This version: December 12, 2013

Applying Deep Learning to Enhance Momentum Trading Strategies in Stocks

Lawrence Takeuchi *
Yu-Ying (Albert) Lee

LTAKEUCH@STANFORD.EDU
YY.ALBERT.LEE@GMAIL.COM

Abstract

We use an autoencoder composed of stacked restricted Boltzmann machines to extract features from the history of individual stock prices. Our model is able to discover an enhanced version of the momentum effect in stocks without extensive hand-engineering of input features and deliver an annualized return of 45.93% over the 1990-2009 test period versus 10.53% for basic momentum.

1. Introduction

Price momentum is the empirical finding (Jegadeesh & Titman, 1993) that stocks with high past returns over 3-to-12 months (winners) continue to perform well over the next few months relative to stocks with low past returns (losers). Subsequent studies find that this momentum effect continues to remain robust in the US after becoming widely known and applies to international stocks as well as other asset classes including foreign exchange, commodities, and bonds (Asness et al., 2013). For finance academics the fact that a simple strategy of buying winners and selling losers can apparently be profitable challenges the notion that markets quickly incorporate available information into asset prices. Indeed, Fama and French (2008) describe momentum as the "premier anomaly" in stock returns.

The momentum trading strategy, along with its many refinements, is largely the product of a vast, ongoing effort by finance academics and practitioners to hand-engineer features from historical stock prices. Recent advances in deep learning hold the promise of allowing machine learning algorithms to extract discriminative information from data without such labor-intensive feature engineering and have been successfully applied to fields such as speech recognition, image recognition, and natural language processing (Bengio et al., 2012).

In this paper we examine whether deep learning techniques can discover features in the time series of stock prices that can successfully predict future returns. The objective is a challenging one. While most research in deep learning considers tasks that are easy for humans to accomplish, predicting stock returns using publicly available information is notoriously difficult even for professional investors given the high level of noise in stock price movements. Furthermore, any patterns that exist are subject to change as investors themselves learn over time and compete for trading profits.

Considering the pervasive use of historical price charts by investors and noting that the primary mode of analysis is visual, we take an approach similar to that used by Hinton and Salakhutdinov (2006) to classify handwritten digits. In particular, we use an autoencoder composed of stacked restricted Boltzmann machines (RBMs) to extract features from stock prices, which we then pass to a feedforward neural network (FFNN) classifier.

2. Data

We obtain data on individual US stocks from the Center for Research in Security Prices.

2.1. Sample selection

We restrict our analysis to ordinary shares trading on NYSE, AMEX, or Nasdaq. To mitigate the impact of any market microstructure-related noise, we exclude stocks with monthly closing prices below $5 per share at the time of portfolio formation. This step also reduces the number of examples with extreme returns.

The training set covers the period from January 1965 to December 1989, which coincides with the period examined by Jegadeesh and Titman (1993), and contains 848,000 stock/month examples. The test set covers the period from January 1990 to December 2009 and contains 924,300 stock/month examples. On average

*Corresponding author.

there are 3,282 stocks in the sample each month.

2.2. Input variables and preprocessing

We want to provide our model with information that would be available from the historical price chart for each stock and let it extract useful features without the need for extensive feature engineering. For every month t, we use the 12 monthly returns for month $t-2$ through $t-13$ and the 20 daily returns approximately corresponding to month t.[1] We also use an indicator variable if the holding period, month $t+1$, falls in January.[2] Thus we have a total of 33 input variables for each stock/month example.

Next we compute a series of 12 cumulative returns using the monthly returns and 20 cumulative returns using the the daily returns. We note that price momentum is a cross-sectional phenomenon with winners having high past returns and losers having low past returns relative to other stocks. Thus we normalize each of the cumulative returns by calculating the z-score relative to the cross-section of all stocks for each month or day. Figure 1 illustrates preprocessing pipeline using the 12 monthly returns for one example in the test set.

Finally we use returns over the subsequent month, $t+1$, to label the examples with returns below the median as belonging to class 1 and those with returns above the median to class 2.

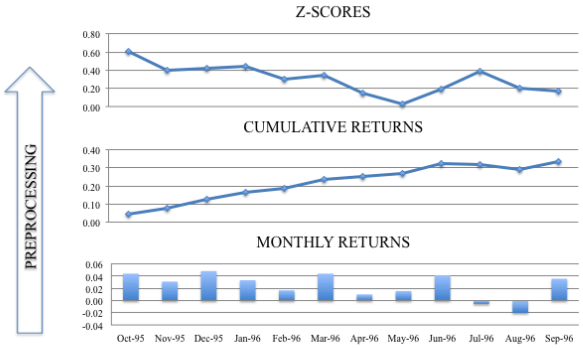

Figure 1. Preprocessing of inputs

[1]We choose these inputs given the casual observation that investors tend to use higher frequency prices when viewing charts covering shorter windows of time.

[2]Turn-of-the-year patterns in stock returns were well-known prior to Jegadeesh and Titman (1993). In any case, using dummy variables for each calendar month gives similar results.

3. Deep Learning Model

3.1. Model and learning algorithm

We follow an approach similar to that introduced by Hinton and Salakhutdinov (2006) to train networks with multiple hidden layers. Our model consists of a stack of RBMs, which unlike full Boltzmann machines have no intra-layer connections. Each RBM consists of one layer of visible units (the inputs) and one layer of hidden units connected by symmetric links. The output of each RBM serves as the input to the next RBM in the stack.

We train the encoder network (see Figure 2) layer by layer in a pretraining step. Following Hinton (2010), we split the dataset into smaller, non-overlapping mini-batches. The RBMs in the encoder are then unrolled to form an encoder-decoder, which is fine tuned using backpropagation. In our implementation, the number of hidden units in the final layer of the encoder is sharply reduced, which forces a reduction in dimensionality. As described below, however, the size of this bottleneck layer is an outcome of the optimization procedure we use to specify the network dimensions rather than an explicit design choice.[3]

At this stage, the encoder outputs a low dimensional representation of the inputs. The intention is that it retains interesting features from the historical stock chart that are useful for forecasting returns, but eliminates irrelevant noise. We use the weights estimated thus far to initialize the corresponding weights in the full network, which is composed of the encoder and a FFNN classifier. The final step is to train the entire network using the labeled examples via backpropagation.[4]

3.2. Network specification

We use hold-out cross validation to determine the number of layers and number of units per layer in our network. In particular, we further divide the training set into two subsets covering 1965-1982 and 1983-1989, respectively. Each model specification is trained on the first subset and then tested on the second. We choose this approach over k-fold cross validation since we have a sufficiently large dataset and more importantly want to avoid the look-ahead bias that could arise from training the model with data not available at a given historical date.

[3]Hinton and Salakhutdinov (2006), by contrast, omit an explicit bottleneck layer in their digit classification example and use a 784-500-500-2000-10 network.

[4]See Ng et al. (2013) for a tutorial on training deep networks.

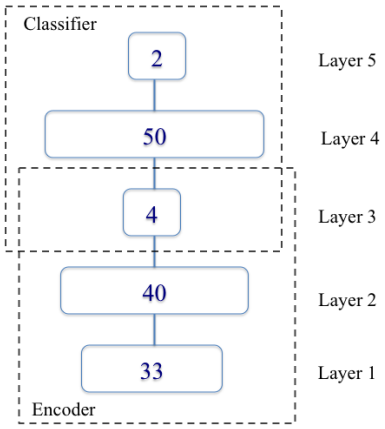

Figure 2. Network Architecture

To keep the task manageable, we fix the number of units in the penultimate hidden layer at 50.[5] Thus, our first set of candidate specifications is $33-s_2-50-2$ and the second is $33-s_2-s_3-50-2$ where s_l denotes the number of units in layer l. A grid search over the number of units using our hold-out cross validation scheme finds that $s_2 = 40$ and $s_3 = 4$ give the lowest classification error. The next set of candidate specifications is $33-s_2-s_3-s_4-50-2$. Given the large number of dimensions, we use $s_2 = 40$ and $s_4 = 4$ and only search over s_3. We find that adding another layer does not reduce the classification error.[6] Thus our final specification is the five-layer network $33-40-4-50-2$ consisting of an encoder that takes 33 inputs and reduces them to a 4 dimensional code and a classifier that takes these 4 inputs and outputs the probabilities for the two classes. While the approach taken is admittedly heuristic, we believe that it provides a disciplined method to specify a base case model.

4. Results

We train our model using the examples from 1965-1989 and test it using those from 1990-2009. In this section we keep both the network configuration and weights fixed, but discuss later how these could be updated over time.

[5]However, once we determine the specification of the stacked autoencoders we verify that this is a reasonable choice.

[6]Our ongoing work suggests that additional layers may be useful when the number of features are increased.

Table 1. Confusion Matrix

		Predicted	
		1	2
Actual	1	22.38%	27.45%
	2	19.19%	30.97%

4.1. Classification performance

Table 1 summarizes the results in a confusion matrix with the entries scaled by the total number of test examples. The model achieves an overall accuracy rate of 53.36%. The model is correct 53.84% of the time when it predicts class 1 and a somewhat lower 53.01% of the time when it predicts class 2.

These probabilities alone, however, do not provide a complete picture of the model's performance since investors actually care more about returns in their objective function. In particular, we would like to compare the realized returns for the stocks predicted to be in class 1 against those for stocks predicted to be in class 2. Pooling all months in the test set, the average one-month holding period return is 0.11% for stocks predicted to be in class 1 and 1.50% for those predicted to be in class 2, or a difference of 1.39%. Alternatively, we can use past 12 month returns (from month $t-13$ to $t-2$) and predict that examples with past returns below the median will be in class 1 and those above the median in class 2. This basic momentum strategy produces an average holding period return of 0.64% for stocks in class 1 and 1.20% for those in class 2, or a difference of 0.55%.

4.2. Information content of class probabilities

While these results appear promising, we have used only a portion of the information produced by the model. Figure 3 shows the relation between the estimated probability of being in class 2 according to the model versus the holding period return that is actually realized for the test set, where we use a Gaussian kernel regression to smooth the curve. We see an increasing relation indicating that a higher class 2 probability leads to higher realized returns on average. Since there is no requirement to hold every stock, an investor could clearly do better if he bought and sold stocks in the tails of the distribution rather than using the 50% threshold to form long and short portfolios.

We rank all stocks each month by their class 2 probabilities and buy those in the top decile and sell those in the bottom decile. Next month we close out these positions and form new long and short portfolios. Repeat-

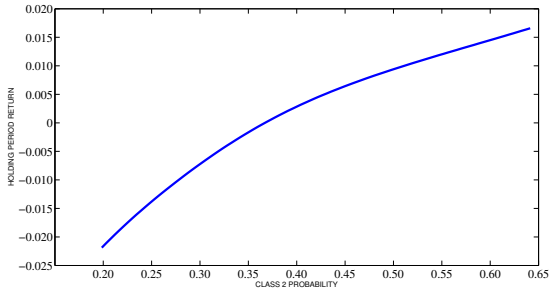

Figure 3. Holding period returns by class 2 probability

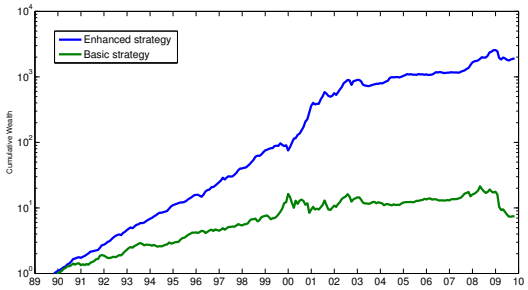

Figure 4. Log Growth in Cumulative Wealth

Table 2. Average monthly momentum returns, 1990-2009.

STRATEGY	DECILE 1	DECILE 10	10 - 1
ENHANCED	-1.02%	2.33%	3.35%
T-STATISTIC	-2.05	7.75	9.26
BASIC	0.25%	1.35%	1.10%
T-STATISTIC	0.47	2.76	2.39

ing this process over the test set generates a monthly time series of investment returns. For comparison we also perform this analysis using past 12 month returns to form deciles. Table 2 presents the average returns for these two strategies, which we call enhanced and basic momentum, respectively. The enhanced strategy generates an average monthly return of 3.35% with a t-statistic of 9.26. In contrast, the basic strategy produces a more modest, but still statistically significant, 1.10% per month.[7]

On an annualized basis, the returns from our model are a very impressive 45.93% versus 10.53% for basic momentum. Figure 4 shows the growth, on a logarithmic scale, of $1 invested in each of the enhanced and basic strategies from 1990 to 2009. Ending wealth, before considering implementation costs, is $7.41 for the basic strategy and $1,919, or 259 times higher, for the enhanced strategy.

4.3. Source of investment returns

We now show that the enhanced strategy is, in fact, a momentum strategy in the sense that the decile 10 stocks have higher past 12 month returns than the decile 1 stocks. Table 3 shows the past 12 month (month $t-13$ to $t-2$) and past 20 day (month t) returns expressed as z-scores for the enhanced and basic strategies. We focus on these two features since they are highlighted in the finance literature (Jegadeesh & Titman, 1993) as being potential predictors of stock returns. By construction, the basic momentum strategy takes very extreme positions based on past 12 month returns with a $10-1$ spread in this feature of more than 3 standard deviations. The enhanced strategy has a $10-1$ spread of 0.62 showing that it is also making a bet based on past 12 month returns, but in a less extreme way than basic strategy. The lower half of the table shows that the enhanced strategy buys stocks that have had poor recent returns and sells those with high recent returns, consistent with the short-term reversal effect (Jegadeesh, 1990). The basic momentum strategy, however, has similar past 20 day returns in deciles 1 and 10.

While we highlight these two features, we emphasize that the model is likely picking up other more subtle patterns in the historical price chart that are less intuitive to interpret. Table 3 shows that the enhanced strategy does not take overly extreme positions based on past 12 month and past 20 day returns as would be the case if we double sort on these features to form portfolios as is common practice in finance studies. We regard this finding, together with the size of the investment returns, as encouraging for deep learning as it suggests that our model is not merely rediscovering known patterns in stock prices, but going beyond what humans have been able to achieve.

5. Discussion and Further Work

This study represents one of the first, as far as we are aware, applications of deep learning to stock trading and makes two main contributions to the applied machine learning literature. First, we show that stacked autoencoders constructed from RBMs can extract useful features even from low signal-to-noise time series

[7] These returns are calculated as the arithmetic mean of the time series of monthly returns.

Table 3. Stock characteristics by strategy.

	Decile 1	Decile 10	10 - 1
Past 12m ret			
- Enhanced	-0.39	0.23	0.62
- Basic	-1.05	2.03	3.08
Past 20d ret			
- Enhanced	0.41	-0.51	-0.92
- Basic	-0.06	0.05	0.11

data such as financial asset prices if the inputs are appropriately preprocessed. Second, we illustrate the potential for deep learning to reduce the need for extensive feature engineering in an application area (financial markets) that has long been of interest to machine learning researchers. Our model easily accommodates returns of different frequencies as well as non-return data and produces investment results that exceed those of most strategies in the vast finance literature on momentum strategies.

In ongoing work, we are considering additional features such as industry and aggregate market returns as well as non-return data such as firm characteristics and macroeconomic indicators. An open question as we expand the number of inputs is whether separate autoencoders for various categories of features would perform better than combining all features in a single autoencoder.

We are also examining the impact of updating the weights in our network over time. Our current methodology, in which we train the model once and then hold parameters fixed for the entire test period, is unlikely to be optimal as investor behavior as well as the institutional framework of the market change over time. Furthermore, even if the data were stationary, we could improve performance by training with all the available data at each date. An important implementation issue is computational cost, especially as the number of features and depth of the network increase. We are exploring a parallel implementation of the learning algorithm that could be run on GPUs. This approach should lead to a substantial decrease in training time as the algorithm can take advantage of parallelization at the data-level (since it uses mini-batches) as well as at the network layer level. Alternatively, a more straightforward approach would be to retrain the classifier each month, but update the autoencoder less frequently in order to limit computational costs.

References

Asness, Clifford, Moskowitz, Tobias, and Pedersen, Lasse. Value and momentum everywhere. *Journal of Finance*, 68:929–985, 2013.

Bengio, Yoshua, Courville, Aaron, and Vincent, Pascal. Representation learning: a review and new perspectives. *ArXiv e-prints*, 2012.

Fama, Eugene and French, Kenneth. Dissecting anomalies. *Journal of Finance*, 63:1653–1678, 2008.

Hinton, Geoffrey. A practical guide to training restricted Boltzmann machines. Technical Report MTML TR 2010-003, University of Toronto, 2010.

Hinton, Geoffrey and Salakhutdinov, Ruslan. Reducing the dimensionality of data with neural networks. *Science*, 313:504–507, 2006.

Jegadeesh, Narasimhan. Evidence of predictable behavior of security returns. *Journal of Finance*, 45: 881–898, 1990.

Jegadeesh, Narasimhan and Titman, Sheridan. Returns to buying winners and selling losers: implications for stock market efficiency. *Journal of Finance*, 48:65–91, 1993.

Ng, Andrew, Ngiam, Jiquan, Foo, Chuan Yu, Mai, Yifan, and Suen, Caroline. UFLDL Tutorial, 2013. URL http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial.