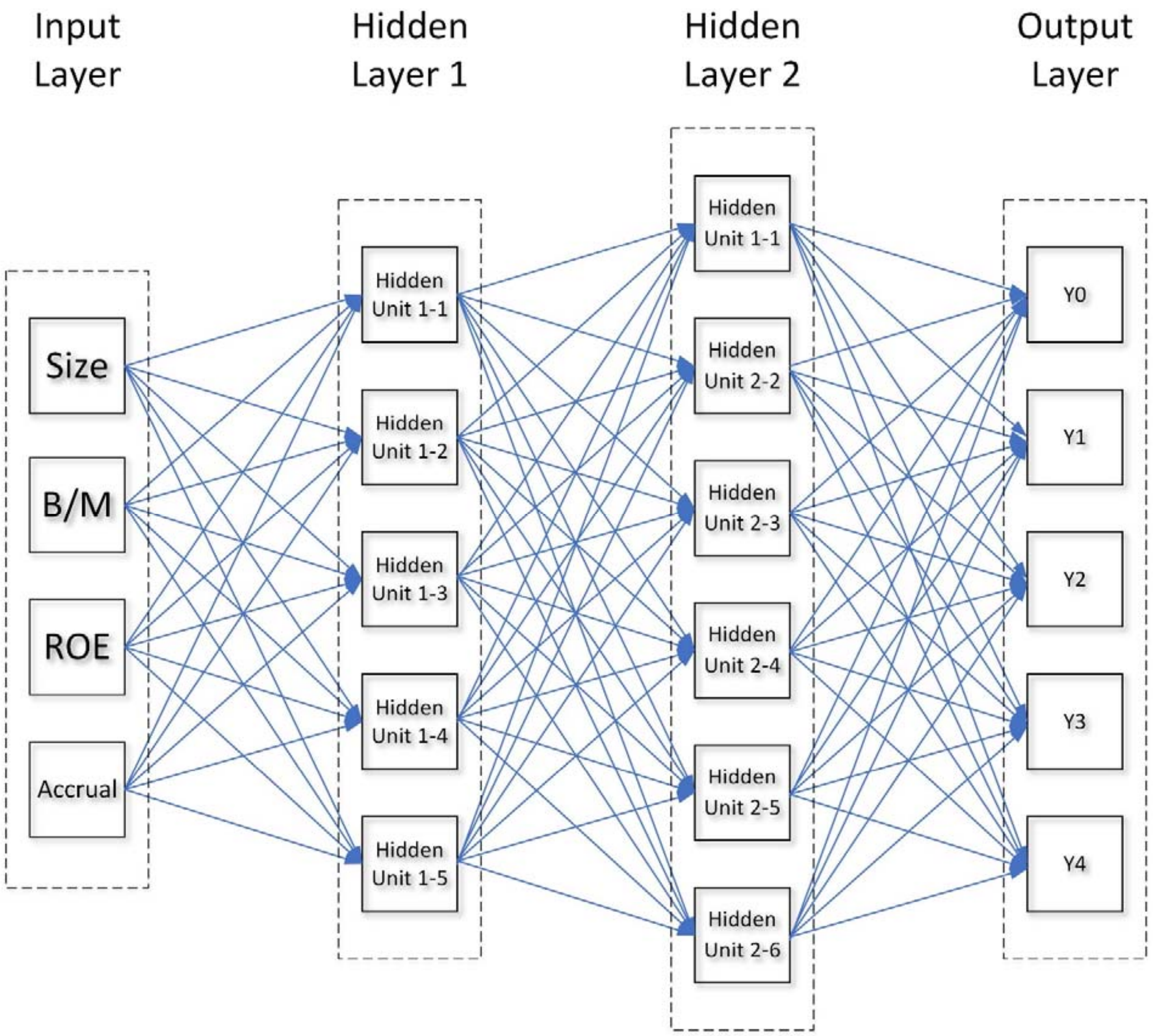


Figure A.1, an example of MLP neural network

In Figure A.1, the input “x” is a 4-dimensional vector that consists of the information of firm size, book-to-market ratio, ROE and accrual at time t-1. We will use these input variables to predict whether the stock will outperform in the next period. The neural network generates a function $\hat{y} = \mathrm{f(x)}$ and predict a fitted value $\hat{y}$. The function $\hat{y} = \mathrm{f(x)}$ relies on many weights, which can be seen as a set of “knobs”. We adjust the knobs in the training process to minimize the objective function. The objective function measures the average distance between the real y and predicted $\hat{y}$ of all training samples. The training process use procedures called forward-propagation and backward-propagation to adjust the weights. We will start with the forward-propagation process from a single neuron.

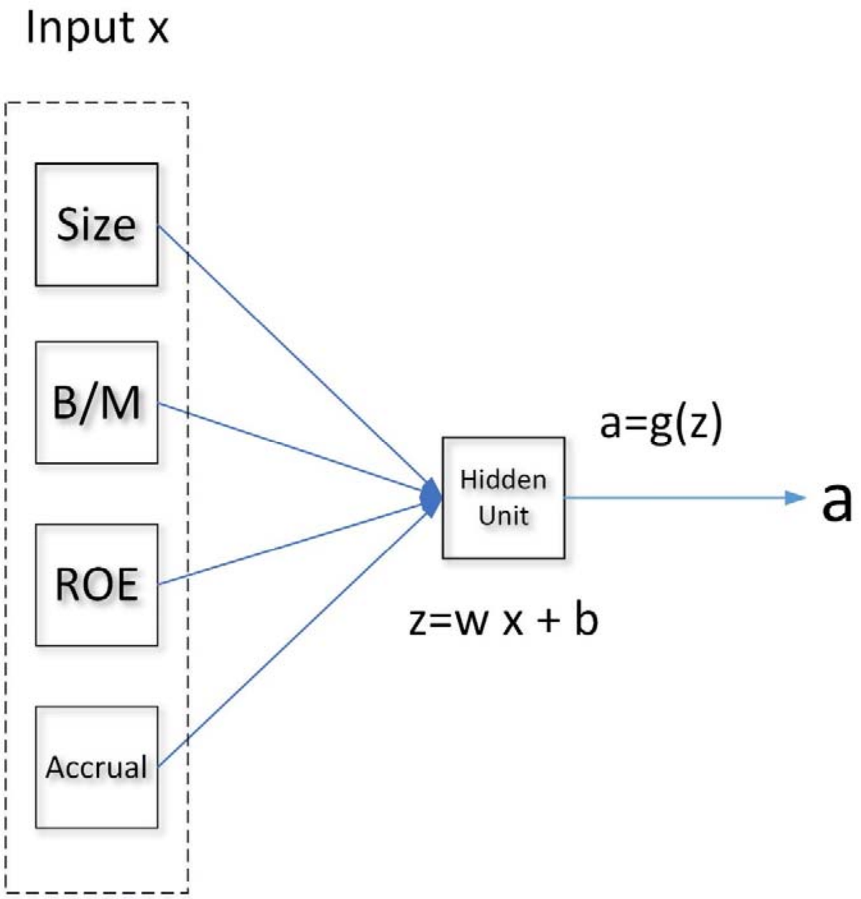


Figure A.2, a basic neuron

The forward-propagation process generates the function $\hat{y}$ = f(x) through the hidden units or neurons in each hidden layer. For example the neuron in Figure A.2 has 4 inputs. The input vector is $x = [x_1, x_2, x_3, x_4]^T$. We will use 4 weights $w = [w_1, w_2, w_3, w_4]$ and a bias term b to calcualte a linear combination “z” of the 4-dimensional input x.

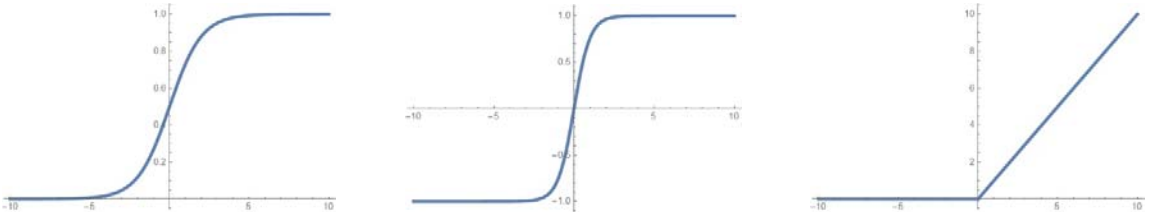
$$z = wx + b \tag{A.1}$$

Then we apply an activation function on z.

$$a = g(z) \tag{A.2}$$

The activation “a” will serve as the input to the next layer. The activation function must be non-linear because multiple layers of the linear combination is still a linear combination of input x. With non-linear activation function, we can simulate any function f(x) through multiple hidden layers of neurons. The often-used activation function has three forms: Sigmoid, Tanh and Relu. The formula and the shape of the functions is shown in equation (A.3) and Figure A.3.

$$\begin{aligned} &\text{Sigmoid:} \quad a = \frac{1}{1+e^{-z}} \\ &\text{Tanh:} \quad a = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \\ &\text{Relu:} \quad a = \begin{cases} 0, & z<0 \\ z, & z \geq 0 \end{cases} \end{aligned} \tag{A.3}$$



Sigmoid Tanh Relu

Figure A.3, the shape of the Sigmoid, Tanh and Relu activation function.

The Relu function is more commonly used today. The activation function is also called “squash” function. You can see from the shape of the sigmoid function that when z is either very positive or very negative, it will be squashed to 0 or 1. We will use a number in square bracket to represent the hidden layer number in the following equations (e.g. $\mathrm{z}^{[1]}$ is z for the first layer.). Please note that z and a are all vectors because we have more than one hidden unit in a hidden layer. From now on, we strictly follow the structure of the neural network in Figure A.1. The input is 4-dimensional. The first and second layer is 5 and 6 dimensional, respectively. The output layer is 5 dimensional. So $\mathrm{z}^{[1]}$ is 5-dimensional and $\mathrm{z}^{[2]}$ is 6-dimensional. So the weight $\mathrm{w}^{[1]}$ for the first hidden layer is no longer 4-dimensional vector, but a [5,4] matrix. Please note that when a [n,p] matrix is multiplied by a [p,m] matrix, we obtain a [n,m] matrix. For the first layer, we have,

$$\mathrm{z}^{[1]} = \mathrm{w}^{[1]}x + b^{[1]} \tag{A.4}$$

From now on, we refer a n-dimensinal vector as a [n,1] dimensional matrix. x is a [4,1] dimensional matrix and $b^{[1]}$ is a [5,1] dimensional matrix. $\mathrm{w}^{[1]}$ is [5,4] dimensional matrix. [5,4] multiplied by [4,1] is [5,1], which is consistent with the dimension of $\mathrm{z}^{[1]}$. We then need an activation of $\mathrm{z}^{[1]}$

$$\mathrm{a}^{[1]} = \mathrm{g}^{[1]}\left(\mathrm{z}^{[1]}\right) \tag{A.5}$$

$a^{[1]}$ is [5,1] dimensional matrix. $\mathrm{z}^{[1]}$ is [5,1] dimensional and $\mathrm{g}^{[1]}$ is the activation function of scaler that applys to each element of $\mathrm{z}^{[1]}$. Then $a^{[1]}$ is fed to the hidden layer 2, we have,

$$\mathrm{z}^{[2]} = \mathrm{w}^{[2]}a^{[1]} + b^{[2]} \tag{A.6}$$

$\mathrm{z}^{[2]}$ is [6,1] dimensional matrix, $\mathrm{w}^{[2]}$ is [6,5], $a^{[1]}$ is [5,1], $b^{[1]}$ is [6,1] . We then need the activation of hidden layer 2.

$$\mathrm{a}^{[2]} = \mathrm{g}^{[2]}\left(\mathrm{z}^{[2]}\right) \tag{A.7}$$

$a^{[2]}$ is [6,1] dimensional matrix, $\mathrm{z}^{[2]}$ is [6,1] and $g^{[2]}$ is the activation that applys to each dimension of $\mathrm{z}^{[2]}$. The output of the second hidden layer will be fed to the output layer. The output layer has 5 units.

$$\mathrm{z}^{[3]} = \mathrm{w}^{[3]} a^{[2]} + b^{[3]} \tag{A.8}$$

$z^{[3]}$ is [5,1] dimensional matrix, $a^{[2]}$ is [6,1], $\mathrm{w}^{[3]}$ is [5,6] , $b^{[3]}$ is [5,1] . The predicted $\hat{y}$ is

$$\hat{y} = \mathrm{softmax}\left(\mathrm{z}^{[3]}\right) \tag{A.9}$$

$\mathrm{z}^{[3]}$ and $\hat{y}$ are both [5,1] matrix and the softmax function is defined as

$$\hat{y}_{\mathrm{i}} = \frac{e^{z_i}}{\sum_{i=1}^{5} e^{z_i}} \tag{A.10}$$

Please note that, the 5 components of $\hat{y}$ sum to 1. The softmax function is essentially a exp() function applied on $\mathrm{z}^{[3]}$ and then performs a normalization across the 5 dimensions. $\hat{y}$ can then be interpretted as the probabilities that a stock falls into the 5 return quintiles. For this output layer, we temporarily write $\mathrm{z}^{[3]}$ as z for simplicity.

The above is the full procedure of the forward propagation, which propagates from the input x to predicted vector $\hat{y}$. This forward propagation procedure serves as the funtion $\hat{y} = f(x)$ we mentioned before. The parameters in this function is the weights w and the biases b in each hidden layer. We then compare the real y and the predicted $\hat{y}$ and use the errors to adjust the weights and biases (i.e., adjust f(x)). This adjustment process is called back-propagation.

Firstly, we need a cost function to measure the distance between real y and the predicted $\hat{y}$. In our case, the output layer is a softmax function. The cost function for one data point of y and $\hat{y}$ in 5-way classification is

$$\mathcal{L}(\mathrm{y}, \hat{y}) = -\sum_{i=1}^{5} y_i \log(\hat{y}_i) \tag{A.11}$$

We need to minimize this cost function. Equivalently, it is also to maximize $\sum_{i=1}^{5} \log(\hat{y}_i^{y_i})$ or $\prod_{i=1}^{5} \hat{y}_i^{y_i}$. Recall that the probability mass function of bernuolli distribution is $P(y|\hat{y}) =$

$\hat{y}^y\ (1-\hat{y})^y$. Since $\sum_{i=1}^{5}\hat{y}_i = 1$, $P(y|\hat{y}) = \prod_{i=1}^{5}\hat{y}_i^{y_i}$ can serve as the probability mass function of the "multinuolli" distribution. (Please note that in the bernuolli probability mass function, y and $\hat{y}$ are scaler. In the $P(y|\hat{y}) = \prod_{i=1}^{5}\hat{y}_i^{y_i}$, y and $\hat{y}$ are vectors.) As a result, minimizing the cost function $\mathcal{L}(\mathrm{y},\hat{y})$ is same as maximizing the likelihood of $P(y|\hat{y})$. Here is a concrete example, if,

$$\hat{y}_1 = [0.1, 0.1, 0.14, 0.16, 0.5]$$

$$\hat{y}_2 = [0.5, 0.1, 0.14, 0.16, 0.1]$$

$$\mathrm{y} = [0, 0, 0, 0, 1]$$

From intuition, we can directly see that $\hat{y}_1$is closer to y than $\hat{y}_2$. From the equation $P(y|\hat{y}) = \prod_{i=1}^{5}\hat{y}_i^{y_i}$, one can calculate that $P(y|\hat{y}_1) = 0.5^1 = 0.5$ and $P(y|\hat{y}_2) = 0.1^1 = 0.1$, then, $\mathcal{L}(\mathrm{y}_1,\hat{y}) = -\log(0.5) = \log(2)$ and $\mathcal{L}(\mathrm{y}_2,\hat{y}) = -\log(0.1) = \log(10)$. So $\mathcal{L}(\mathrm{y},\hat{y}_1) < \mathcal{L}(y,\hat{y}_2)$, which means $\hat{y}_1$ is a better predictor of y than $\hat{y}_2$. Despite of this maximum likelihood explanation, for simplicity, you can just think of the cost function as the distance between the real y and the predicted $\hat{y}$. We then need to minimize this distance or find out the lowest point of the cost function. In each step, we adjust the weights along the fastest direction down the hill of the cost function $\mathcal{L}$. The direction is given by $\frac{\partial\mathcal{L}}{\partial w}$. Given the learning rate α as a step size, we have the numerical updating rule of the weight as

$$\mathrm{w} \coloneqq \mathrm{w} - \alpha\ \frac{\partial\mathcal{L}}{\partial w} \qquad \text{(A.12)}$$

The equation means that we decrease the weights by $\alpha\ \frac{\partial\mathcal{L}}{\partial w}$ in each step. Because w is a matrix, $\partial\mathcal{L}/\partial w$ is also a matrix. $\mathcal{L}$ is simply a scaler function. $\partial\mathcal{L}/\partial w$ and w have the same dimension. In practice, we will start with a random weight w and use the above updating rule to go down the hill of the cost function step by step. Of course, in the w space that is more than 3-dimensional, the shape of the cost function can not be easily imagined. The optimization process is not simply a going-down-the-hill process.

How do we calculate $\partial \mathcal{L}/\partial w$? This is a process called back-propagation. The back-propagation is basically an application of the chain rule. The cost function for one data point is $\mathcal{L}(\mathrm{y}, \hat{y}) = -\sum_{i=1}^{5} y_i \log(\hat{y}_i)$, in which, y_i can be treated as constant because they are directly from the training data. The fitted $\hat{y}_\mathrm{i}$ are variables and are connected to $\mathrm{w}^{[3]}$ through $\hat{y}_\mathrm{i} = \frac{e^{z_i}}{\sum_{i=1}^{5} e^{z_i}}$ and $\mathrm{z}^{[3]} = \mathrm{w}^{[3]} a^{[2]} + b^{[3]}$. From the chain rule, in order to calculate ${}^{\partial \mathcal{L}}/_{\partial w^{[3]}}$, we need to first calculate ${}^{\partial \mathcal{L}}/_{\partial \hat{y}_i}$. From the equation (A.11), calculating the patial derivative against each element of $\hat{y}_i$,

$$\frac{\partial \mathcal{L}}{\partial \hat{y}_i} = -\frac{y_i}{\hat{y}_i} \tag{A.13}$$

Each $\hat{y}_i$ is a function of $z_1^{[3]}$to $z_5^{[3]}$, so we need to consider ${}^{\partial \hat{y}_i}/_{\partial z_j}$ respectively. In the following equations, z_j refer to $z_j^{[3]}$

(1) If i=j,

$$\begin{aligned} \frac{\partial \hat{y}_i}{\partial z_i} &= \frac{e^{z_i}\left(\sum_{i=1}^{5} e^{z_i}\right) - e^{z_i} e^{z_i}}{\left(\sum_{i=1}^{5} e^{z_i}\right)^2} \\ &= \frac{e^{z_i}}{\sum_{i=1}^{5} e^{z_i}} (1 - \frac{e^{z_i}}{\sum_{i=1}^{5} e^{z_i}}) \\ &= \hat{y}_i (1 - \hat{y}_i) \end{aligned} \tag{A.14}$$

(2) If i≠j,

$$\frac{\partial \hat{y}_i}{\partial z_j} = \frac{-e^{z_i} e^{z_j}}{\left(\sum_{i=1}^{5} e^{z_i}\right)^2} \tag{A.15}$$

$$= -\frac{e^{z_i}}{\sum_{i=1}^{5} e^{z_i}} (\frac{e^{z_j}}{\sum_{i=1}^{5} e^{z_i}})$$

$$= -\hat{y}_i \, \hat{y}_j$$

Let us take z_1 as an example. z_1 links to $\mathcal{L}$ through 5 paths, i.e. $\hat{y}_1$ to $\hat{y}_5$, So to apply the chain rule and calculate ${\partial \mathcal{L}}/{\partial z_1}$, we need to sum over the 5 terms of $\frac{\partial \mathcal{L}}{\partial \hat{y}_i} \times \frac{\partial \hat{y}_i}{\partial z_1}$, i=1,2,…,5

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z_1} &= \sum_{i=1}^{5} \frac{\partial \mathcal{L}}{\partial \hat{y}_i} \times \frac{\partial \hat{y}_i}{\partial z_1} \\ &= \frac{\partial \mathcal{L}}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1} + \frac{\partial \mathcal{L}}{\partial \hat{y}_1}\frac{\partial \hat{y}_1}{\partial z_1} \end{aligned} \tag{A.16}$$

Substituting equation (A.14) and (A.15) into equation (A.16), we obtain

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial z_1} &= -\frac{y_1}{\hat{y}_1}\hat{y}_1(1-\hat{y}_1) + \frac{y_2}{\hat{y}_2}\hat{y}_2\hat{y}_1 + \frac{y_3}{\hat{y}_3}\hat{y}_3\hat{y}_1 + \frac{y_4}{\hat{y}_4}\hat{y}_4\hat{y}_1 + \frac{y_5}{\hat{y}_5}\hat{y}_5\hat{y}_1 \\ &= -y_1(1-\hat{y}_1) + y_2\hat{y}_1 + y_3\hat{y}_1 + y_4\hat{y}_1 + y_5\hat{y}_1 \\ &= -y_1 + y_1\hat{y}_1 + y_2\hat{y}_1 + y_3\hat{y}_1 + y_4\hat{y}_1 + y_5\hat{y}_1 \end{aligned} \tag{A.17}$$

Note that $\sum_{i=1}^{5} y_i = 1$, so,

$$\frac{\partial \mathcal{L}}{\partial z_1} = \hat{y}_1 - y_1 \tag{A.18}$$

We can use a similar procedure to generalize this result and obtain,

$$\frac{\partial \mathcal{L}}{\partial z_j} = \hat{y}_j - y_j \tag{A.19}$$

Or in the vector form,

$$\frac{\partial \mathcal{L}}{\partial z^{[3]}} = \hat{y} - y \tag{A.20}$$

Now we restore z to $z^{[3]}$. Because $z^{[3]} = \mathrm{w}^{[3]}a^{[2]} + b^{[3]}$, we can apply the chain rule,

$$\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \frac{\partial \mathcal{L}}{\partial z^{[3]}} a^{[2]T}$$
$$\frac{\partial \mathcal{L}}{\partial b^{[3]}} = \frac{\partial \mathcal{L}}{\partial z^{[3]}} \tag{A.21}$$

In the above equation, $\frac{\partial \mathcal{L}}{\partial w^{[3]}}$ is [5,6] dimensional. $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$ is [5,1] dimensional , $a^{[2]T}$ is [1,6] dimensional. $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$ can be calculated from $\hat{y}$ and y in equation (A.20). $a^{[2]T}$ of the current step is known in the forward propagation.

In the above process, we first calculate $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$. Using $\mathrm{z}^{[3]} = \mathrm{w}^{[3]} a^{[2]} + b^{[3]}$ and applying the chain rule, we obtain $\frac{\partial \mathcal{L}}{\partial w^{[3]}}$ and $\frac{\partial \mathcal{L}}{\partial b^{[3]}}$, which give us a direction to adjust the weights of the third layer. We need to go further backward to calculate $\frac{\partial \mathcal{L}}{\partial w^{[2]}}$ and $\frac{\partial \mathcal{L}}{\partial b^{[2]}}$. $w^{[2]}$ and $b^{[2]}$ are connect to the $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$ through a path of $\mathrm{z}^{[2]} = \mathrm{w}^{[2]} a^{[1]} + b^{[2]}$ and then $\mathrm{a}^{[2]} = \mathrm{g}^{[2]}(\mathrm{z}^{[2]})$ and then $\mathrm{z}^{[3]} = \mathrm{w}^{[3]} a^{[2]} + b^{[3]}$. In order to know $\frac{\partial \mathcal{L}}{\partial w^{[2]}}$ and $\frac{\partial \mathcal{L}}{\partial b^{[2]}}$, we need to know $\frac{\partial \mathcal{L}}{\partial z^{[2]}}$. In order to know $\frac{\partial \mathcal{L}}{\partial z^{[2]}}$, we need to know $\frac{\partial \mathcal{L}}{\partial a^{[2]}}$ and $\frac{\partial a^{[2]}}{\partial z^{[2]}}$. In order to know $\frac{\partial \mathcal{L}}{\partial a^{[2]}}$, we need to know $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$, which we already know. Applying chain rule on $\mathrm{z}^{[3]} = \mathrm{w}^{[3]} a^{[2]} + b^{[3]}$, we get,

$$\frac{\partial \mathcal{L}}{\partial a^{[2]}} = \mathrm{w}^{[3]T} \frac{\partial \mathcal{L}}{\partial z^{[3]}} \tag{A.22}$$

From $\mathrm{a}^{[2]} = \mathrm{g}^{[2]}(\mathrm{z}^{[2]})$, we directly obtain

$$\frac{\partial a^{[2]}}{\partial z^{[2]}} = \mathrm{g}^{[2]\prime}(\mathrm{z}^{[2]}) \tag{A.23}$$

Applying the chain rule for $\frac{\partial \mathcal{L}}{\partial z^{[2]}}$, we have,

$$\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} = (\mathrm{w}^{[3]T} \frac{\partial \mathcal{L}}{\partial z^{[3]}}) \circ \mathrm{g}^{[2]\prime}(\mathrm{z}^{[2]}) \tag{A.24}$$

Please note, in the above equation, “∘” is element-wise multiplication. The transpose of $\mathrm{w}^{[3]}$ or $\mathrm{w}^{[3]T}$ is [6,5] dimensional. $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$ is [5,1] dimensional , so $(\mathrm{w}^{[3]T}\frac{\partial \mathcal{L}}{\partial z^{[3]}})$ is [6,1] dimensional. $\mathrm{g}^{[2]\prime}(\mathrm{z}^{[2]})$ and $\frac{\partial \mathcal{L}}{\partial z^{[2]}}$ is also[6,1] dimensional. So $\frac{\partial \mathcal{L}}{\partial z^{[2]}}$ is also [6,1] dimensional.

Following these steps and back propagate all the way to the input layer, we will have,

Forward Propagation	Backward Propagation
$\mathrm{z}^{[1]} = \mathrm{w}^{[1]}x + b^{[1]}$ $\mathrm{a}^{[1]} = \mathrm{g}^{[1]}(\mathrm{z}^{[1]})$ $\mathrm{z}^{[2]} = \mathrm{w}^{[2]}a^{[1]} + b^{[2]}$ $\mathrm{a}^{[2]} = \mathrm{g}^{[2]}(\mathrm{z}^{[2]})$ $\mathrm{z}^{[3]} = \mathrm{w}^{[3]}a^{[2]} + b^{[3]}$ $\hat{y} = \frac{e^{z_i^{[3]}}}{\sum_{i=1}^{5} e^{z_i^{[3]}}}$ Cost function: $\mathcal{L}(\mathrm{y}, \hat{y}) = -\sum_{i=1}^{5} y_i \log(\hat{y}_i)$	$\frac{\partial \mathcal{L}}{\partial z^{[3]}} = \hat{y} - y$ $\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \frac{\partial \mathcal{L}}{\partial z^{[3]}} a^{[2]T}$ $\frac{\partial \mathcal{L}}{\partial b^{[3]}} = \frac{\partial \mathcal{L}}{\partial z^{[3]}}$ $\frac{\partial \mathcal{L}}{\partial z^{[2]}} = \left(\mathrm{w}^{[3]T}\frac{\partial \mathcal{L}}{\partial z^{[3]}}\right) \circ\ \mathrm{g}^{[2]\prime}(\mathrm{z}^{[2]})$ $\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}} a^{[1]T}$ $\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial z^{[2]}}$ $\frac{\partial \mathcal{L}}{\partial z^{[1]}} = \left(\mathrm{w}^{[2]T}\frac{\partial \mathcal{L}}{\partial z^{[2]}}\right) \circ\ \mathrm{g}^{[1]\prime}(\mathrm{z}^{[1]})$ $\frac{\partial \mathcal{L}}{\partial w^{[1]}} = \frac{\partial \mathcal{L}}{\partial z^{[1]}} x^{T}$ $\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial z^{[1]}}$
“∘” is element-wise multiplication.	

Table A.1, the forward and backward propagation for a single data point.

The above equations are all for a single data point. $\mathrm{z}^{[1]}$ and $\mathrm{a}^{[1]}$ are [5,1] dimensional. The $\hat{y}$ and y in the cost function are also [5,1] dimensional. What if we have millions of data points? How do we use these equations? The answer is that we need to vectorize the above equations. Suppose we have “m” data points. Two principles is needed in the vectorization:

(1) Currently all the variables that relate to the input data, the hidden units and predicted values are [n,1] dimensional. We need to change them to [n,m] dimensional and use an uppercase letter to represent them. For example, X will be [4,m] dimensional and each column of X is one input data point. $\mathrm{Z}^{[1]}$ and $\mathrm{A}^{[1]}$ will be [5,m] dimensional. $\mathrm{Z}^{[2]}$ and $\mathrm{A}^{[2]}$ will be [6,m] dimensional. $\widehat{\mathrm{Y}}$ and Y will be [5,m] dimensional.

(2) All the weights remain the same shape as before. The bias column vector b will copy m times and become [n,m] dimensional. So we express this [n,m] dimensional matrix as $\boldsymbol{bI}$, where $\boldsymbol{I}$ is a [1,m] dimensional row vector with value 1 in each element.

We also need to change the cost function to

$$\mathcal{L}(Y, \hat{Y}) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{5} y_i^{(j)} \log(\hat{y}_i^{(j)}) \qquad \text{(A.25)}$$

Where $y_i^{(j)}$ and $\hat{y}_i^{(j)}$ refer to the ith element of the jth output data y and predicted $\hat{y}$. The cost function is an average distance between the real y and predicted $\hat{y}$ of m data points. The equations in Table A.1 hence becomes the equations in Table A.2.

The formula for the forward and backward propagation for MLP neural network in Figure A.1 in vectorized form is illustrated in Table A.2. We list the dimensions of each formula in Table A.2 to facilitate reader. The dimension values strictly follow the structure of the neural network of Figure A.1

Vectorized Forward Propagation	Vectorized Backward Propagation
$\mathrm{Z}^{[1]} = \mathrm{w}^{[1]}X + b^{[1]}I$ the dimension is $[5,m] = [5,4][4,m] + [5,m]$ $A^{[1]} = \mathrm{g}^{[1]}(Z^{[1]})$ the dimension is $[5,m] = [5,m]$ $Z^{[2]} = \mathrm{w}^{[2]}A^{[1]} + b^{[2]}I$ the dimension is $[6,m] = [6,5][5,m] + [6,m]$ $A^{[2]} = \mathrm{g}^{[2]}(\mathrm{Z}^{[2]})$ the dimension is $[6,m] = [6,m]$ $\mathrm{Z}^{[3]} = \mathrm{w}^{[3]}A^{[2]} + b^{[3]}I$ the dimension is $[5,m] = [5,6][6,m] + [5,m]$ $\hat{Y}_i = \frac{e^{Z_i^{[3]}}}{\sum_{i=1}^{5} e^{Z_i^{[3]}}}$ the dimension is $[5,m] = [5,m]$ Cost function is a scaler: $\mathcal{L}(Y,\hat{Y}) = -\frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{5} y_i^{(j)} \log(\hat{y}_i^{(j)})$	$\frac{\partial \mathcal{L}}{\partial Z^{[3]}} = \frac{1}{m}(\hat{Y} - Y)$ the dimension is $[5,m] = [5,m] - [5,m]$ $\frac{\partial \mathcal{L}}{\partial w^{[3]}} = \frac{\partial \mathcal{L}}{\partial Z^{[3]}} A^{[2]T}$ the dimension is $[5,6] = [5,m][m,6]$ $\frac{\partial \mathcal{L}}{\partial b^{[3]}} = \frac{\partial \mathcal{L}}{\partial Z^{[3]}} I^{T}$ the dimension is $[5,1] = [5,m][m,1]$, where I is a [1,m] dimensional matrix with 1 in each element. This equation is equivalent to summing over m elements of each row of $\frac{\partial \mathcal{L}}{\partial z^{[3]}}$ and obtaining a [5,1] dimensional matrix. $\frac{\partial \mathcal{L}}{\partial Z^{[2]}} = \left(\mathrm{w}^{[3]T} \frac{\partial \mathcal{L}}{\partial Z^{[3]}}\right) \circ \mathrm{g}^{[2]\prime}(Z^{[2]})$ the dimension is $[6,m] = [6,5][5,m] \circ [6,m]$ $\frac{\partial \mathcal{L}}{\partial w^{[2]}} = \frac{\partial \mathcal{L}}{\partial Z^{[2]}} A^{[1]T}$ the dimension is $[6,5] = [6,m][m,5]$ $\frac{\partial \mathcal{L}}{\partial b^{[2]}} = \frac{\partial \mathcal{L}}{\partial Z^{[2]}} I^{T}$ the dimension is $[6,1] = [6,m][m,1]$ $\frac{\partial \mathcal{L}}{\partial Z^{[1]}} = \left(\mathrm{w}^{[2]T} \frac{\partial \mathcal{L}}{\partial Z^{[2]}}\right) \circ \mathrm{g}^{[1]\prime}(Z^{[1]})$ the dimension is $[5,m] = [5,6][6,m] \circ [5,m]$ $\frac{\partial \mathcal{L}}{\partial w^{[1]}} = \frac{\partial \mathcal{L}}{\partial Z^{[1]}} X^{T}$ the dimension is $[5,4] = [5,m][m,4]$ $\frac{\partial \mathcal{L}}{\partial b^{[1]}} = \frac{\partial \mathcal{L}}{\partial Z^{[1]}} I^{T}$ the dimension is $[5,1] = [5,m][m,1]$
Note: The above dimension values are adopted from the neural network structure in Figure 1. There are m training data points. Input x is 4 dimensional. Hidden layer 1 has 5 units. Hidden layer 2 has 6 units. Output y is 5 dimensional. "∘" is element-wise multiplication.	

Table A.2, the formula for forward and backward propagation of an MLP neural network example

A.2 The Recurrent Neural Network (RNN) And Its Vanishing Or Exploding Gradient Problem

The above is the forward and backward-propagation process of the MLP neural network. If the inputs are time-series data and the "order" of the time-series data is important, a RNN network will be used. For example, we use the past 12 month return to predict the next month return. The RNN network structure is illustrated in Figure A.3

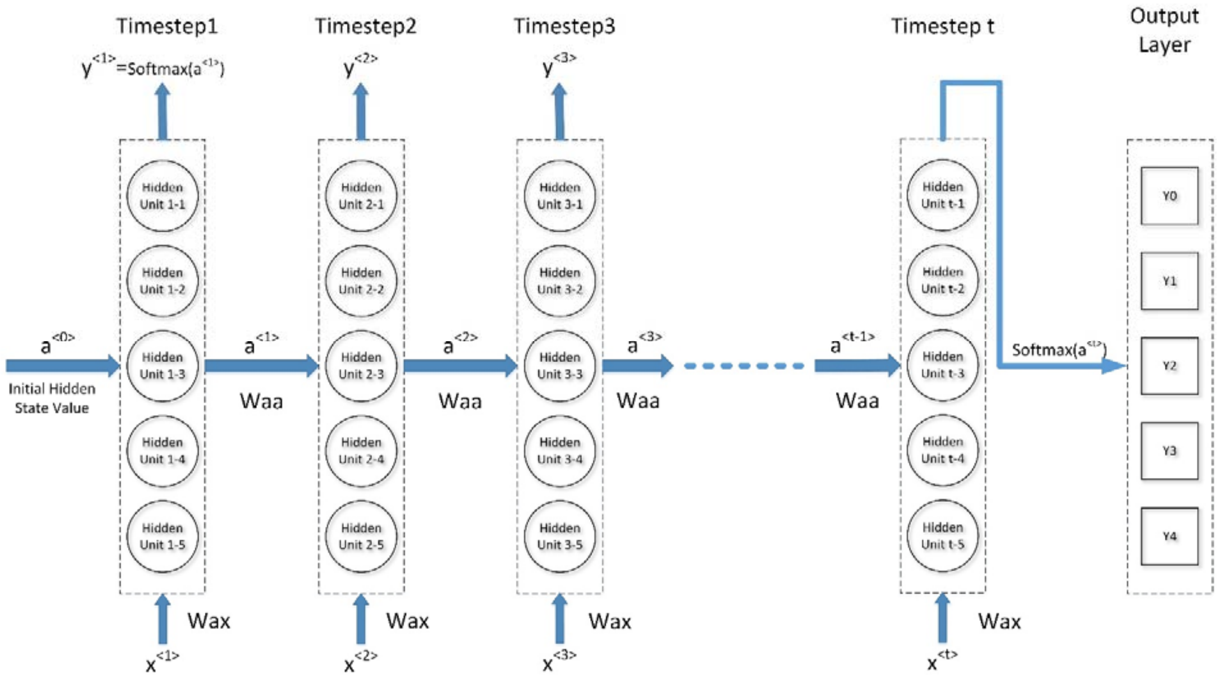


Figure A.3, an example of Recurrent Neural Network (RNN)

RNN is similar to the MLP neural network in terms of the input and output interface. We just need to give the neural network an input vector x, for example, a sequence of 12 returns $\mathrm{Return}_{t-12}, \mathrm{Return}_{t-11}, \ldots, \mathrm{Return}_{t-1}$. The RNN network will produce another sequence of 12 numbers, $a^{<1>}$ to $a^{<12>}$. In the one-layer RNN neural network, we only need $a^{<12>}$ to predict a output $\hat{y} = softmax(a^{<12>})$, in which $\hat{y}$ is the quintile classification of the return in the next period and softmax is a function that convert the multi-dimensional vector $a^{<12>}$ into a 5-dimensional probability vector $\hat{y}$. $a^{<12>}$ contains all the information of the 12 inputs including the order information. If the inputs have t returns, the RNN will have t timesteps. For the notation we use in Figure A.3, $\mathrm{x}^{<1>}$ will be $Return_{t-12}$ and $\mathrm{x}^{<12>}$ will be $Return_{t-1}$. The input $\mathrm{x}^{<i>}$ could also include more than one feature. For example, $\mathrm{x}^{<1>} = [\mathrm{Return}_{t-12}, \mathrm{Volume}_{t-12}]$. The hidden units $\mathrm{a}^{<i>}$ could be seen as an “interpretation” or “memory” of all the previous inputs. In the real training process, from left to right in Figure A.3, the elements of $\mathrm{a}^{<0>}$ are all initialized to be 0 or small random values, which means we have empty memory at the beginning. We then receive the input $\mathrm{x}^{<1>}$ at timestep 1 and generate an interpretaton $\mathrm{a}^{<1>}$ with respect to the inputs with equation (1).

$$\mathrm{a}^{<1>} = \mathrm{g}(\mathrm{w}_{\mathrm{ax}} x^{<1>} + w_{aa} a^{<0>} + b_a) \tag{A.26}$$

g is the activation function and is usually a tanh function. The $\mathrm{a}^{<1>}$ is then passed to the timestep 2. And at timestep t, the hidden layer will synthesize "the memory of last timestep $\mathrm{a}^{<t-1>}$" and "the new feed of information, input $\mathrm{x}^{<t>}$" to generate a new memory $\mathrm{a}^{<t>}$ with equation (2).

$$a^{<t>} = \mathrm{g}(\mathrm{w}_{\mathrm{ax}} x^{<t>} + w_{aa} a^{<t-1>} + b_a) \tag{A.27}$$

At the output layer, the 5-dimensional vector $a^{<t>}$ will be fed to the softmax function.

$$y = \mathrm{softmax}(\mathrm{w}_{\mathrm{ya}} a^{<t>} + b_y) \tag{A.28}$$

From the above feed-forward process of RNN, we can see that there are 2 main differences between the MLP neural network and the RNN.

(1) MLP neural network usually receives all inputs parallelly at once in the first layer of the neural network. In contrast, RNN receives a sequence of input features step by step.

(2) Classic multi-layer neural network usually uses different weights and bias in different layers, while RNN use "a same set of weights and biases" in all timesteps. Of course the output layer is not part of the RNN and has its own weights.

Given these two differences, how do we proceed with the back-propagation? The answer is that it is same with the MLP neural network and we use exactly the same formula in the spirit of Table A.2. However, we need a constraint in the back-propagation process of RNN, which is *the weight of different timesteps must be equal.* To enforce this constraint, we need to sum over or do an average of the derivatives of the weights across different timesteps. In practice, summing over different timesteps is usually used, i.e., $\frac{\partial \mathcal{L}}{\partial w} = \sum_{t=1}^{12} \frac{\partial \mathcal{L}}{\partial w^{<t>}}$ for our exampe. This process is called back propagration through time (BPTT).

However, we have a further problem called Vanishing or Exploding Gradient in RNN. This Vanishing or Exploding Gradient problem is the reason why RNN cannot be widely used in the early years and why researchers invent the LSTM. We will illustrate the problem from the following equations.

$$\begin{aligned} \mathrm{a}^{<1>} &= \mathrm{g}(\mathrm{w}_{\mathrm{ax}} x^{<1>} + w_{aa} a^{<0>} + b_a) \\ \mathrm{a}^{<t>} &= \mathrm{g}(\mathrm{w}_{\mathrm{ax}} x^{<t>} + w_{aa} a^{<t-1>} + b_a) \end{aligned} \tag{A.29}$$

Suppose the left most return at timestep 1 is very important in determining the return at time t and t is large. In this case, we will need a long-term dependency. So we need the errors generated at time t to propagate all the way back to the left and affect the weights updates at the leftmost timestep 1. As a result, we need to calculate terms like $\partial L / \partial a^{<1>}$, in which, L is the cost function and $a^{<1>}$ is the timestep 1's activation. From chain rule of derivatives, to calculate $\partial L / \partial a^{<1>}$ we need to know $\partial L / \partial a^{<t>}$ and $\partial a^{<t>} / \partial a^{<1>}$ (t is the last timestep). We know that $\partial L / \partial a^{<t>}$ can be calculated from the errors in the output layer, i.e. $\partial L / \partial a^{<t>} = w_{ya}^{T} \, \partial L / \partial z^{<output>}$, where $z^{<output>} = \mathrm{w}_{\mathrm{ya}} a^{<t>} + b_y$. So all we need to calculate is $\partial a^{<t>} / \partial a^{<1>}$. Applying derivative recursively on equation (A.29), we have

$$\begin{aligned} \frac{\partial a^{<t>}}{\partial a^{<1>}} &= \frac{\partial a^{<t>}}{\partial a^{<t-1>}} \frac{\partial a^{<t-1>}}{\partial a^{<t-2>}} \cdots \frac{\partial a^{<2>}}{\partial a^{<1>}} \\ &= \mathrm{g}'(\mathrm{z}^{<t>}) \mathrm{g}'(\mathrm{z}^{<t-1>}) \ldots \mathrm{g}'(\mathrm{z}^{<2>}) \, (\mathrm{w}_{aa})^{t-1} \end{aligned} \tag{A.30}$$

g' is the derivatives of the activation function. $\mathrm{z}^{<t>} = \mathrm{w}_{\mathrm{ax}} x^{<t>} + w_{aa} a^{<t-1>} + b_a$. No matter what activation function we use, the maximum of function g's derivative is 1. For the activation function tanh and relu, the maximum of their derivative is 1. For the activation function sigmoid, the maximum of its derivative is 1/4. As a result,

$$\frac{\partial a^{<t>}}{\partial a^{<1>}} = \mathrm{g}'(\mathrm{z}^{<t>})\mathrm{g}'(\mathrm{z}^{<t-1>}) \ldots \mathrm{g}'(\mathrm{z}^{<2>})\ (\mathrm{w}_{aa})^{t-1} \leq 1 * \ (\mathrm{w}_{aa})^{t-1} \quad \text{(A.31)}$$

If $W_{aa} < 1$, as t goes big, $\frac{\partial a^{<t>}}{\partial a^{<1>}}$ will go to zero, so does $\partial L / \partial a^{<1>}$. If $W_{aa} > 1,$ as t goes big, $\frac{\partial a^{<t>}}{\partial a^{<1>}}$ will go to infinity, so does $\partial L / \partial a^{<1>}$. As a result, the earlier timestep will have no influence on the weights updates or the weights will explode. This is the problem of Vanishing or Exploding Gradient .

A.3 Long Short-Term Memory(LSTM) And How It Solves The Vanishing Or Exploding Gradient Problem

To tackle the Vanishing or Exploding Gradient problem, essentially we need w_{aa} in equation (A.31) to be 1. To achieve this, Long Short-Term Memory(LSTM) employs an cell internal state (or memory) vector $c^{<t>}$. So in each time step, we have a cell state vector $c^{<t>}$ and an output vector $a^{<t>}$ and they have the same dimension. The Vanilla RNN as we showed in last section simply let them equal. $c^{<t>}$ is responsible for perserving the memory and $a^{<t>}$ is responsible for the outputs (to the next time step and to the next layer). The LSTM network has three "gates" in each timestep: the forget gate f_t, the update gate u_t and the output gate o_t. Firstly, we need to calculate a condidate of the memory cell.

$$\tilde{c}_t = \tanh(W_c[a_{t-1}, x_t] + b_c) \quad \text{(A.32)}$$

This equation is almost same as $\mathrm{a}^{<t>} = \mathrm{g}(w_{ax}x^{<t>} + w_{aa}a^{<t-1>} + b_a)$ in RNN. $\tilde{c}_t$is derived from time t-1 output a_{t-1} and time t input x_t. The nonation of $[a_{t-1}, x_t]$ is the concatation of a_{t-1} and x_t and is just for simplifing the expression. Similarly W_c is a concatation of w_{ax} and

w_{aa}. $\tilde{c}_t$ can be interpreted as the fresh information and serve as a candidate for the real c_t. The real c_t will be determined by the $\tilde{c}_t$ and c_{t-1} and two gates.

$$\begin{aligned} f_t &= \sigma\left(W_f[a_{t-1}, x_t] + b_f\right) \\ u_t &= \sigma(W_u[a_{t-1}, x_t] + b_u) \\ c_t &= u_t \circ \tilde{c}_t + f_t \circ c_{t-1} \end{aligned} \tag{A.33}$$

In which, f_t is the forget gate and σ is the sigmoid function. u_t is the update gate. f_t and u_t will have the same dimension as the internal state vector c_t. Please note that "∘" is element-wise multiplication. Although the sigmoid function could have value any where between 0 and 1, in practice, the $W_f[a_{t-1}, x_t] + b_f$ will be either very positive or very negative for most of the time. So the f_t and $u_t$will be close to either 0 or 1. The interpretatoin of the above equation is that the internal state or memory will be determined by the old memory c_{t-1} as well as the new information $\tilde{c}_t$. We will forget some of the old memory which is determined by the element-wise multiplication of $f_t \circ c_{t-1}$. And we will add some new information $\tilde{c}_t$ to the memory cell c_t which is determined by $u_t \circ \tilde{c}_t$. For f_t, 1 means to remember and 0 means to forget. For u_t, 1 means to update and 0 means not to update.

Finally, we need to calculate the output a_t. The a_t will feed to the next timestep or the output layer or and next LSTM layer.

$$\begin{aligned} o_t &= \sigma(W_o[a_{t-1}, x_t] + b_o) \\ a_t &= o_t \circ \tanh(c_t) \end{aligned} \tag{A.34}$$

o_t is the output gate. We mentioned in the RNN paragraph that "To tackle the Vanishing or Exploding Gradient problem, essentially we need w_{aa} in equation (6) to be 1." w_{aa} is the weight between the memory cell at time t-1 and time t in the back-propagation and is the reason for the Vanishing or Exploding Gradient . Here in LSTM, the w$_{aa}$ is equavalent to the weights between c_{t-1} and c_t, which is f_t. The f_t is a vector like [1,1,1,0,1] if the memory cell is 5-dimensional. From $c_t = u_t \circ \tilde{c}_t + f_t \circ c_{t-1}$, we can see that the weight between c_{t-1} and c_t will be 1 if the forget gate determines to preserve the cell memory. An intuition of how LSTM tackle the Vanishing or Exploding Gradient problem is illustrated in the Figure A.4.

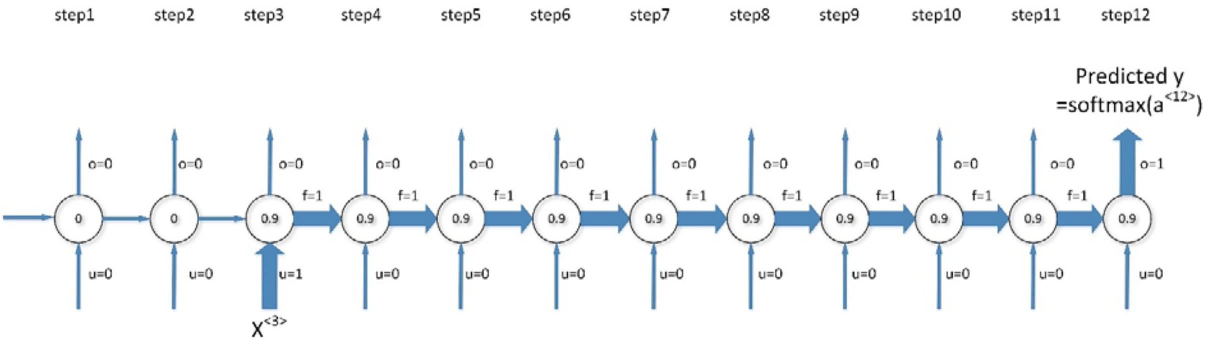


Figure A.4, an intuitive illustration of how LSTM neural network solved the Vanishing or Exploding Gradient problem

In Figure A.4, u is update gate. f is forget gate. o is output gate. There are 12 timesteps. We have an input $x^{<3>}$ at timestep 3 that activates one of the memory cell to the value of 0.9. During the forward propagation process from timestep 3 to the timestep 12, the update gates for the cell are all closed and the forget gates are all open (means to remember). The output gates are all closed except for timestep 12. In this process, the information will flow along the thick line and every gates along the thick line have value 1. The error message will be generated at the position of "predicted $\hat{y}$" in timestep 12 and propagates back to timestep 3. All the weights between the memory cell c_{t-1} and c_t along the path will be 1. So in this way, LSTM avoid the Vanishing or Exploding Gradient problem in the back-propagation.

Reference

Asness, C. S., A. Frazzini and L. H. Pedersen, (2017) Quality Minus Junk. Available at SSRN: https://ssrn.com/abstract=2312432

Cochrane, J. H. (2011), Presidential Address: Discount Rates. The Journal of Finance, 66: 1047-1108.

Fama, E. F., and K. R. French. (1997) Industry costs of equity, Journal of Financial Economics, Volume 43, Issue 2, 153-193

Fama, E. F., and K. R. French. (2006). Profitability, investment and average returns. Journal of Financial Economics 82:491–518.

Fama, E. F., and K. R. French. (2008). Dissecting anomalies. Journal of Finance 63:1653–78.

Fischer, T. and C. Krauss (2017) Deep learning with long short-term memory networks for financial market predictions, European Journal of Operational Research, ISSN 0377-2217, https://doi.org/10.1016/j.ejor.2017.11.054.

Green, J., J. R. M. Hand and X. F. Zhang (2013). The supraview of return predictive signals. Review of Accounting Studies 18 (3), 692-730.

Harvey, C. R., Y. Liu and H. Zhu (2016) … and the Cross-Section of Expected Returns, *The Review of Financial Studies*, Volume 29, Issue 1, Pages 5–68,

Heaton, J. B., N. G. Polson and J. H. Witte. (2016), Deep Learning in Finance, Available at: https://arxiv.org/abs/1602.06561

Hirshleifer, D., K. Hou, S. H. Teoh, and Y. Zhang. (2004). Do investors overvalue firms with bloated balance sheets? Journal of Accounting and Economics 38:297–331.

Hochreiter, S. and J. Schmidhuber (1997). Long short-term memory. Neural Computation 9 (8), 1735-1780.

Huck, N., (2009). Pairs selection and outranking: An application to the S&P 100 index. European Journal of Operational Research 196 (2), 819-825.

Jacobs, H., (2015). What explains the dynamics of 100 anomalies? Journal of Banking & Finance 57, 65-85.

Jegadeesh, N., (1990). Evidence of predictable behavior of security returns. The Journal of Finance 45 (3), 881.

Jegadeesh, N. and S. Titman (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. The Journal of Finance 48 (1), 65-91.

Krauss, C., X. A. Do and N. Huck (2017). Deep neural networks, gradient-boosted trees, random forests: Statistical arbitrage on the S&P 500. European Journal of Operational Research 259 (2), 689-702.

LeCun, Y., Y. Bengio and G. Hinton (2015). Deep learning. Nature 521 (7553), 436-444.

Light, N., D. Maslov and O. Rytchkov (2017) Aggregation of Information About the Cross Section of Stock Returns: A Latent Variable Approach, *The Review of Financial Studies*, Volume 30, Issue 4, Pages 1339–1381

Moskowitz, T. J. and M. Grinblatt (1999), Do Industries Explain Momentum? The Journal of Finance, 54: 1249-1290.

Novy-Marx, R. (2012) Is momentum really momentum? Journal of Financial Economics, Volume 103, Issue 3, 429-453

Piotroski, J. (2000). Value Investing: The Use of Historical Financial Statement Information to Separate Winners from Losers. *Journal of Accounting Research, 38*, 1-41.

Schmidhuber, J., (2015). Deep learning in neural networks: An overview. Neural Networks 61, 85-117.

Sloan, R. G. (1996). Do stock prices fully reflect information in accruals and cash flows about future earnings? Accounting Review 71:289–315.

Takeuchi, L. and Y. Y. Lee (2013). Applying deep learning to enhance momentum trading strategies in stocks. Working paper, Stanford University.

Table 2: The Return And 3-Factor α of 2-Layer LSTM Trading Strategy with Past 80 Days' Return As Input, Before The Trading Costs

Panel A and B reports the result of equal-weighted portfolio and Panel C and D reports the result of value-weighted portfolio. Panel A and C are the return and the Fama French 3-factor α of 2-layer LSTM trading strategy with individual stocks' returns as inputs. Panel B and D are the factor loadings of the Fama French 3-factors. All results are before the trading costs and cover the period from 1971 through 2017. We employ the past 80 days' return as input and use them to predict the next day's return. We long (short) the 10% stocks with the highest probability to outperform (underperform) in the next day. We follow Jeehadesh and Titman (1993) for the definition of holding period. If the holding period is n days, we will rebalance 1/n of the portfolio every day and hold the 1/n stocks for n days. The Newey-West t-statistics are reported below each number.

Panel A: The monthly equal-weighted return and the Fama French 3-factor α of 2-layer LSTM trading strategy before the trading costs. The Newey-West t-statistics are reported below each number.

	Holding Periods							
Year	1-day holding period return	1-day holding period α	5-day holding period return	5-day holding period α	10-day holding period return	10-day holding period α	21-day holding period return	21-day holding period α
All years	29.58% (26.81)	29.50% (25.60)	6.65% (17.12)	6.68% (17.32)	3.61% (12.60)	3.62% (12.72)	1.75% (9.04)	1.77% (9.17)
1971-1980	18.94% (42.38)	19.33% (47.35)	7.43% (15.51)	7.70% (15.97)	4.47% (10.90)	4.65% (10.89)	2.24% (7.95)	2.23% (7.46)
1981-1990	38.02% (19.62)	40.16% (20.02)	7.59% (17.81)	7.80% (17.07)	4.04% (17.22)	4.13% (17.43)	1.98% (15.41)	2.03% (15.04)
1991-2000	63.00% (27.39)	63.29% (24.02)	11.15% (22.61)	11.19% (22.73)	5.83% (17.33)	5.91% (17.99)	2.71% (11.82)	2.75% (11.69)
2001-2010	12.99% (20.55)	12.89% (20.72)	3.54% (9.11)	3.51% (8.93)	1.86% (6.36)	1.84% (6.07)	0.99% (4.59)	0.98% (4.81)
2011-2017	5.21% (15.99)	5.20% (15.12)	1.44% (6.59)	1.47% (6.60)	0.64% (4.40)	0.65% (4.53)	0.22% (2.31)	0.23% (2.33)

Panel B: The factor loadings of equal-weighted returns of 2-layer LSTM trading strategy before the trading costs. The return in the 3-factor regression is from all samples from 1971 to 2017. The Newey-West t-statistics are reported below each number.

Factors	1-day holding period	5-day holding period	10-day holding period	21-day holding period
3-factor α	29.50% (25.60)	6.68% (17.32)	3.62% (12.72)	1.77% (9.17)
Beta of Mkt-Rf	0.169 (0.63)	-0.057 (-1.63)	-0.037 (-1.60)	-0.031 (-2.14)
Beta of SMB	-0.672 (-1.79)	-0.163 (-2.84)	-0.083 (-2.05)	-0.033 (-1.13)
Beta of HML	-0.021 (-0.05)	0.079 (1.20)	0.054 (1.42)	0.009 (0.43)

Table 2 Continued

Panel C: The monthly value-weighted return and the Fama French 3-factor α of 2-layer LSTM trading strategy before trading costs. The Newey-West t-statistics are reported below each number.

	Holding Periods							
Year	1-day holding period return	1-day holding period α	5-day holding period return	5-day holding period α	10-day holding period return	10-day holding period α	21-day holding period return	21-day holding period α
All years	19.34% (23.71)	19.19% (22.60)	5.28% (14.24)	5.31% (14.34)	2.81% (10.25)	2.83% (10.23)	1.37% (7.60)	1.40% (7.77)
1971-1980	15.27% (32.37)	15.69% (36.12)	6.43% (13.96)	6.71% (13.66)	3.74% (8.99)	3.95% (8.98)	1.76% (5.51)	1.92% (5.86)
1981-1990	24.86% (22.16)	26.14% (22.29)	6.05% (15.19)	6.32% (15.47)	3.22% (13.83)	3.30% (13.77)	1.64% (12.63)	1.68% (12.66)
1991-2000	43.11% (22.14)	43.47% (19.59)	9.42% (16.30)	9.37% (16.29)	4.84% (10.70)	4.78% (10.58)	2.13% (7.01)	2.07% (6.84)
2001-2010	5.04% (9.69)	5.21% (9.55)	2.12% (7.07)	2.12% (7.05)	1.06% (3.85)	1.10% (3.87)	0.72% (2.95)	0.71% (2.80)
2011-2017	1.10% (2.30)	1.07% (2.17)	0.43% (2.13)	0.47% (2.09)	0.17% (1.02)	0.15% (0.98)	0.11% (0.99)	0.13% (0.99)

Panel D: The factor loading of value-weighted returns of 2-layer LSTM trading strategy. The return in the 3-factor regression is all samples from 1971 to 2017. The t-statistics are reported below each number.

Factors	1-day holding period	5-day holding period	10-day holding period	21-day holding period
3-factor α	19.19% (22.60)	5.31% (14.34)	2.83% (10.23)	1.40% (7.77)
Beta of Mkt-Rf	0.103 (0.52)	-0.080 (-1.34)	-0.058 (-1.76)	-0.052 (-2.33)
Beta of SMB	-0.634 (-2.29)	-0.214 (-3.51)	-0.133 (-2.48)	-0.054 (-1.23)
Beta of HML	0.218 (0.73)	0.145 (2.05)	0.115 (2.06)	0.042 (1.10)

Figure 5: The Time-Series of Monthly Equal-Weighted Return of 2-Layer LSTM Trading Strategy For 1-Day And 5-Days Holding Period Before The Trading Cost

This figure depicts the time-series of monthly equal-weighted return of 2-layer LSTM trading strategy for 1-day and 5-days holding period before the trading cost. The sample covers the period from 1971 through 2017. We employ the past 80 days' return as input and use them to predict the next day's return. We long (short) the 10% stocks with the highest probability to outperform (underperform) in the next day. We follow Jeehadesh and Titman (1993) for the definition of holding period. If the holding period is n days, we will rebalance 1/n of the portfolio every day and hold the 1/n stocks for n days.

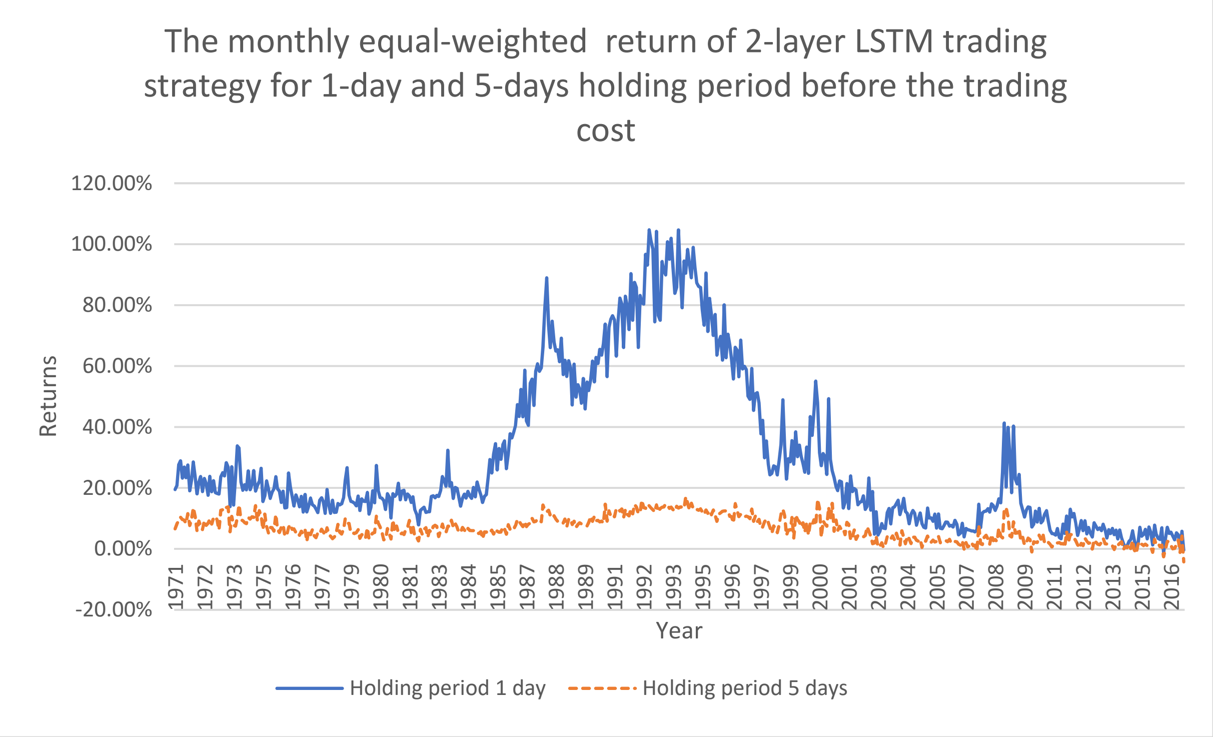


Table 3: The Monthly Return And The Fama-French 3-Factor A of 2-Layer LSTM Trading Strategy of 3 Size Subgroups, Before The Trading Costs

This table reports the equal-weighted monthly return and the Fama-French 3-factor α of 2-layer LSTM trading strategy of 3 size subgroups before the trading costs. We use the NYSE 20% and 50% as the size breakpoints in each day. We use the stock's size at the end of day(t-1) to compare to the breakpoints. The sample covers the period from 1971 through 2017. We employ the past 80 days' return as input and use them to predict the next day's return. We still use all the stocks in the training process to adjust the weights and use the weights for all three size subgroups. The Newey-West t-statistics are reported below each number.

The monthly equal-weighted return and the Fama-French 3-factor α of the 2-layer LSTM trading strategy of 3 size subgroups. We use the NYSE 20% and 50% as the size breakpoints. The Newey-West t-statistics are reported below each number.

	Holding Periods							
Size subgroups	1-day holding period return	1-day holding period α	5-day holding period return	5-day holding period α	10-day holding period return	10-day holding period α	21-day holding period return	21-day holding period α
All stocks	29.58% (26.81)	29.50% (25.60)	6.65% (17.12)	6.68% (17.32)	3.61% (12.60)	3.62% (12.72)	1.75% (9.04)	1.77% (9.17)
Micro stocks	48.09% (25.47)	48.27% (24.46)	9.57% (18.80)	9.58% (18.93)	5.24% (14.40)	5.25% (14.22)	2.64% (10.84)	2.66% (10.87)
Small Stocks	15.45% (30.35)	15.26% (27.78)	4.67% (13.55)	4.63% (13.36)	2.61% (9.24)	2.56% (9.15)	1.22% (6.07)	1.20% (6.01)
Big stocks	5.91% (26.46)	5.92% (25.50)	2.07% (11.66)	2.10% (11.69)	1.04% (7.43)	1.07% (7.57)	0.43% (4.27)	0.45% (4.42)

Figure 6: The Time-Series Of Monthly Equal-Weighted Return Of The 2-Layer LSTM Trading Strategy For Three Size Subgroup

This figure depicts the time-series of monthly equal-weighted return of the 2-layer LSTM trading strategy for three size subgroup before the trading costs. The returns is for 1-day holding period. The sample covers the period from 1971 through 2017. We employ the past 80 days' return as input and use them to predict the next day's return. We still use all the stocks in the training process to adjust the weights and use the same weights for all three size subgroups.

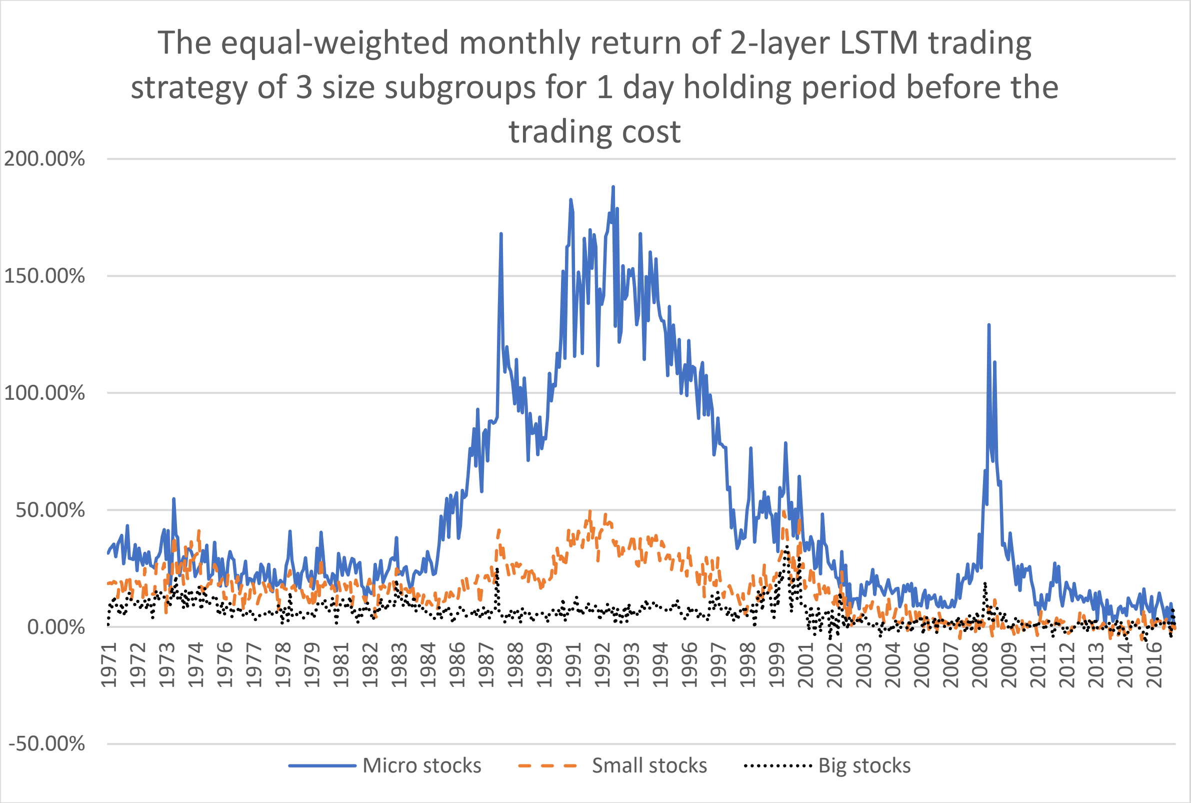


Figure 7: The Average Z-Score Of Return Before And After The Portfolio Formation Day Of The 2-Layer LSTM Trading Strategy For Individual Stocks

This figure depicts the average z-score of return before and after the portfolio formation day of the 2-Layer LSTM trading strategy for individual stocks. The sample covers the period from 1971 through 2017. We employ the past 80 days' return as input and use them to predict the next day's return. We long (short) the 10% stocks with the highest probability to outperform (underperform) in the next day. The x-axis is the days before and after the portfolio formation, in which 0 represents the portfolio formation day(t). The y-axis is the z-score of the returns of the long and short portfolio. The solid orange line is the long portfolio and the dotted blue line is the short portfolio.

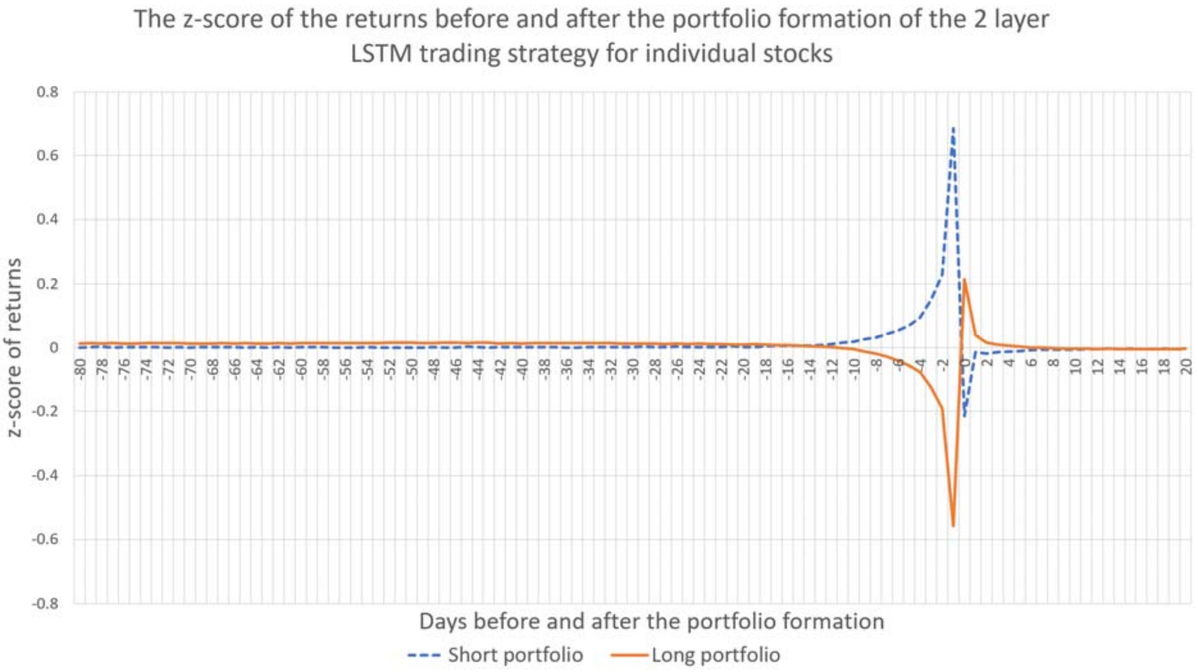


Table 4: The Proportional Effective Spread Of 10 Size And 10 Idiosyncratic Volatility Groups For Each Year From 1993 To 2017

This table reports the Proportional Effective Spread (PES) of 10 size and 10 idiosyncratic volatility groups for each year from 1993 to 2017. In each March and September, we rank stocks independently into 10 size and 10 idiosyncratic volatility(IVOL) portfolios according to their size and IVOL at the end of last month. The results in each year in this table are the average of the PES in March and September. Panel A reports the PES for each size decile for every year. Panel B reports the PES for each idiosyncratic decile for every year. In our test, we use the PES of the 10x10 portfolio as the proxy for trading costs. The PES of the 10x10 portfolio is also different in each March and September.

Panel A: The Proportional Effective Spread of 10 size groups from 1993 to 2017

	Size Decile									
Year	Smallest	2	3	4	5	6	7	8	9	Biggest
1993	3.002%	2.489%	2.038%	1.684%	1.351%	1.140%	0.786%	0.672%	0.509%	0.320%
1994	3.378%	2.726%	2.172%	1.749%	1.472%	1.245%	0.913%	0.719%	0.574%	0.366%
1995	2.971%	2.182%	1.834%	1.550%	1.297%	1.072%	0.758%	0.621%	0.489%	0.315%
1996	2.841%	2.205%	1.849%	1.341%	1.253%	0.971%	0.814%	0.626%	0.453%	0.294%
1997	3.025%	2.185%	1.711%	1.469%	1.167%	0.945%	0.705%	0.581%	0.405%	0.259%
1998	2.837%	2.022%	1.708%	1.424%	1.064%	0.882%	0.673%	0.517%	0.378%	0.229%
1999	3.074%	2.167%	1.688%	1.353%	1.068%	0.863%	0.644%	0.462%	0.396%	0.222%
2000	2.864%	2.143%	1.536%	1.268%	1.040%	0.783%	0.603%	0.493%	0.376%	0.240%
2001	2.736%	1.783%	1.356%	0.859%	0.696%	0.539%	0.389%	0.297%	0.213%	0.156%
2002	1.807%	1.115%	0.767%	0.539%	0.373%	0.281%	0.221%	0.161%	0.134%	0.100%
2003	1.594%	1.036%	0.779%	0.503%	0.383%	0.285%	0.206%	0.136%	0.100%	0.066%
2004	1.795%	0.939%	0.546%	0.351%	0.231%	0.166%	0.133%	0.089%	0.070%	0.045%
2005	1.600%	0.804%	0.377%	0.253%	0.178%	0.125%	0.099%	0.080%	0.059%	0.039%
2006	1.338%	0.624%	0.298%	0.188%	0.137%	0.102%	0.079%	0.068%	0.051%	0.035%
2007	1.143%	0.544%	0.278%	0.179%	0.125%	0.092%	0.070%	0.059%	0.042%	0.029%
2008	2.226%	0.871%	0.413%	0.252%	0.157%	0.113%	0.091%	0.078%	0.056%	0.038%
2009	2.199%	0.605%	0.316%	0.199%	0.154%	0.105%	0.085%	0.065%	0.051%	0.038%
2010	1.497%	0.412%	0.242%	0.152%	0.112%	0.077%	0.065%	0.049%	0.037%	0.030%
2011	1.493%	0.428%	0.211%	0.150%	0.115%	0.082%	0.069%	0.052%	0.041%	0.029%
2012	1.529%	0.375%	0.235%	0.130%	0.096%	0.070%	0.059%	0.045%	0.034%	0.023%
2013	1.526%	0.396%	0.233%	0.144%	0.103%	0.081%	0.058%	0.046%	0.033%	0.023%
2014	1.232%	0.394%	0.213%	0.142%	0.110%	0.080%	0.064%	0.056%	0.031%	0.023%
2015	1.549%	0.408%	0.264%	0.153%	0.108%	0.087%	0.067%	0.052%	0.034%	0.023%
2016	1.239%	0.386%	0.184%	0.121%	0.092%	0.076%	0.060%	0.045%	0.029%	0.021%
2017	1.485%	0.436%	0.234%	0.171%	0.123%	0.086%	0.062%	0.044%	0.029%	0.020%

Table 4 Continued

Panel B: The Proportional Effective Spread of 10 idiosyncratic volatility groups from 1993 to 2017

	Idiosyncratic Volatility Decile									
Year	Smallest	2	3	4	5	6	7	8	9	Biggest
1993	0.464%	0.527%	0.690%	0.924%	1.051%	1.349%	1.635%	1.925%	2.304%	3.114%
1994	0.526%	0.646%	0.806%	0.979%	1.191%	1.509%	1.636%	1.930%	2.613%	3.477%
1995	0.441%	0.556%	0.745%	0.933%	1.105%	1.220%	1.480%	1.686%	2.090%	2.827%
1996	0.428%	0.576%	0.801%	0.873%	1.012%	1.291%	1.489%	1.694%	1.988%	2.493%
1997	0.388%	0.522%	0.785%	0.955%	1.225%	1.330%	1.473%	1.480%	1.930%	2.359%
1998	0.460%	0.630%	0.845%	0.993%	1.133%	1.274%	1.436%	1.553%	1.555%	1.851%
1999	0.608%	0.780%	0.938%	1.067%	1.163%	1.247%	1.338%	1.470%	1.764%	1.562%
2000	0.745%	0.928%	1.079%	1.162%	1.187%	1.167%	1.209%	1.101%	1.218%	1.546%
2001	0.550%	0.533%	0.644%	0.796%	0.985%	1.151%	1.086%	1.101%	1.043%	1.134%
2002	0.312%	0.333%	0.409%	0.454%	0.537%	0.588%	0.673%	0.624%	0.751%	0.812%
2003	0.247%	0.291%	0.317%	0.410%	0.498%	0.544%	0.607%	0.629%	0.716%	0.824%
2004	0.136%	0.188%	0.261%	0.345%	0.359%	0.456%	0.501%	0.547%	0.638%	0.932%
2005	0.095%	0.145%	0.199%	0.271%	0.344%	0.321%	0.440%	0.400%	0.589%	0.810%
2006	0.098%	0.130%	0.162%	0.229%	0.238%	0.296%	0.327%	0.339%	0.473%	0.624%
2007	0.079%	0.113%	0.156%	0.175%	0.207%	0.239%	0.297%	0.345%	0.353%	0.594%
2008	0.140%	0.141%	0.189%	0.373%	0.283%	0.469%	0.381%	0.594%	0.653%	1.062%
2009	0.064%	0.134%	0.149%	0.147%	0.248%	0.359%	0.450%	0.541%	0.695%	1.025%
2010	0.041%	0.067%	0.088%	0.156%	0.220%	0.234%	0.273%	0.367%	0.504%	0.724%
2011	0.042%	0.068%	0.103%	0.144%	0.190%	0.252%	0.309%	0.356%	0.422%	0.781%
2012	0.035%	0.066%	0.095%	0.161%	0.193%	0.239%	0.247%	0.433%	0.468%	0.657%
2013	0.037%	0.069%	0.105%	0.155%	0.203%	0.327%	0.270%	0.361%	0.431%	0.683%
2014	0.039%	0.062%	0.098%	0.164%	0.186%	0.241%	0.311%	0.363%	0.400%	0.480%
2015	0.043%	0.064%	0.120%	0.159%	0.201%	0.282%	0.361%	0.428%	0.461%	0.624%
2016	0.034%	0.055%	0.111%	0.153%	0.178%	0.230%	0.280%	0.361%	0.371%	0.474%
2017	0.035%	0.059%	0.106%	0.160%	0.186%	0.219%	0.361%	0.411%	0.497%	0.654%

Figure 8: The Proportional Effective Spread Of 10x10 Portfolios For Size And Idiosyncratic Volatility

This figure depicts the Proportional Effective Spread of 10x10 portfolios for size and idiosyncratic volatility. The data covers the period from 1993 to 2017. In each March and September, we rank stocks independently into 10 size and 10 idiosyncratic volatility (IVOL) portfolios according to their size and IVOL at the end of last month. The results in the picture is the average PES from 1993 to 2017 of the 10x10 portfolio. In our test, we use different PES of the 10x10 portfolio for different year and month. The label 0.02 in the z-axis (vertical axis) means a PES level of 2%.

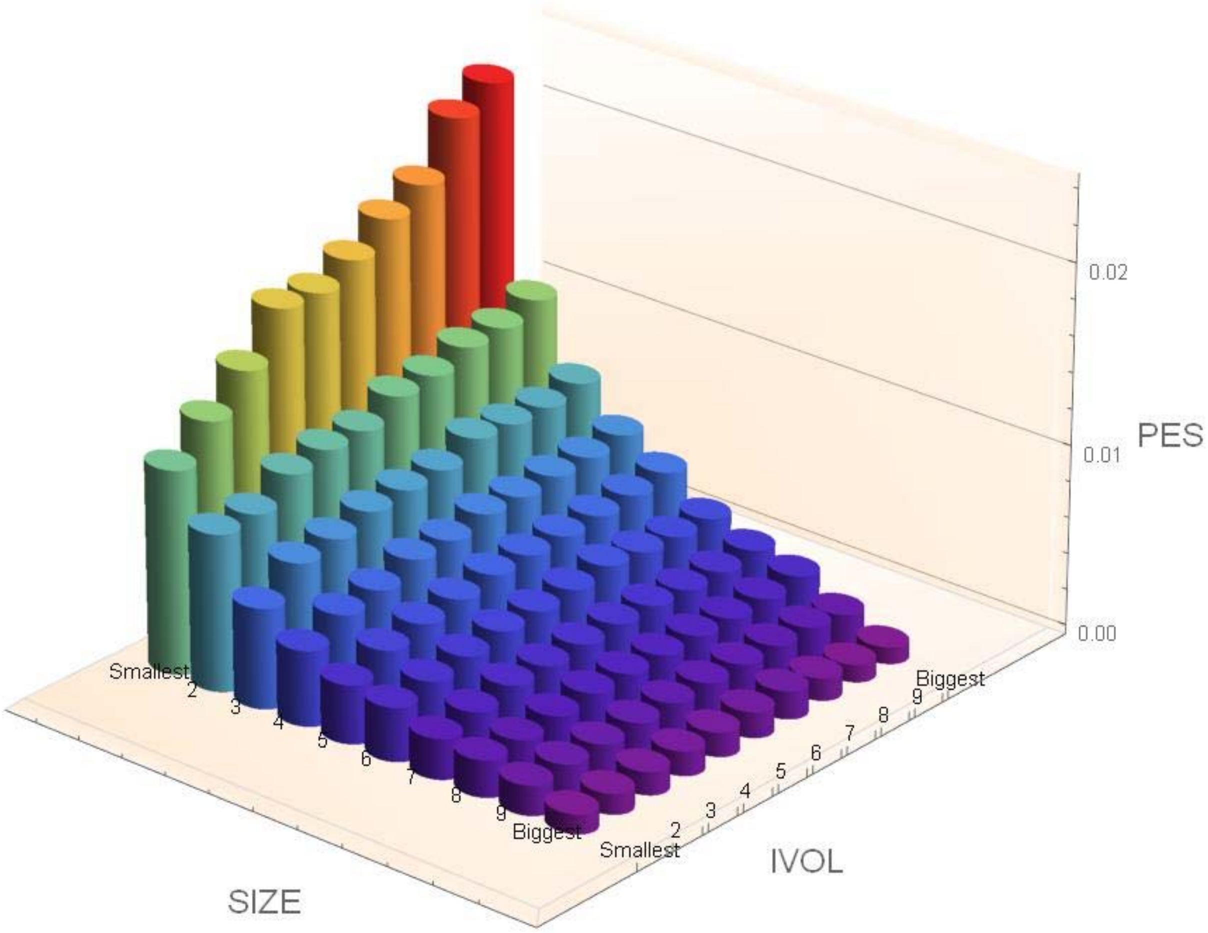


Table 5: The Monthly Equal-Weighted And Value-Weighted Return of the 2-Layer LSTM Trading Strategy Before And After the Trading Costs

This table reports the monthly equal-weighted and value-weighted return of 2-layer LSTM trading strategy before and after the trading costs. In each March and September, we estimate the average Proportional Effective Spread (PES) for each stock. To avoid the look-ahead bias, the March PES is used for the 6 months from April through September of the same year and the September PES is used for the 6 months from October through next March. If the stock's PES is not missing, we will use it. If a stock's PES is missing, we estimate its PES from its size and idiosyncratic volatility (IVOL). In each month, we rank stocks independently into 10 size and 10 IVOL portfolios according to their size and IVOL at the end of last month. We use the average Proportional Effective Spread (PES) of 10x10 size and IVOL portfolio as the proxy for the trading costs of the stocks in each portfolios. The trading costs estimates update every half year. We do not report FF-3 α in this table because Table 2 already shows that the before trading cost returns have low factor loading on the Fama French 3 factors. FF-3 factors are all before trading cost returns. Regressing an after trading cost return on before trading cost factors is not very meaningful. TAQ database starts from 1993, so our data covers from 1993 through 2017. The Newey-West t-statistics are reported below each number.

Panel A: The monthly equal-weighted return of 2-layer LSTM trading strategy before and after trading cost. The data is from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods			
Equal-Weighted Returns	1-day holding period return	5-day holding period return	10-day holding period return	21-day holding period return
Before-trading-costs return for all years	26.03% (16.12)	5.41% (8.94)	2.80% (6.46)	1.33% (4.60)
After-Trading-Costs return for all years	-6.66% (-17.61)	-0.27% (-1.12)	-0.041% (-0.22)	-0.027% (-0.19)

Panel B: The monthly value-weighted return of 2-layer LSTM trading strategy before and after trading cost. The data is from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods			
Value-Weighted Returns	1-day holding period return	5-day holding period return	10-day holding period return	21-day holding period return
Before-trading-costs return for all years	15.55% (12.22)	3.99% (6.92)	2.01% (4.83)	0.99% (3.78)
After-Trading-Costs return for all years	0.64% (1.61)	1.47% (5.59)	0.75% (3.47)	0.39% (2.50)

Table 6: The Monthly Equal-Weighted Return And The Fama-French 3-Factor α Of 2-Layer LSTM Trading Strategy For 17 Industry Portfolios

This table reports the monthly equal-weighted return and the Fama-French 3-factor α of 2-layer LSTM trading strategy for 17 industry portfolios. The sample covers periods from 1971 to 2017. In each day, we long (short) 3 industries with the highest probability to outperform (underperform). The definition of holding period is same as the first test in Table 2. The profit is before the trading cost. The Newey-West t-statistics are reported below each number.

Panel A: The monthly equal-weighted return and the 3 Fama French 3-factor α of 2-layer LSTM trading strategy with industry portfolio from 1971 to 2017, before the trading costs. The Newey-West t-statistics are reported below each number.

	Holding Periods			
Equal-weighted return	1-day holding period return	5-day holding period return	10-day holding period return	21-day holding period return
Raw Return	2.02% (11.21)	0.77% (5.47)	0.58% (4.31)	0.29% (2.77)
3-factor α	1.84% (9.91)	0.71% (4.82)	0.56% (3.94)	0.28% (2.56)

Figure 9: The Z-Score Of The Returns Before And After The Portfolio Formation Of The 2-Layer LSTM Trading Strategy For 17 Industry Portfolios

This figure depicts the z-score of the returns before and after the portfolio formation of the 2-layer LSTM trading strategy for 17 industry portfolios. The sample covers periods from 1971 to 2017. In each day, we long (short) 3 industries with the highest probability to outperform (underperform). The x-axis is the days before and after the portfolio formation, in which 0 represents the portfolio formation day(t). The y-axis is the z-score of the returns of the long and short portfolio. The solid orange line is the z-score of the returns of the long portfolio. The dotted blue line is the z-score of the returns of the short portfolio.

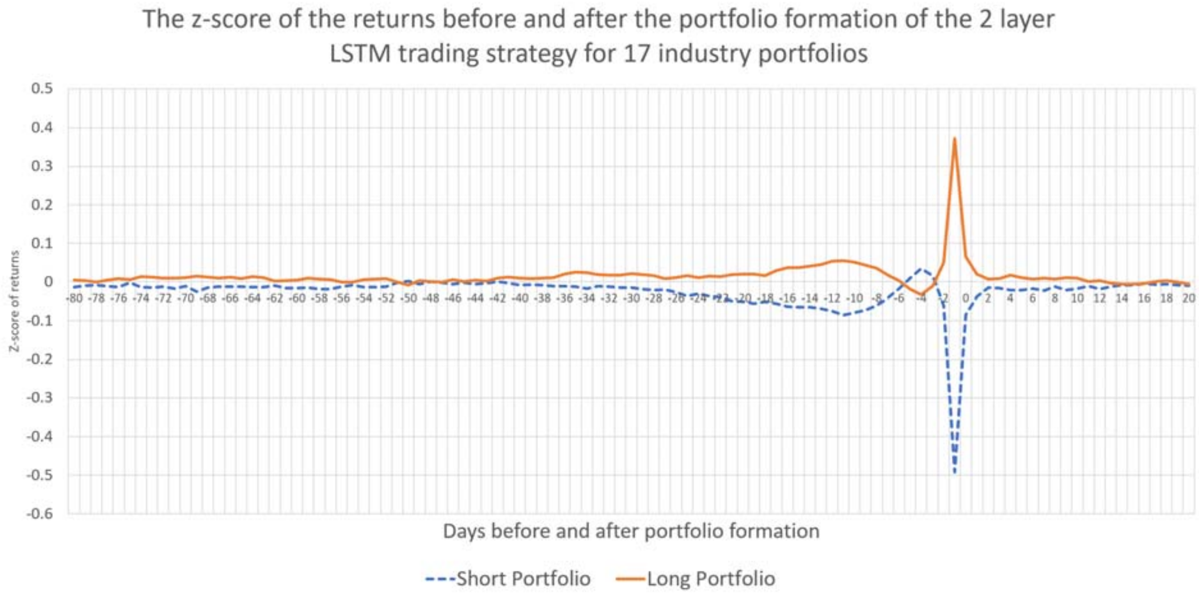


Table 7: The Before Trading Costs Returns And The Fama French 3-Factor α of the Trading Strategy Learned by 2-layer LSTM and MLP Hybrid Neural Network

This table reports the returns and the Fama French 3-factor α of the trading strategy learned by 2-layer LSTM and MLP hybrid neural network. Panel A and B report the returns, the Fama French 3-factor α and FF3 factor loading of equal-weighted portfolio. Panel C and D report the corresponding results of value-weighted portfolio. All results are before trading costs and cover the period from 1981 through 2017. We feed the past 12 month returns to a 2-layer LSTM network and output a 30-dimensional vector. We then concatenate the vector with 15 firm characteristics signal and feed the combined 45-dimensional vector to a 4-layer MLP neural network. According to the output of the MLP neural network, we long (short) the 4% of stocks with the highest probability to outperform (underperform) in the next month. We follow Jeehadesh and Titman (1993) for the definition of holding period. If the holding period is n months, we will rebalance 1/n of the portfolio every month and hold the 1/n stocks for n months. The Newey-West t-statistics are reported below each number.

Panel A: The monthly equal-weighted return and the Fama French 3-factor α of the 2-layer LSTM and MLP hybrid model. The sample covers from 1981 to 2017. The Newey-West t-statistics are reported below each return.

	Holding Periods					
Year	1-month holding period return	1-month holding period α	2-month holding period return	2-month holding period α	3-month holding period return	3-month holding period α
All years	2.49% (12.63)	2.57% (12.39)	2.05% (12.58)	2.09% (12.54)	1.69% (11.49)	1.74% (11.88)
1981-1990	2.96% (11.37)	3.12% (10.70)	2.29% (9.88)	2.39% (10.22)	1.78% (8.60)	1.92% (9.35)
1991-2000	3.08% (7.96)	3.10% (7.44)	2.38% (7.19)	2.41% (6.44)	2.08% (7.11)	2.16% (7.03)
2001-2010	2.56% (5.43)	2.64% (5.81)	2.37% (6.51)	2.36% (7.16)	2.10% (6.65)	2.07% (7.60)
2011-2017	0.90% (2.26)	0.66% (1.76)	0.74% (3.02)	0.68% (2.60)	0.40% (2.02)	0.41% (2.04)

Panel B: The Fama French 3-factor loadings of the equal-weighted return of the 2-layer LSTM and MLP hybrid model. The sample covers from 1981 to 2017. The Newey-West t-statistics are reported below each value.

Factors	Holding period 1 month	Holding period 2 months	Holding period 3 months
3-factor α	2.57% (12.39)	2.09% (12.54)	1.74% (11.88)
Beta of Mkt-Rf	-0.054 (-1.08)	-0.066 (-1.89)	-0.059 (-2.40)
Beta of SMB	-0.061 (-0.50)	0.036 (0.50)	0.016 (0.25)
Beta of HML	-0.123 (1.25)	-0.027 (-0.35)	-0.033 (-0.55)

Table 7 Continued

Panel C: The monthly value-weighted return and the Fama French 3-factor α of the 2-layer LSTM and MLP hybrid model. The sample covers from 1981 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods					
Year	1-month holding period return	1-month holding period α	2-month holding period return	2-month holding period α	3-month holding period return	3-month holding period α
All years	1.56% (5.00)	1.76% (5.52)	1.31% (5.13)	1.39% (5.19)	1.04% (4.88)	1.07% (4.88)
1981-1990	1.97% (5.14)	2.30% (5.99)	1.69% (5.66)	1.89% (6.46)	1.33% (4.76)	1.54% (5.72)
1991-2000	2.01% (2.98)	2.25% (3.18)	1.52% (2.88)	1.77% (3.04)	1.23% (2.79)	1.32% (2.97)
2001-2010	1.78% (2.42)	2.03% (3.05)	1.53% (2.46)	1.54% (2.67)	1.34% (2.72)	1.34% (2.70)
2011-2017	0.0007% (0.01)	-0.16% (-0.31)	0.16% (0.34)	0.11% (0.23)	-0.09% (-0.25)	-0.11% (-0.31)

Panel D: The Fama French 3-factor loadings of the value-weighted return of the 2-layer LSTM and MLP hybrid trading strategy. The sample is all samples from 1981 to 2016. The t-statistics are reported below each number.

Factors	Holding period 1 month	Holding period 2 months	Holding period 3 months
3-factor α	1.76% (5.52)	1.39% (5.19)	1.07% (4.88)
Beta of Mkt-Rf	-0.112 (-1.42)	-0.086 (-1.53)	-0.029 (-0.67)
Beta of SMB	-0.407 (-2.44)	-0.083 (-0.87)	-0.075 (-1.02)
Beta of HML	-0.28 (-1.82)	-0.025 (-0.20)	-0.013 (-0.15)

Table 8: The Before Trading Costs Returns And The Fama French 3-Factor α of the Trading Strategy Learned by 2-layer LSTM and MLP Hybrid Neural Network, With Time Series Inputs Skipping Month(t-1)'S Return (No Short-Term Reversal Effect)

This table reports the returns and Fama French 3-factor α without short-term reversal effect. The neural network stucture is same as Table 7. However, different from Table 7, when feeding the time series returns to the first LSTM layer, we skipping month(t-1)'s return and only feed returns from month(t-12) to month(t-2). All results are before the trading costs and cover the period from 1981 through 2017. We feed 11 months' returns to a 2-layer LSTM network and output a 30-dimensional vector. We then concatenate the vector with 15 firm characteristics signals and feed the combined 45-dimensional vector to a 4-layer MLP neural network. According to the output of the MLP neural network, we long (short) the 4% of stocks with the highest probability to outperform (underperform) in the next month. We follow Jeehadesh and Titman (1993) for the definition of holding period. If the holding period is n months, we will rebalance 1/n of the portfolio every month and hold the 1/n stocks for n months. The Newey-West t-statistics are reported below each value.

The monthly equal-weighted and value-weighted return and the Fama French 3-factor α of the 2-layer LSTM and MLP hybrid model. When feeding the time series returns to the first LSTM layer, we skip month(t-1)'s return. The sample covers from 1981 to 2017. The Newey-West t-statistics are reported below each return.

	Holding Periods					
Year	1-month holding period return	1-month holding period α	2-month holding period return	2-month holding period α	3-month holding period return	3-month holding period α
Equal-weighted return	1.87% (9.87)	1.92% (9.69)	1.70% (11.48)	1.72% (11.74)	1.52% (10.51)	1.54% (10.77)
Value-weighted return	1.39% (4.51)	1.51% (4.77)	1.17% (4.73)	1.21% (4.70)	1.13% (5.38)	1.14% (5.25)

Figure 10: The Z-Score Of Return And Volume Before And After The Portfolio Formation Of The Trading Strategy Learned by 2-Layer LSTM And MLP Hybrid Neural Network

(A) and (C) depict the average z-score of the returns before and after the portfolio formation month of the 2 layer LSTM and MLP hybrid model. (B) and (D) shows the log the z-score of the dollar volumes before and after the portfolio formation. In (A) and (B) we feed past 12 months return to the first LSTM layer, while in (C) and (D) we skip month(t-1) and only feed month(t-12) to month(t-2)'s return to the first LSTM layer. The z-score of return is the $(\text{return}_{it} - mean(return_t))/\text{stdev}(\text{return}_\text{t})$ for stock i in month t. The z-score of volume is normalized by (divided by) the volume in month(t-12), where t is the portfolio formation month. As a result the z-score of volume for month(t-12) is always 1 and its log value is always 0. The log of the z-score of the volume is shown on the y-axis of (B) and (D). We use log of volume's z-score because we want the increase and decrease of the volume to have same impact on the average. For example, if the volume increase by 100%, the log of z-score of the volume is log(2). If the volume decreased by 50%, the log of the z-score of the volume is log(0.5)=-log(2). The months before and after the portfolio formation month is on the x-axis. The sample covers the period from 1981 through 2017.

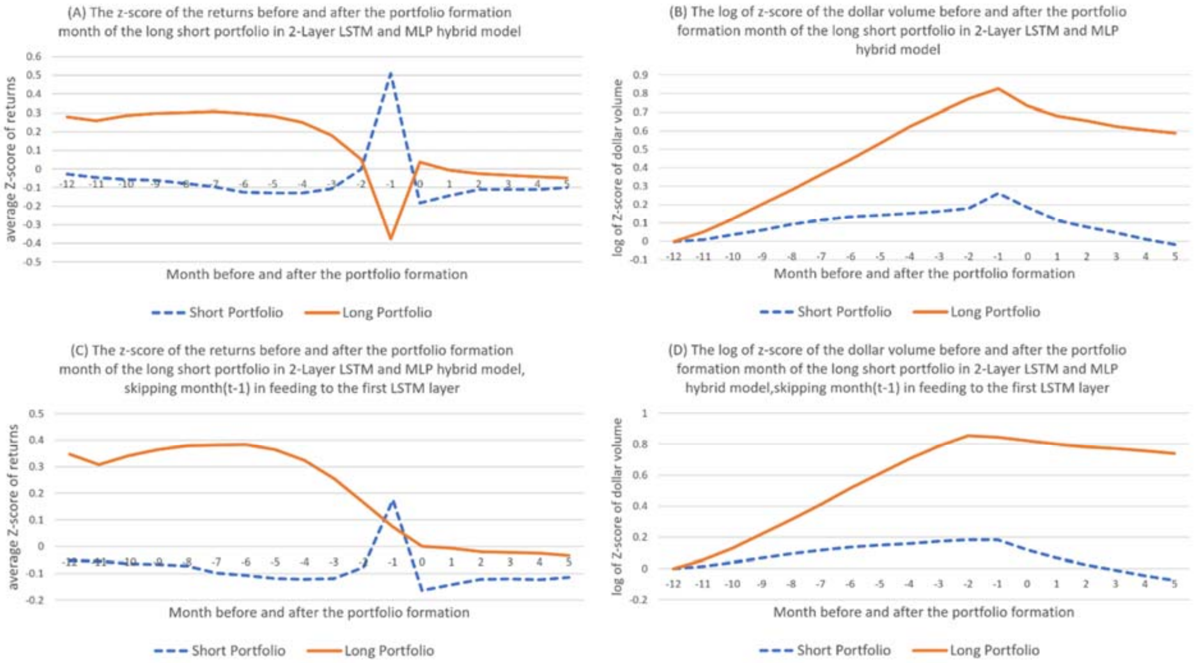


Table 9: The Before and After Trading Cost Return of the Trading Strategy Learned by the 2-layer LSTM and MLP Hybrid Neural Network

This table reports the monthly equal-weighted and value-weighted return before and after the trading cost of the trading strategy learned by the 2-layer LSTM and MLP hybrid neural network. Panel A reports the result of equal-weighted return. Panel B reports the result of value-weighted return. We estimate the trading cost from TAQ database and the detail of the procedure is described in section 5.1.2. Because the TAQ database start from 1993, the sample covers from 1993 to 2017. The before trading cost returns are slightly different from the ones in Table 7 because Table 7 covers from 1981 to 2017. We do not report FF-3 α in this table because Table 7 already shows that the before trading cost returns have low factor loading on the Fama French 3 factors. FF-3 factors are all before trading cost returns. Regressing an after trading cost return on before trading cost factors is not very meaningful. The Newey-West t-statistics are reported below each number.

Panel A: The monthly equal-weighted return before and after the trading cost of the 2-layer LSTM and MLP hybrid model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
The equal-weighted return of 2-Layer LSTM and MLP hybrid model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	2.37% (8.97)	2.02% (9.46)	1.73% (8.86)
After-trading-cost return	1.57% (6.03)	1.63% (7.80)	1.46% (7.70)

Panel B: The monthly value-weighted return before and after the trading cost of the 2-layer LSTM and MLP hybrid model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
The value-weighted return of 2-Layer LSTM and MLP hybrid model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	1.49% (3.54)	1.24% (4.27)	0.99% (4.32)
After-trading-cost return	1.19% (2.79)	1.04% (3.59)	0.86% (3.75)

Table 10: The Before and After Trading Cost Return of the Trading Strategy Learned by the MLP-only Neural Network

This table reports the monthly equal-weighted and value-weighted return before and after the trading cost of the trading strategy learned by the *MLP-only* neural network. Panel A reports the result of equal-weighted return. Panel B reports the result of value-weighted return. We estimate the trading cost from TAQ database and the detail of the procedure is described in section 5.1.2. Because the TAQ database start from 1993, the sample covers from 1993 to 2017. Other than the neural networks stucture, the settings of this test are same as the 2-layer LSTM and MLP hybrid model in Table 9.

Panel A: The monthly equal-weighted return before and after the trading cost of the MLP-only model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
The equal weighted return of the MLP-only model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	1.91% (8.67)	1.59% (8.36)	1.27% (7.14)
After-trading-cost return	0.49% (2.21)	0.87% (5.28)	0.79% (4.96)

Panel B: The monthly value-weighted return before and after the trading cost of the MLP-only model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
The value-weighted return of the MLP-only model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	1.47% (3.34)	1.22% (4.08)	0.94% (4.13)
After-trading-cost return	0.89% (2.15)	0.86% (3.27)	0.82% (3.15)

Table 11: The Before and After Trading Cost Return of The Momentum And Short-Term Reversal 5x5 Double Sort Trading Strategy

This table reports the monthly equal-weighted and value-weighted return before and after the trading cost of the momentum and short-term reversal 5x5 double sort trading strategy. In each month, we first sort the stocks into 5 momentum quintiles according to the sum of return from month(t-12) to month(t-2), where t is the portfolio formation month. We further sort each momentum quintile into 5 short-term reversal quintiles. We thus have a 5x5 portfolio. We long the short-term reversal loser in the momentum winner quintile and short the short-term reversal winner in the momentum loser quintile. The sample covers the period from 1993 to 2017. The Newey-West t-statistics are reported below each number.

Panel A, The equal weighted return of the momentum and short-term reversal 5x5 double sort trading strategy. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
The equal-weighted return of the 5x5 double sort strategy	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	2.08% (6.32)	1.50% (6.20)	1.03% (5.68)
After-trading-cost return	0.58% (1.76)	0.74% (3.27)	0.53% (2.53)

Panel B, The value weighted return of the momentum and short-term reversal 5x5 double sort trading strategy. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
The value-weighted return of the 5x5 double sort strategy	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	1.43% (3.60)	1.13% (3.90)	0.95% (3.93)
After-trading-cost return	0.92% (2.23)	0.85% (2.83)	0.76% (2.97)

Figure 11: Z-Score Of The Returns And Volumes of The Long And Short Portfolio Of The Momentum And Short-Term Reversal 5x5 Double Sort Trading Strategy

Figure (A) depicts the average z-score of the returns of the long-short portfolio of the momentum and short-term reversal 5x5 double sort trading strategy, before and after the portfolio formation. Figure (B) reports the log of volume before and after the portfolio formation. In Figure (A), the y-axis is the average z-score of returns. In (B), the y-axis is the log of the average of the z-scores of the volume. For both (A) and (B), the x-axis is the month before and after the portfolio formation. We use log in (B) because we want the increase and the decrease of the volume to have same impact on the averages. In each month, we first sort the stocks into 5 momentum quintiles according to the sum of returns from month(t-12) to month(t-2), where t is the portfolio formation month. We further sort each momentum quintile into 5 short-term reversal quintiles. We thus have a 5x5 portfolio. We long the short-term reversal loser in the momentum winner quintile and short the short-term reversal winner in the momentum loser quintile. The sample covers the period from 1993 to 2017.

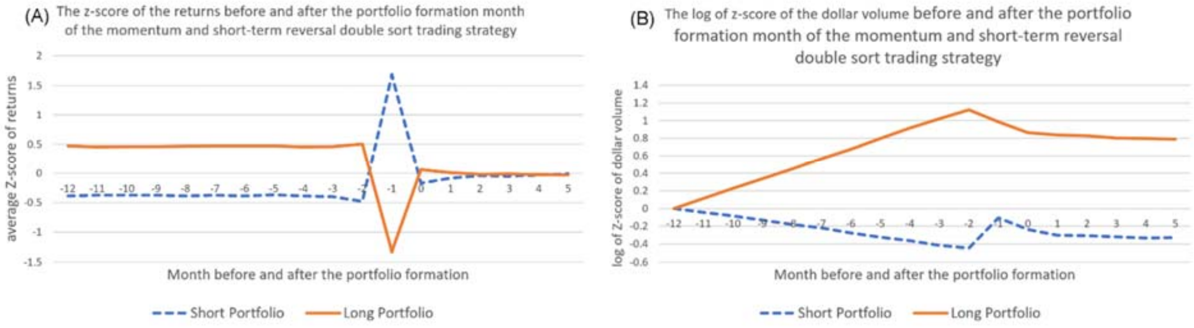


Table 12: The Before and After Trading Cost Return of The Random Forest, XGBoost and Ensemble Learning

This table reports the monthly equal-weighted return before and after the trading cost of the The Random Forest and XGBoost model. We then combine the LSTM_MLP hybrid model, Random Forest and XGBoost model together and build an ensemble learning model and report the monthly equal-weighted return of this model. Panel A is the result of Random forest model. Panel B is the result of XGBoost model. Panel C is the result of ensemble learning model. We estimate the trading cost from TAQ database and the detail of the procedure is described in section 5.1.2. The sample covers the period from 1993 to 2017. The Newey-West t-statistics are reported below each number.

Panel A, The equal-weighted return of the Random Forest model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
Return of the Random Forest model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	1.59% (7.64)	1.39% (8.24)	1.15% (7.49)
After-trading-cost return	0.15% (0.73)	0.68% (3.96)	0.67% (4.41)

Panel B, The equal-weighted return of the XGBoost model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

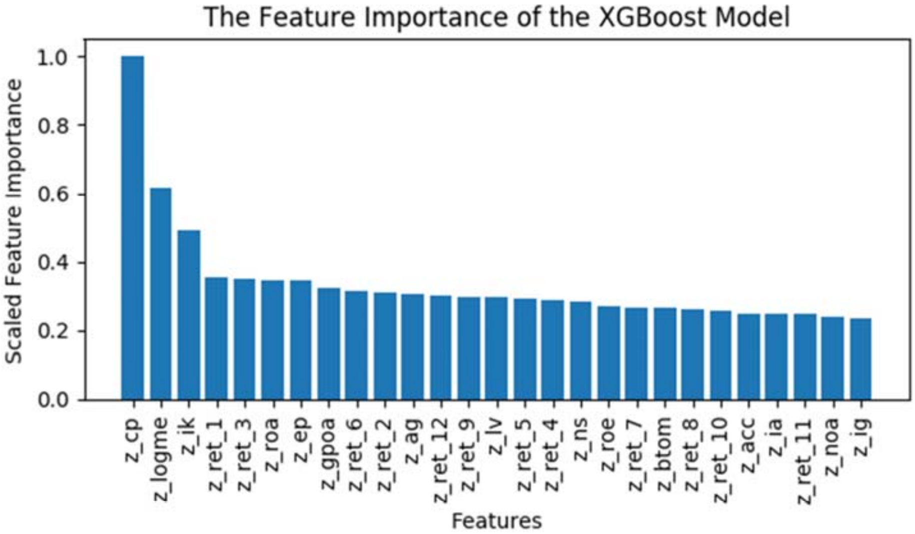
	Holding Periods		
Return of the XGBoost model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	2.36% (9.72)	1.72% (8.01)	1.34% (6.69)
After-trading-cost return	1.03% (4.24)	1.04% (5.02)	0.90% (4.54)

Panel C, the equal-weighted return of the ensemble learning model that combines the LSTM-MLP hybrid model, the random forest model and the XGBoost model. The sample covers from 1993 to 2017. The Newey-West t-statistics are reported below each number.

	Holding Periods		
Return of the ensemble learning model	1-month holding period return	2-month holding period return	3-month holding period return
Before-trading-cost return	2.53% (9.74)	2.02% (9.35)	1.71% (8.45)
After-trading-cost return	1.40% (5.40)	1.46% (6.87)	1.33% (6.69)

Figure 12: The Feature Importance of XGBoost and Random Forest model

Figure 12 depicts the feature importance of the XGBoost and Random Forest model. The sample covers 1993 to 2017. In each month, XGBoost and Random Forest model classify each stock into a return quintile. The feature that make the most decrease in the impurity in the tree node splits will have higher feature importance. In each month, the model give each feature a feature importance score. The results below are the average feature importance score across the 300 months from 2013 to 2017. The features are all z-scores. The meaning of the feature names can be found in section 4.2.



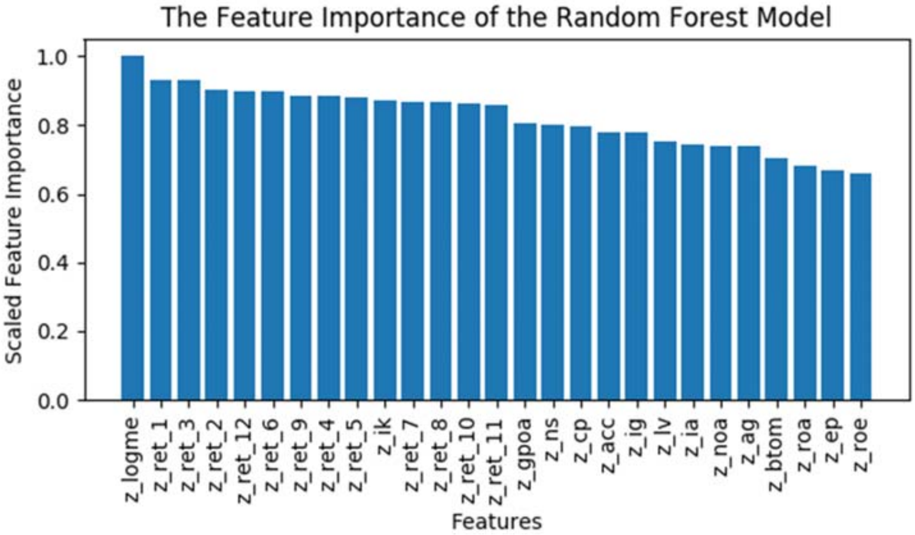


Table 13: Regression of The Returns Of LSTM-MLP Hybrid Model On The Returns of Other Models

This table reports the results of regression of returns of the 2-Layer LSTM and MLP hybrid model on the returns of other models. Each regression has 300 data points, which is the returns of the 300 months from 1993 through 2017. The returns of momentum strategy is the profits when we long the top 20% momentum winner and short the bottom 20% momentum loser. The returns of short-term reversal is the profits when we long the 20% of stocks that have lowest returns in month(t-1) and long the 20% of stocks that have the highest returns in month(t-1). The returns of LSTM-MLP hybrid, 5x5 double sort, MLP-only, random forest and XGBoost are from the trading strategy we describe in section 5.3. The t-statistics are reported below each slope.

The result of regression of returns of the LSTM-MLP Hybrid Model on the returns of other models. The sample covers the 300 months from 1993 through 2017.

	Dependent Variables							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Independent Variables	Double Sort	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid
Double sort		0.210 (4.70)			-0.062 (-1.10)			
Momentum	0.914 (39.89)		0.392 (8.62)		0.439 (7.05)			
Short-term reversal	0.899 (31.65)			-0.242 (-3.94)				
MLP-only						0.670 (12.06)		
Random forest							0.665 (10.61)	
XGBoost								0.610 (11.67)
R^2	87.76%	6.91%	19.95%	4.94%	20.27%	32.81%	27.43%	31.35%

Table 13 continued

The result of regression of returns of the LSTM-MLP Hybrid Model on the returns of other models. The sample covers the 300 months from 1993 through 2017.

	Dependent Variables					
	(9)	(10)	(11)	(12)	(13)	(14)
Independent Variables	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid	LSTM_MLP hybrid
Double sort						-0.056 (-1.21)
Momentum					0.194 (4.69)	0.236 (4.37)
Short-term reversal						
MLP-only	0.483 (7.90)		0.430 (6.43)	0.378 (5.66)	0.350 (5.40)	0.352 (5.43)
Random forest	0.397 (5.98)	0.362 (4.80)		0.277 (3.78)	0.263 (3.71)	0.255 (3.59)
XGBoost		0.415 (6.43)	0.364 (5.84)	0.245 (3.57)	0.168 (2.46)	0.172 (2.52)
R^2	40.03%	36.29%	39.73%	42.50%	46.50%	47.76%