Predicting Stock Returns Using Neural Networks

Murat Aydogdu
Rhode Island College

Predicting Stock Returns Using Neural Networks

April 5, 2018

Murat Aydogdu

Department of Economics and Finance
Rhode Island College
Alger Hall 237
Providence, RI 02908
maydogdu@ric.edu

Abstract

A single hidden layer neural network can be trained to predict whether a stock will be in the top, middle, or bottom third of sample stocks based on its return over the next month based on return, trading volume, and volatility measures available at the end of this month. In my preliminary work using S&P 500 stocks, the network has limited success in predicting which stocks are likely to go up but the prediction strength is not strong enough to help build profitable portfolios. While neural networks have pushed artifical intelligence forward in many fields, and while the investment industry has been shifting more towards quantitative prediction using neural networks and other machine learning models, their place in empirical finance research has been limited. My work aims to contribute to this growing literature.

All the programs used in this project are on github: https://github.com/MAydogdu/StockPriceDirectionPrediction_NeuralNetworks

Can we train a neural network to predict whether the price of a security will go up or down over the next month by using only publicly available price, volatility, and trading volume data? Preliminary results from my analysis show that a network can predict the direction of stock prices slightly better than random guessing but this improvement in oerformance does not result in profitable portfolios. More specifically, the neural network I trained in this project can predict which stocks are likely to be in the top third and middle third tertile based on their next month returns somewhat better than the two benchmarks I use. Neural networks typically perform better with larger datasets. This initial work suggests that the network performance can possibly improve with a larger dataset especially if other data (for instance fundamental data, market microstructure data, and data on less liquid securities) get added to the dataset.

The interest in neural networks has not quite been widespread in empirical finance research. Conversely, there has been a growing industry interest in machine learning models in general and neural networks as one of those models, partly due to neural networks' vast potential and recent success in artifical intelligence, in fields like image and speech recognition. As computing power gets cheaper and neural networks become commoditized, their use in finance research is likely to increase. My goal is to contribute to the pool of papers that establish a link between empirical finance and neural networks.

1 Background

This paper relates to two lines of empirical analysis: predicting short-term stock returns and using machine learning models in finance. Momentum and reversal in stock returns are widely studied, resulting in a large literature. I will provide only a cursory review here. While machine learning in finance is not new, it is not widespread either. Also, there seems to be some divergence in interest: while there's more of a push towards quantitative models in the investment industry, there doesn't seem to be as much of an increase on the research side. I will review sevaral papers below that use machine learning in empirical finance.

Studies analyzing one-month price movements often start with Jagadeesh (1990), who finds reversals at the one month frequency, as well as momentum at the one year frequency. This paper looks at the performance of the extreme portfolios (top decile, or winners, and bottom decile, or losers) when stocks are sorted into ten decile portfolios based on their one month risk-adjusted returns. Long-short portfolios that buy losers and sell losers earn a statistically significant 1.99% abnormal return per month; yearly long short portfolios (buy winners, sell losers) earn 0.93% per month. These results establish reversals in monthly returns and momentum in yearly returns.

This paper spawned a sizeable literature that sometimes confirm this finding and delve deeper, like Avramov et al (2004)who find that monthly return reversals are mostly due to loser stocks, Da et al (2014) who find stronger reversals for price movements that are not explained by stock fundamentals, and Simpson et al (2016) who find industry effects in reversals. But a recent paper by Hou et al (2017) cast a strong doubt on short term reversal effect and also, to a lesser extent, the one-year price momentum. They report "The high-minus-low short-term reversal (Srev) decile earns on average only −0.26% per month (t = −1.31)" when decile portfolios are formed using NYSE-listed stocks. They find the statistical significances found in a majority of anomaly literature for hundreds of anomalies are mostly driven by microcaps.

My work is similar in some ways to Liew et al (2017). They use daily price and trading volume data from yahoo over five years from the beginning of 2011 to the end of 2016 to predict returns of 10 very active ETFs that track different asset classes. Liew et al use three machine learning models: neural networks, support vector machines, and random forests in their work. Their empirical analysis strategy is to use returns and trading volume as well as lagged values of these variables to predict future returns, from 1 to 250 days. They find that these models can predict future returns, especially for the 20-40 day range. All models appear to work well, with neural networks lagging a bit behind in some cases.

Along the lines of forecasting future returns using neural networks, Lin et al (2009)find that a neural network can be trained to give buy / hold / sell signals. It can then be used to implement trading strategies that outperform buy and hold strategies. They use daily returns on the Taiwan Weighted Index and the S&P 500 Index. Their best-performing network yields trading strategies (adjusted for transaction costs and capital gains taxes) yield 10.21% annualized return, compared to 7.46% from the buy and hold strategy.

Messmer (2017) uses a deep network to construct long-short portfolios based on firm characteristics. He finds significant, risk-adjusted returns to these portfolios and notes the 1-month return and the 12-month return to be the most significant determinants of portfolio returns.

Perhaps one reason for limited research that uses machine learning in finance research is in explanation.[1] The linear / logistic regression framework provides a direct, clear relationship between a set of explanatory variables and an output (target) variable. The effects of each variable can be isolated, measured, and compared to the other variables. When the goal is eplaining a phenomenon, like a market anomaly, this approach works well. But the disadvantage of this framework is that it limits the research to situations where there is a reasonably linear relationship between the input (as is or after some transformation) and output variables. This may result in poor model performance in situations where the actual relationship is more complex. Predictions based on machine learning expand on the available models by expanding into many "non-parametric" models like random forests, support vector machines, and neural networks. While these models may perform better than, say logistic regression, when predicting an outcome, the direct link between the inputs and the ouputs gets lost due to the way these models work. The fundamental tradeoff is between model performance and model tractability.

2 Analysis framework and data

For all my analysis work in this paper, I provide a technical appendix at the end of this paper that provides all the details of my work. Also, the programs I wrote for this project are on github. I use 409 large cap stocks that are in the Standard and Poor's 500 Index (SP500) that have full data for the period 1/1/2000 - 1/31/2017. I use daily data to compute monthly data from adjusted closing prices and number of shares traded. Once I construct the dataset as detailed in the appendix, I use tensorflow to train a neural network to predict the direction of a security's price. More specifically, using return and volume data available this month, I want to predict whether a stock will be in the top, middle, or bottom return tertile next month (Up / Mid / Down, or 2 / 1 / 0). I then analyze the prediction performance of the neural network and compare the network to two benchmarks.

2.1 Neural networks

A neural network is a predictive model that accepts a set of inputs ("features") and computes an output as a prediction. There are other models used in predictive analysis: logistic regression, decision tree and its variants (like random forests and boosted trees), support vector machines, k-nearest neighbors and so forth. In this paper, I use neural networks for a few reasons. First, this model generally works well for a variety of prediction problems and has done well in my initial, exploratory work of comparing models with this dataset. Second, neural networks tend to perform better than other models when datasets expand and network models used get more complex as the data scales up. This is one reason why a lot of the improvements in the state of the art artifical intelligence are driven by neural networks. More data plus cheaper computing power seem to give neural networks some edge in some prediction problems.

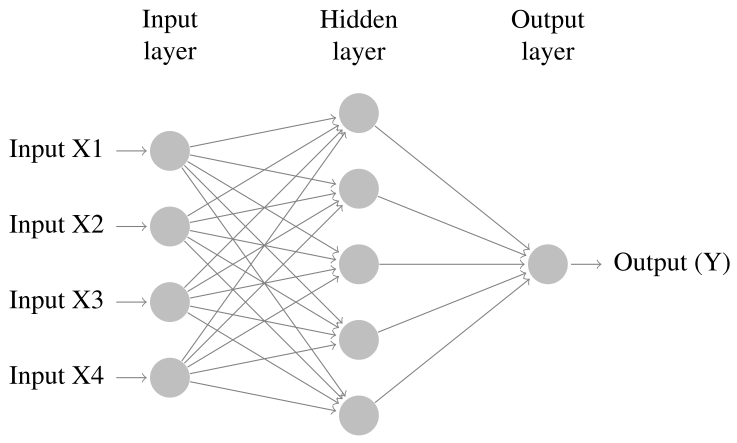
2.1.1 Building a neural network

A leading textbook on neural networks is Deep Leaning by Goodfellow et al (2016). I will only briefly mention the steps involved from problem formulation to getting results here, leaving implementation details for the appendix. A neural

[1] As an anectodal evidence, searching the word "neural" in FMA's 2017 program yields no finds.

Figure 1: A Neural Network

This graph shows a typical neural network. It has four inputs (features) that are fed into the hidden layer which has five nodes. The hidden layer is then fed into the output layer that has one node. In this paper, the output is whether the return will be positive or negative, and the network computes the probability of an uptick (positive price move) in its output layer. This is a feedforward network since outputs from a layers are not fed back to the previous layers, thus creating a cycle. It has one hidden layer and an ouput layer, so it is a two-layer network. (The input layer isn't counted.)



network is essentially a set of nodes organized in layers as shown in Figure 1. The shape of the network is determined by its inputs (the input layer) and outputs (the output layer). Additional to the input layer and the output layer, a typical network has one or more hidden layers. Those networks with more than one hidden layer are often referred to as deep networks. Inputs flow through the network towards the outputs. If a network's layers only have inputs from the preceding layer (so that no output from one layer is fed back into the previous layer, thus forming a cycle), it is called a feedforward network.[2] While conventions vary, the input layer is typically not counted; a network with an input layer and an output layer is a two-layer network.

Figure 1 shows a two-layer, feedforward network. The input layer has 4 variables (features) and the hidden layer has 5 nodes. The number of nodes in the input layer is simply the number of inputs. The number of hidden layers and the number of nodes in each hidden layer are paramaters that need to be determined. Once the set of inputs are determined, the prediction question is translated into an output (What is this person's salary? / Is this a cat photo? / Will the stock's price go up, stay in the middle range, or go down?), and the hidden layers and associated nodes are established, the shape of the network is complete.

Nodes in the hidden and output layers have weight vectors (w) and "bias" parameter vectors (b) associated with them as well activation functions. The nodes accept inputs from all the nodes in the preceding layer, multiply the inputs by weights, then add the bias ($A = WX + b$). The resulting vector is then passed through an activation function, like the hyperbolic tangent ($Z = tanh(A)$) or the rectifier. The output vector is then fed into the next layer. All of this is carried out as matrix computation where the inputs are propogated through the network. In a multi-class classification problem like the one this paper deals with, the sigmoid function can be used to compute the respective probabilities of outcomes. The outcome with the highest probability is the predicted outcome.

[2]Convolutional neural networks also have data moving unidirectionally from inputs to outputs, but they are somewhat more complicated in the way they manipulate data through layers.

2.1.2 Training the neural network

The main task in building a network then is to come up with a reasonable network structure and decide on the weights, biases, and activation functions. This phase is known as *training* the network. This is an iterative, potentially time consuming process. The typical strategy involves first coming up with a paramater space or a set (or a range) of reasonable values for hyperparameters. For instance, the number of hidden layers is a hyperparamater; and two hidden layers could be considered as the parameter space. There are other hyperparameters that impact the eventual performance of the network that are detailed in the appendix.

Then, for a chosen set of hyperparameters, and using a subset of data (training set), the network's weights and biases are determined through an iterative optimization algorithm. The most commonly used optimization algorithm now is the gradient descent and its variants. Starting with initial values (and there are various strategies to pursue to get started), gradient descent finds the optimal weights and biases by minimizing a cost function iteratively. At each iteration, gradient descent algorithm first propogates the inputs through the network using the current weights, biases, and activations and computes the value of a cost function. It then takes partial derivatives of the cost with respect to weights and biases ("gradients") and subtracts these from the current values of weights and biases, a process known as backpropogation. This readjustment of weights and biases continue until a cutoff benchmark is met, for instance, the cost stops falling by a meaningful amount and levels off. At this point, the network is fully defined.

Once the network is fully defined, its performance is then evaluated on new, unseen data (a test dataset or a holdout sample). The goal is to build a network that learns as much as possible from the training set (so that it has "low bias") while at the same time performs reasonably good predictions on new, unseen data (so that it has "low variance"). Essentially, we want the network to learn as much as possible from training data so that its predictions are as accurate as possible, but not memorize the training data so that it performs accurate predictions when it faces a new dataset.

The parameter space can then be investigated either thoroughly (grid search, which considers all combinations of the model parameters) or by random search, which selects a value for each parameter randomly and evaluates model performance using the selected paramater values, and iterates this process a large number of times. While the grid search is exhaustive, it is costly as the parameter space can explode as the number of parameters increase. Random search works well and can be performed much faster.

I use python and tensorflow to build a neural network. Developed by Google, tensorflow is a set of libraries that provide support for building a neural network. It allows for great control over most aspects of the network. The network I use in my analysis has a single hidden layer with 104 nodes and uses leaky rectifier as the activation function (i.e., it uses leaky rectified linear units, or ReLUs). It has 91 features that form the input layer and predicts whether a stock will be in the top, middle, or bottom tertile with respect to its return over the network, thus has three output nodes. The results I report in this paper are based on test set predictions of this network.

2.2 Data set

The details of data acquisition and dataset construction are in the technical appendix. In this project, I use publicly available daily stock data (adjusted closing price and number of shares traded) as the starting point. For data source, I use quandl. Starting with the Standard & Poor's 500 index constituents, I use 409 stocks that had data for the time frame 1/1/2000 - 1/31/2018. From daily data, I compute three "monthly" variables: return, average daily dollar trading volume, and the standard deviation of daily returns (volatility) over the month.

I use a 20-trading day frequency in my work as opposed to calendar months. The 20-trading day window roughly translates to monthly frequency. This makes it easy to compare my results to those papers that study monthly return predictability. Most of those studies use calendar months, though. That regularity has advantages of its own but it also introduces calendar effects that may confound results. Some of these papers then attempt to remove calendar effects by skipping days, for instance.

2.3 Features and target variable

Machine learning terminology can be different from empirical finance terminology at times but the concepts are often identical. I want to predict the direction of a stock's price (or its return) over the next month ($t+1$) using a dataset that's available at the end of this month (t). The variables I will use in prediction constitute the feature set.

I standardize (or scale) the returns and average volumes in two ways to construct the feature set. First, using the last 12 months (including this month), I compute time-series standardized returns by subtracting the mean return and dividing by the standard deviation of returns. Second, using returns of other stocks this month, I standardize returns cross-sectionally in a similar way. I perform the same calculations for volume and volatility measures. Finally, I use 12 lags of each variable as features. This process gives six sets of features (78 features in total) for each ticker-month:

- RT, RT01, ..., RT12: time-series scaled monthly return and its 12 lags
- AVT, AV01, ..., AVT12: time-series scaled average daily dollar volume and its 12 lags
- SDT, SD01, ..., SD12: time-series scaled standard deviation of daily returns and its 12 lags
- RC, RC01, ..., RC12: cross-sectionally scaled monthly return and its 12 lags
- AVC, AVC01, ..., AVC12: cross-sectionally scaled average daily dollar volume and its 12 lags
- SDC, SDC01, ..., SDC12: cross-sectionally scaled average daily dollar volume and its 12 lags

In order to capture market movements, I add SPY's (S&P 500 tracking Exchange Trading Fund) time-series standardized returns along with its 12 lags to the dataset, bringing the number of features to 91. Once all the variables are computed, along with their lags, I end up with 178 months (11/26/2003 to 12/19/2017) .

The typical momentum / reversal strategy splits a set of stocks into ten deciles based on their returns during a period. It then looks at the performance of a zero-investment combined portfolio which is long decile 10 (high returns) and short decile 1 (low returns) over the next period. If there's momentum in returns, this period's winners will win next period as well. Similarly, this period's loser will be losers next period. The combined portfolio will thus profit from both legs of the portfolio. Plus, each leg of the strategy can be analyzed separately. Reversal strategies are similar except they go long the losers and short the winners, expecting prices to reverse.

While the momentum strategy doesn't attempt to predict what next period's decile will be for a stock, it does expect the top decile stocks to do well, better than others over the next period. Thus we can think of momentum as predicting top decile over the next period for this period's top decile stocks. Reversal is just the opposite strategy.

I create the target variable along these lines. Instead of deciles (ten subgroups), I look at tertiles (three subgroups). If we group all stocks into three categories based on their returns this period, and code these categories as 0 / 1 / 2 (or Down / Mid / Up), momentum predicts a downward movement (0 /Down) over the next period for each stock in the Down portfolio this period. The only information momentum uses is a stock's current tertile. Reversal strategy is simply the opposite of momentum: it predicts Up for a stock in the current Down portfolio. When creating the target / outcome variable (Y), I use 33% and 67% cutoffs. In this scheme, if a stock's performance over the next month (YR) places it in the top tertile, then Y = 2 (Up).

There are some similarities between my data set construction strategy and that of Liew et al (2017). Both draw from past price and volume data, building from daily frequency to longer frequencies. they consider frequencies ranging from 1 day to

Table 1: Summary Statistics

This table reports the summary statistics for data used in this paper: return (R), target return (YR), average daily volume (AV), volatility or standard deviation of daily returns (DRSD), time-series standardized versions of return, volume, and volatility (RT, AVT, SDT) and cross-sectionally standardized return, volume, and volatility (RC, AVC, and SDC). Other features (RT01, ..., RT12) have similar distributions. The data set covers 409 stocks and 178 months (=72,802 observations).

	R	YR	AV	DRSD	RC	AVC	SDC	RT	AVT	SDT
mean	0.01	0.01	202.99	0.02	0.00	0.00	0.00	-0.03	0.16	-0.03
std	0.08	0.08	382.96	0.01	1.00	1.00	1.00	0.95	1.06	1.03
min	-0.81	-0.81	0.09	0.00	-8.03	-0.74	-2.51	-3.04	-2.62	-2.60
0.25	-0.03	-0.03	54.47	0.01	-0.58	-0.41	-0.65	-0.68	-0.64	-0.79
0.50	0.01	0.01	108.00	0.01	-0.02	-0.26	-0.21	-0.02	0.02	-0.24
0.75	0.06	0.06	211.17	0.02	0.55	0.02	0.41	0.62	0.87	0.57
max	1.82	1.82	16801.99	0.44	15.90	19.38	14.48	3.03	3.16	3.16

250: I focus on the 20-day frequency which, they find, can be predicted well.[3]They are interested in whether a security's return will be positive or negative (Up or Down), thus they have two categories and use 0% as the cutoff.

Table 1summarizes the dataset. It shows the raw returns (R, YR), volume (AV), and volatility (DRSD) as well as standardized versions of return, volume, and volatility features. Average monthly return is about 1% and avegare daily dollar trading volume is about $203Million. The volatility of daily returns averages 2%. Small deviations from 0-mean unit standard deviation for the scaled features are likely to be due to removing observations with missing values (after lagging returns and volumes). All series show outliers. The variation in cross sectionally-scaled data items is especially striking.

3 Results

How well does the neural network predict future returns? And how does its performance compare to other possible strategies? In this section, I anayze the performance of the network and compare it to those of two simple benchmarks. First, I introduce a few metrics from machine learning literature and then use them in my evaluations.

Given the setup in this project, in order to establish a long-short portfolio strategy, we need to predict which stocks are likely to be in the Up portfolio, and which are likely to be in the Down portfolio. The Mid portfolio provides a benchmark but would not be used in a portfolio strategy. Also, if there are short selling constraints, it might be useful to just look at the long side, in other words, the Up portfolio. So the focus is first identifying stocks that are likely to be in the Up portfolio, then those that are likely to be in the Down portfolio.

3.1 Benchmark strategies

Before evaluating the network's performance, I introduce two simple benchmark strategies. First, we can randomly assign a stock to one of the Up/Mid/Down portfolios. This strategy uses no information other than the fact that there are three

[3]Liew et al (2017) use daily data and construct overlapping observations. For the 20-day frame, they compute the return and average volume for day t using the last 20 observations, then again for day $t+1$ using the last 20 observations. Since 19 of the 20 observations used in the consecutive return and volume computations are the same, this method will result in a highly correlated time series dataset.

categories. I refer to this as the Random Strategy.

Second, given the (somewhat mixed) empirical evidence from momentum / reversal literature, if there are reversals in monthly returns, we can use this month's return to predict next month's return (Reversal Strategy). If a stock is in the Up portfolio this month, it is likely to reverse and end up in the Down portfolio next month. Similarly, Down stocks are likely to be in the Up portfolio. I use this scheme as the second benchmark strategy. (Mid stocks are placed in the same portfolio next month.) Note that a momentum strategy would just mean placing this month's Up stocks in the same portfolio next month. Also, momentum / reversal strategies typically use deciles and thus look at the more extreme positive and negative returns.

3.2 Precision, recall, and f1-score

Two of the standard tools used to assess the performance of a machine learning model are the classification report and the confusion matrix. They are shown in Table 2. These tools show how the model performs for each outcome (0 = Down, 1 = Mid, and 2 = Up). Each strategy (random benchmark, reversal benchmark, and neural network) is in a separate panel.

Classification reports show precision, recall, and F_1-scores as well as the number of observations in each class (support). Confusion matrices are reported in both counts and percentages. They present the counts of true positives (TP: 1 predicted as 1, etc), false negatives (FN: 1 predicted as 0 or 2), as well as false positives (FP: 0 or 2 predicted as 1). Precision is defined as predictions of an outcome that are actually correct: $(TP/(TP+FP))$. Recall is defined as predictions of an outcome that are predicted as such: $(TP/(TP+FN))$. F_1-score, given below, is the harmonic mean of precision and recall. It summarizes a model's combined performance along these two dimensions. The "perfect" model would have a perfect precision of 100% (i.e., when the model predicts Up, the outcome is indeed Up) and a perfect recall of 100% (i.e., the model predicts all Up stocks as such), which would result in an F_1-score of 100%.

$$F_1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

Panel A reports the results for random strategy. The model predicts each class a third of the time and it is correct a third of the time. So its precision and recall are 33% (or 34% depending on rounding). Somewhat larger support for Mid is the result of 33\% and 67\% cutoffs. While there are no surprises here, this panel does give a baseline.

Reversal strategy assigns this month's Up stocks to next month's Down portfolio, Down stocks to Up portfolio, and keeps Mid stocks in the Mid portfolio. If there were strong reversals in monthly returns, the majority of Down stocks would end up in Up portfolio, thus resulting in high precision and high recall for this class. This doesn't seem to be the case. Precision and recall for Up and Down portfolios are not any diffrent than those of the random strategy. The somewhat higher precision and recall (37%) reflects the slighly larger number of observations in this class.

Finally, in Panel C, the neural network's predictions are presented. The network differs from the benchmarks in one important way: it predicts more Ups and fewer Downs. 41% of its predictions are Up and 26% of the predictions are Down. This increases the recall to 43% for Up. While the network predicts more Ups than the benchmarks, its "false positives" for Up also increases: it predicts 2 when the actual outcome is 0 14% of the time, compared to 11% for the benchmarks. Similarly it predicts 2 when the outcome is 1 13% of the time, comparted to 11% (Random) and 10% (reversal). The increase in the number of predictions does not result in poorer precision for Up, though. Its precision is the same or very slighltly better for the three outcomes compared to the benchmarks. The increase in predicting Up without rteducing precision is the only success that the network accomplishes.

In Figure 2, the changes in precision and recall over test period are shown for Up, Mid, and Down predictions at each test date as well as the return on SPY for the priod for which the predictions are made (SPY_{YR}). Precision across classes vary

Table 2: Confusion Matrices and Classification Reports

This table reports the classification reports and confusion matrices obtained from the predictions of the neural network as well as the random and reversal benchmarks. The outcomes are 0 (Down), 1 (Mid), and 2 (Up). Confusion matrices present the counts of true positives (TP: 1 predicted as 1, etc), false negatives (FN: 1 predicted as 0 or 2), as well as false positives (FP: 0 or 2 predicted as 1). Precision is defined as percent predictions of an outcome that are actually correct $(TP/(TP+FP))$. Recall is defined as percent predictions of an outcome that are predicted as such $(TP/(TP+FN))$. F_1-score is the harmonic mean of these two metrics and summarizes model performance in these two dimensions.

$$F_1 = 2 * \frac{1}{\frac{1}{precision} + \frac{1}{recall}}$$

Classification Report					Confusion Matrix (Counts)					Confusion Matrix (Percent)				
Panel A: Random Strategy							Predicted					Predicted		
	prec.	recall	f1-sc.	support	Actual	0	1	2	Total		0	1	2	Total
0	0.33	0.33	0.33	4866	0	1605	1636	1625	4866	0	0.11	0.11	0.11	0.33
1	0.34	0.34	0.34	4995	1	1694	1677	1624	4995	1	0.12	0.11	0.11	0.34
2	0.34	0.34	0.34	4863	2	1597	1619	1647	4863	2	0.11	0.11	0.11	0.33
av/tot	0.33	0.33	0.33	14724	Total	4896	4932	4896	14724	Total	0.33	0.33	0.33	1.00
Panel B: Reversal Strategy														
	prec.	recall	f1-sc.	support	Actual	0	1	2	Total		0	1	2	Total
0	0.33	0.33	0.33	4866	0	1616	1596	1654	4866	0	0.11	0.11	0.11	0.33
1	0.37	0.37	0.37	4995	1	1607	1845	1543	4995	1	0.11	0.13	0.10	0.34
2	0.34	0.34	0.34	4863	2	1642	1551	1670	4863	2	0.11	0.11	0.11	0.33
av/tot	0.35	0.35	0.35	14724	Total	4865	4992	4867	14724	Total	0.33	0.34	0.33	1.00
Panel C: Neural Network														
	prec.	recall	f1-sc.	support	Actual	0	1	2	Total	Actual	0	1	2	Total
0	0.33	0.26	0.29	4866	0	1260	1585	2021	4866	0	0.09	0.11	0.14	0.33
1	0.38	0.37	0.38	4995	1	1257	1865	1873	4995	1	0.09	0.13	0.13	0.34
2	0.35	0.43	0.39	4863	2	1292	1483	2088	4863	2	0.09	0.10	0.14	0.33
av/tot	0.35	0.35	0.35	14724	Total	3809	4933	5982	14724	Total	0.26	0.34	0.41	1.00

Figure 2: Precision and Recall over Test Period

This figure shows the changes in precision and recall of Up (2) and Down (0) predictions over the test period. Confusion matrices present the counts of true positives (TP: 1 predicted as 1, etc), false negatives (FN: 1 predicted as 0 or 2), as well as false positives (FP: 0 or 2 predicted as 1). Precision is defined as percent predictions of a kind of outcome that are actually correct ($TP/(TP+FP)$). Recall is defined as percent predictions of a kind of outcome that are predicted as such ($TP/(TP+FN)$).

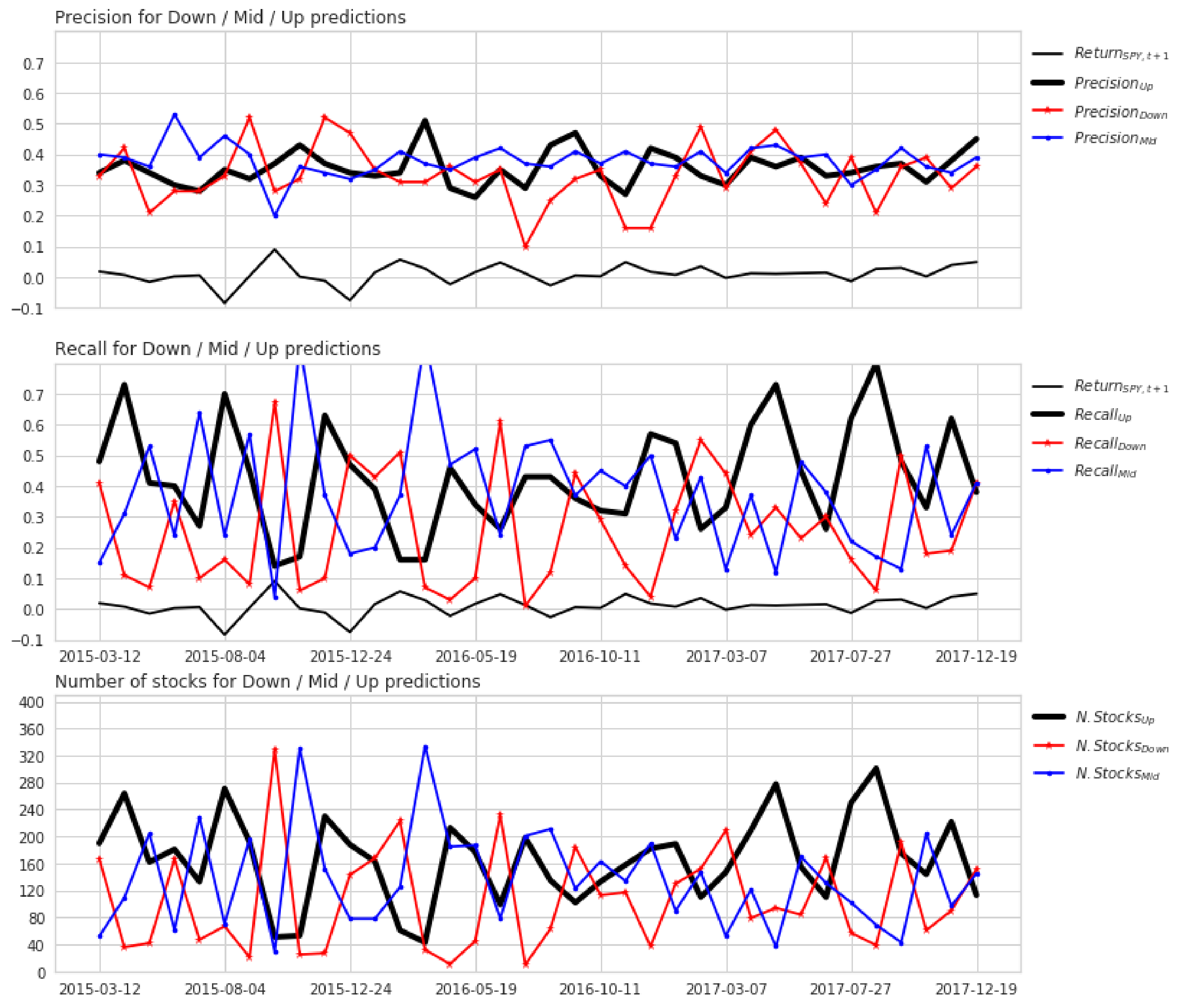


Table 3: Returns on Up and Down Portfolios

This table reports the average returns to Up (2) and Down (0) portfolios as well as the long - short Up - Down portfolios. Actual returns, network predictions, and two benchmark portfolios (Reversal and Random) are reported. There are 36 months in the test period.

	Actual			Network prediction			Reversal Benchmark			Random Benchmark		
	Up	Down	U - D	Up	Down	U - D	Up	Down	U - D	Up	Down	U - D
mean	0.0722	-0.0516	0.1238	0.0123	0.0122	0.0001	0.0102	0.0118	-0.0017	0.0108	0.0098	0.0009
std	0.0393	0.0338	0.0283	0.0352	0.0436	0.0229	0.0378	0.0324	0.0216	0.0337	0.0321	0.0066
min	-0.0260	-0.1691	0.0863	-0.0769	-0.1083	-0.0720	-0.1038	-0.0758	-0.0513	-0.0781	-0.0839	-0.0113
25%	0.0555	-0.0629	0.1022	-0.0060	-0.0060	-0.0095	-0.0078	-0.0046	-0.0100	0.0000	0.0000	-0.0018
50%	0.0685	-0.0483	0.1223	0.0115	0.0127	0.0044	0.0081	0.0140	0.0014	0.0064	0.0086	0.0018
75%	0.0883	-0.0326	0.1358	0.0281	0.0327	0.0109	0.0316	0.0265	0.0096	0.0274	0.0267	0.0050
max	0.1874	0.0107	0.2273	0.0917	0.1133	0.0416	0.0889	0.0723	0.0492	0.0789	0.0716	0.0144

across test dates, with Down showing more variation than the other two classes. They do not appear to be correlated with the market movements. Correlation coefficients between precisions and SPY_{YR} are -0.2136 (Down), -0.1354 (Mid) and 0.0932 (Up). The variation in recall is much more substantial. For instance Recall for Up has an average of 43% and a median of 42%. It peaks at 80% but also falls to 14%. Its standard deviation is 17%, compared to the standard deviation of precision (Up) which is 6%. The correlation coefficients between recalls and SPY_{YR} are 0.3503 (Down), -0.0732 (Mid), and -0.3975 (Up). These numbers and Figure 2imply that the network predicts less Ups (and more Downs) when the market rises and this may be hurting its performance. The bottom panel in the figure confirms that the network's number of predictions vary substantially over time as the recall figures show. The network predicts more Ups than Mids and Downs. The averages are 167, 137, and 106.

3.3 Returns

While the network has a very small advantage over its benchmarks in that it predicts Up stocks a bit better, this may or may not result in profits. Next, I investigate whether the neural network predictions translate into statistically and economically significant returns. Table 3 and Figure 3 summarize this analysis.

In Table 3, the actual Up portfolios earn an average return of 7.22% per month. This fairly large average is probably a result of the bull market that reigned during the test period (March 2015 through December 2017). For reference, the Mid portfolio earns an average of 1.1%. The actual Down portfolio earns an average return of -5.16%. The combined portfolio, long Up and short Down, has an average return of 12.38% percent with a standard deviation of 2.83%.

In contrast, the returns on the network predictions, the reversal portfolio and the random benchmark are essentially random. There seem to be no noticeable differences in Up portfolio returns and Down portfolio returns for neither the network predicted portfolios, nor the two benchmarks. This results in combined portfolios that earn about 0%. These figures suggest that while the network is able to predict Up stocks somewhat better than its benchmarks, this performance doesn't translate into significant profits for the Up portfolio, or the long-short Up-Down portfolio. Also, as measured in this paper, there are no reversals in returns for this time period and this sample. Figure 3 confirms this visually: the three portfolios seem to move pretty much the same way. Return analysis indicates that the neural network is not able to make predictions that will result profits.[4]

[4]Given the averages and standard deviations, I have not performed significance tests.

Figure 3: Average Returns on the Up and Down Portfolios

This figure shows the average returns for Up and Down portfolios over test period. Four portfolios are plotted: actual returns on the Up portfolio, neural network predictions, reversal benchmark returns, and random bechmark returns.

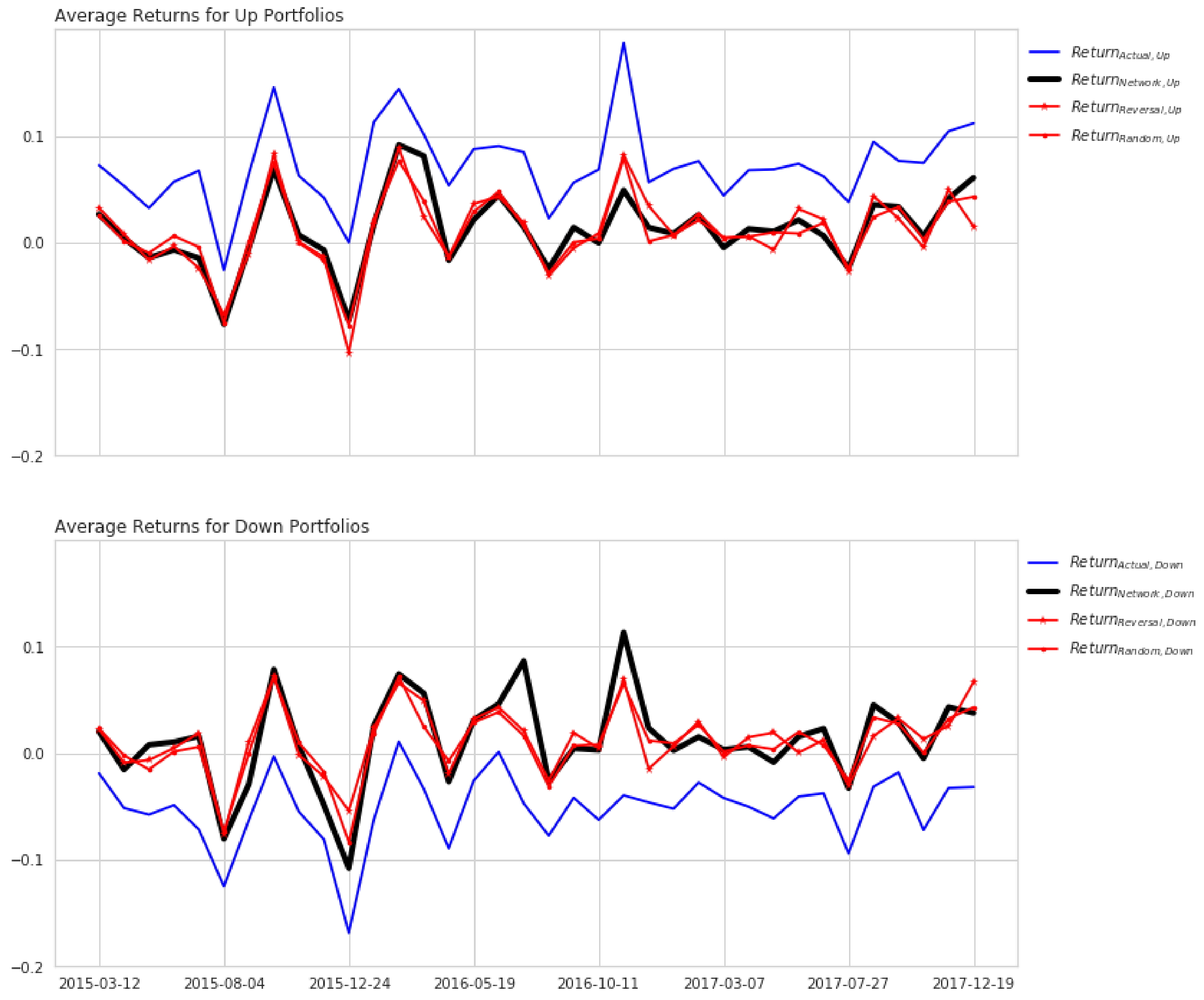


Table 4: Portfolio Characteristics and Predictions

This table reports the averages of several features (RT, AVT, SDT, RC, AVC, SDC, and SPY_RT) across prediction classes (Y_pred). It also reports averages of these charactaristics for correct and incorrect classifications.

Y_pred	RT	AVT	SDT	RC	AVC	SDC	SPY_RT
0	-0.3063	0.1781	0.0446	-0.2857	0.0126	0.1924	0.0097
1	0.1023	0.1905	-0.0251	-0.0010	-0.0184	-0.2825	0.2210
2	0.0683	-0.0865	-0.1256	0.1827	0.0072	0.1105	-0.0798
Differences							
All pairs	***	***	***	***		***	***
0-1	***		***	***		***	
0-2	***	***	***	***		***	***
1-2		***	***	***		***	***

Corr	RT	AVT	SDT	RC	AVC	SDC	SPY_RT
0	-0.0002	0.0715	-0.0397	0.0176	-0.0111	0.0058	0.0430
1	-0.0481	0.0807	-0.0628	-0.0322	0.0202	-0.0106	0.0461
Differences	**			**	.		

Significance codes: 0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

3.4 Further analysis

The neural network may still give some interesting insights into how future returns are related to current stock characteristics. To this end, I look at a few dimensions of network predictions. First, I look at whether correctness (predicted class is the actual class) is linearly related to cross-sectional characteristics of stocks. The model's performance on stocks varies from 58% to 11%. Regressing percent correct on cross-sectionally standardized average firm characteristics (RC, AVC, and SDC) yields no significant t-statistics for any regressors and has an R-squared of 0%. Thus, there doesn't appear to be a direct link between relative firm characteristic and correct predictions.

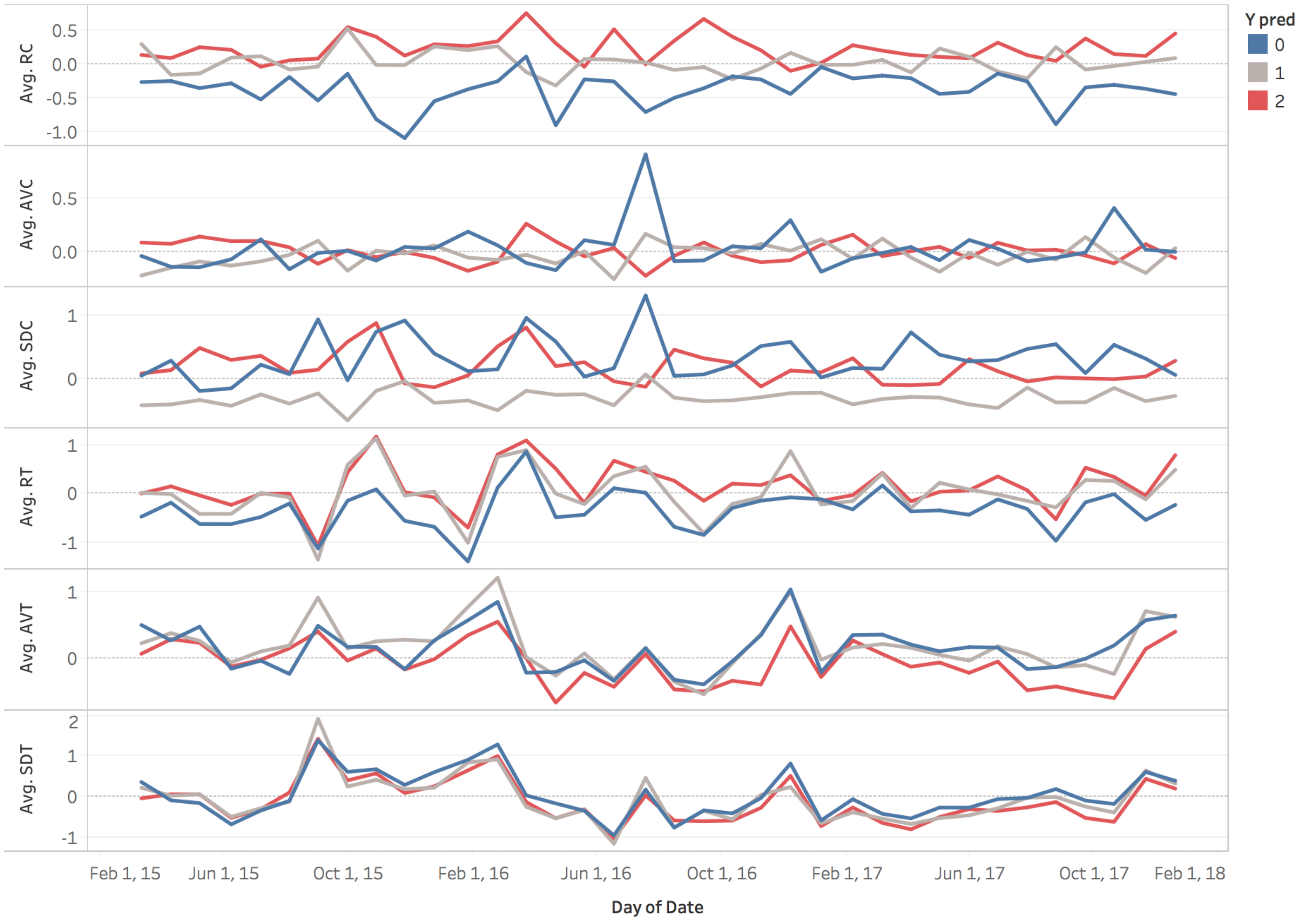
Next, I test whether there are differences in portfolio characteristics across predicted classes. In Table 4, I report several portfolio characteritics for each predicted class as well as whether the prediction is correct or not. Note that the feature set includes these features as well as their lags (91 features altogether). The table reports only seven features but not their lags. I test for differences across classes using Tukey's multiple comparison test. Figure 4 shows these characteristics visually.

Some interesting facts emerge from this analysis. As Figure 4 shows, Up, Down, and Mid predictions vary in some dimensions. Foir instance Down portfolio's RC (cross-sectionally standardized return) has an average of -0.2857, compared to that of Up (0.1827), and Mid (-0.0010). These differences are statistically significant. It appears that the network predicts Up for stocks that tend to have higher returns compared to their opeers for a given period. Similarly Down predictions tend to be associated with stocks that tend to have lower returns. Similarly, Down stocks tend to have lower returns relative to their own past, and have higher time-series and cross-sectional volatilities. While these findings do not necessarily imply a direct, linear relationship between stock characteristics and network predictions, they do suggest that the network is learning a variety of characteristics during training.

My analysis can be extended in a number of ways. Expanding the dataset into more illiquid securities could be another path forward. The current dataset covers some of the most active stocks. Longer time series could also provide stronger results. Another look at the frequency and lags could also be very useful. Momentum and reversal studies tend to look at very short (weeks) to very long (years) past data in predicting future returns. Similarly, this literature ranges from predicting weekly

Figure 4: Portfolio Characteristics and Predictions

This figure shows the averages of several features (RT, AVT, SDT, RC, AVC, and SDC) across prediction classes (Y_pred) over time.



returns to predicting multi-year returns. And there is also the impact of trading in portfolio returns, which is sometimes dealt with by introducing skips when constructing portfolios.

Conclusions

By training a single hidden layer neural network, I show that the direction of future price movements can be predicted, but the prediction strength is very limited and the model does not result in profitable portfolio strategies. Adding more data, more securities, and fine tuning the network are all possible ways to improve model performance. The weak prediction results could be due to low network capacity (which can be rectified by using a more powerful network) or a weak signal (which means that there is just not much information that can be teased of the noise in the dataset).

While neural networks have pushed artifical intelligence forward in many fields, and while investment industry has been shifting more towards quantitative prediction using neural networks and other machine learning models, their place in empirical finance has been limited. This is likely to change in the future.

References

Avramov, Doron, Chordia, Tarun, & Goyal, Amit. 2006. Liquidity and Autocorrelations in Individual Stock Returns. *The Journal of Finance*, **61**(5), 2365–2394.

Da, Zhi, Liu, Qianqiu, & Schaumburg, Ernst. 2014. A Closer Look at the Short-Term Return Reversal A Closer Look at the Short-Term Return Reversal. *Management Science*, **60**(3), 658–674.

Goodfellow, Ian, Bengio, Yoshua, & Courville, Aaron. 2016. *Deep Learning*. MIT Press.

Hou, Kewei, Xue, Chen, & Zhang, Lu. 2017. *Replicating Anomalies*. SSRN: http://ssrn.com/abstract=2961979.

Jegadeesh, Narasimhan. 1990. Evidence of Predictable Behavior of Security Returns. *The Journal of Finance*, **45**(3), 881–898.

Kurt Hornik, Maxwell Stinchcombe, Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks*, **2**(5), 359–366.

Liew, Jim Kyung-Soo, & Mayster, Boris. 2017. *Forecasting ETFs with Machine Learning Algorithms*. SSRN: https://ssrn.com/abstract=2899520.

Lin, Tsong-Wuu, & Yu, Chan-Chien. 2009. *Forecasting stock market with neural networks*. http://ssrn.com/abstract=1327544.

Messmer, Marcial. 2017. *Deep Learning and the Cross-Section of Expected Returns*. SSRN: https://ssrn.com/abstract=3081555.

Simpson, Marc William, Giudici, Emiliano, & Emery, John T. 2016. *One-Month Individual Stock Return Reversals and Industry Return Momentum*. SSRN: https://ssrn.com/abstract=1914629.

Technical Appendix

In this appendix, I provide technical details of my work in four steps: (1) acquiring data and preparing input for neural network training, (2) randomly searching hyperparameter space to help build a well-performing network, (3) sequentially making predictions and retraining the network over test period, and (4) analyzing results. I have developed five programs, each of which perform one of these steps. (There are two programs for the first step.) These programs are available on github:

https://github.com/MAydogdu/StockPriceDirectionPrediction_NeuralNetworks.git

I performed most of my work on a MacBook Pro (2.3 GHz Intel Core i5, 16 GB memory, High Sierra operating system). In order to use tensorflow in a python 2.7 jupyter notebook environment, I used a docker container. I used the directions here (http://deberker.com/archy/?p=135) to get started with docker containers on a Mac.

In previous iterations of this paper, for part of my work (random search - hyperparameter tuning), I used Amazon's cloud computing platform, AWS (Deep Learning AMI with Source Code (CUDA 9, Amazon Linux) on a p2.xlarge, $0.90/hour, 1 GPU) that seemed to work a bit faster than the laptop.

Acquiring data and preparing input

Program: StockDataAcquisition_2018April_SP500ReturnPredictionProject_V4_MA.ipynb

This program gets daily stock price and volume data for S&P 500 stocks from quandl.com and creates a dataset that is later used to train and test neural networks. A free API key is required to access quandl data. For each ticker-day, adjusted closing price and trading volume are needed. Adjusted closing price factors in splits and dividends and allows for a valid time series analysis. Quandl provides free access to stock data but ETF data are not free, so this version of the paper focuses on stocks. Earlier versions of this work used yahoo finance's API, which no longer seemed to be available in February 2018) That data set also includes adjusted prices and volume (number of shares traded) but also has data for ETFs. Google's finance API doesn't seem to have the adjusted closing price.

The inputs to the program are (1) set of ticker symbols, and (2) timeframe, i.e, the beginning and end dates. For this project, I use S&P 500 index constituents and 1/1/2000 - 1/31/2017. Daily adjusted prices and share volume are used to compute returns ($R_{i,t}$), average trading volume ($AV_{i,t}$), and the standard deviation of daily returns ($DRSD_{i,t}$) over 20-trading-day windows:

$R_{i,t} = (P_{i,t}/P_{i,t-20}) - 1$ where $P_{i,t}$ is the adjusted closing price of stock i on day t.

$AV_{i,t} = \frac{1}{20} * \sum_{j=0,19} V_{i,t-j}$where $V_{i,t}$ is the dollar trading volume (adjusted closing price times the number of shared traded) for stock i on day t.

$DRSD_{i,t} = \sqrt{\frac{1}{19} * \sum_{j=0,19} (R_{i,t-j} - \mu_{i,t})^2}$where $\mu_{i,t} = \frac{1}{20} * \sum_{j=0,19} R_{i,t-j}$

Roughly speaking, this corresponds to a monthly frequency and I refer to the dataset as such. This scheme sidesteps issues associated with calendar-month trading (end of the month / beginning of the month). It is relatively easy to modify this program to calendar month. It is also easy to modify this program to investigate other frequencies (weekly, quarterly, etc).

Feature scaling or standardization is important for training networks using gradient descent. I standardize raw returns, volumes, and volatilities in two ways. Time series scaling uses a 12-month frame (this month plus the preceding 11 months) to subtract the mean return from its raw return for each month and divide that by the standard deviation on a rolling basis.

This gives a scaled return, volume, and volatility for each stock, based on the last year. A large standardized return this month would imply a high return relative to the stock's performance over the last year. I also scale returns, volumes, and volatilities cross-sectionally on each date across all stocks. Here, a large standardized return would imply better performance over other stocks for that month. Also, 12 lags of these scaled variables are recorded.

$RT_{i,t} = \frac{R_{i,t}-\mu_{i,t}}{\sigma_{i,t}}$ where $\mu_{i,t} = \frac{1}{12} * \sum_{j=0,11} R_{i,t-j}$ and $\sigma_{i,t} = \sqrt{\frac{1}{11} * \sum_{j=0,11}(R_{i,t-j} - \mu_{i,t})^2}$

$RC_{i,t} = \frac{R_{i,t}-\mu_{it}}{\sigma_{it}}$ where $\mu_t = \frac{1}{n} * \sum_{j=1,n} R_{j,t}$ and $\sigma_t = \sqrt{\frac{1}{n-1} * \sum_{j=1,n}(R_{j,t} - \mu_t)^2}$ and n is the number of stocks.

Missing observations can be problematic when training and testing neural networks. Thus, I filter the stocks that do not span the whole time frame as well as observations with missing values due to lagging operations.[5] The final data set has 409 stocks and 178 dates (months).

The main dataset (SP500_Long_V4.CSV) includes all data items that will be used as features for a stock + date: the time-series and cross-sectional scaled data plus the lagged variables. It also includes the raw values of these features, base variables, next month's return (YR) and its target-coded value (Y). For instance, return-related variables are R, R01, ..., R12 (raw returns), RT, RT01, ..., RT12 (time-series standardized returns), and RC, RC01, ..., RC12 (cross-sectionally standardized returns). Similarly, there are 39 volume related returns and 39 volatility-related returns. Only the standardized variables are used in model building and analysis.

SP500_Long_V4.CSV

Ticker	Date	V	P	R	AV	DRSD	YR	Y	RT	RC	AVT	AVC	SDT	SDC	...	SDC12
AAPL	11/26/03	41	19	0.03	76.68	0.0178	-0.01	0	0.16	1.52	1.02	0.34	-0.45	0.01		0.18

Market indices

Program: MarketETFDataAcquisition_2018April_SP500ReturnPredictionProject_V4_MA.ipynb

This program starts with daily data from yahoo finance for S&P500 index tracking ETF: SPY. ETF data is saved in a CSV file. This file is processed in a way similar to the stocks, as described above. It outputs Indices_Long_V4.CSV. SPY time series standardized return and its 12 lags are used as "common" features. They measure the recent performance of the market and supplement firm-specific data.

Hyperparameters

Program: RandomSearchHyperparameters_2018April_SP500ReturnPredictionProject_V4_MA.ipynb

This program randomly searches the hyperparameter space to find optimal values for key hyperparameters in building and training a neural network. It starts with a dataset to be fed into a network. Then, over 500 trials (in two rounds), for each key hyperparameter, it randomly picks a value from the parameter set and trains and tests the performance of the resulting network.

[5] It is possible that removing stocks that do not have data for the whole time frame may introduce survivorship bias. I have not investigated the existence or impact of it in this study.

Input dataset, target / outcome variable, and training / validation / test split

The main dataset used in this version of the paper is the time series of scaled return and volume data (SP500_Long_V4.CSV) as described above. I add time series standardized SPY returns to this dataset from Indices_Wide_V4.CSV. This process yields in 91 features per ticker-date, 13 of which (SPY data) are common accross tickers on a given date. The results reported in this paper are based on this dataset.[6]

The typical momentum strategy splits a set of stocks into ten deciles based on their returns during a period, then looks at the performance of a zero-investment combined portfolio which is long decile 10 and short decile 1 over the next period. If there's momentum in returns, this period's winners will win next period as well. Similarly, this period's loser will be losers next period. The combined portfolio will thus profit from both legs of the portfolio. Plus, each leg of the strategy can be analyzed separately. Reversal strategy shorts decile 10 and buys decile 1, expecting reversals on both legs.

We can think of momentum / reversal in context of predicting future returns, more specifically predicting a stock's performance relative to other stocks over the next period. If we group all stocks into three categories based on their returns this period, and code these categories as 0 / 1 / 2 (or Down / Mid / Up), momentum implies a downward movement (0 / Down) for each stock in the Down portfolio over the next period. The only information momentum uses is a stock's portfolio membership this period. A reversal strategy is simply the opposite of momentum: reversal predicts Up for a stock in the current Down portfolio.

When creating the target / outcome variable (Y), I make one adjustment to the classification scheme above. If this scheme is used with CRSP data, each decile portfolio will have hundreds of stocks. The "universe" of stocks in this paper is 409 since I start with S&P500 stocks. Also, when training a neural network, it is important that the network sees many examples form each class. Using the 10% and 90% cutoffs, there will be disproportionately small number of observations for Down and Up categories. In order to have a larger set of observations for these subsets, I use 33% and 67% cutoffs. In this scheme, if a stock's performance over the next month (YR) places it in the top tertile, then Y = 2 (Up). In previous versions, I used oversampling (SMOTE in scikit-learn) for the training dataset (which is explained next) that yields a dataset that has an equal number of observations across target values. The results didn't seem too different with the 3-class scheme with or without oversampling, thus, I removed it in this version.

Time series nature of the dataset is important. I sort the dataset by dates and split the dates into training and test dates (80% and 20%) so that the networks will be trained using the earlier data. I further split the first dataset into two: 80% for training and 10% for validation, validation set including the later dates. Training set will be used to train the network, validation set will be used to check the performance of the current network parameters to decide when to end training, and test set will be used assess the performance of the resulting network.

Random search for optimal hyperparameters

Hyperparameters determine the shape of the neural network as well as its predictive performance. There are a large number of parameters that go into the design of the network but some of the parameters are more important than others. Here I focus

[6]Earlier versions of the paper considered pulling in market-wide data in two different ways. First, I considered removing the cross sectional scaled variables (which really capture a stock's performance vis-a-vis other stocks on a given date) with *all* stocks' cross-sectional data, similar to Liew et al (2017). For instance, if there are two stocks in the data set (AAPL and MSFT), for each date, AAPL would have its own cross-sectionally scaled features as well as time-series scaled features for both AAPL and MSFT. With 13 return variables, 13 volume variables, and 13 volatility variables, this "wide" dataset, which by definition is common across all stocks per date, results in 409*(13 + 13 + 13) = 15,951 features that capture market data. The advantage of this scheme is that, for each stock, the network not only has information on that stock's performance but also on all other stocks' performances. I have tried using the actual features as well as a smaller number of features resulting from a principal component analysis (PCA) dimension reduction. Neither strategy seemed to improve model performance. I also considered using time-series scaled returns, volatility, and volume for four ETFs with long trading history: SPY, DIA, MDY, and QQQ to capture market-wide trading. Appending this dataset to the long data didn't seem to improve the model either. In the end, I only used SPY standardized returns as described in the previous paragraph to capture market data.

on a subset of parameters that are commonly used. I considered the several hyperparameters in my network design and training.

First, the shape of the network should be considered. The network's input layer is determined by the features and its output layer is determined by what's predicted. In this project, the output is the respective probabilities of a stocks possible move next month Up / Mid / Down. For activation for the output layer, softmax gives the scaled probabilities of each outcome so that probabilities add up to 1. Beyond the input and output layer, the number of hidden layers need to be determined. Here, I use a single hidden layer network. Earlier experiments indicated that adding a second layer didn't much affect network performance. Besides, a single hidden-layer network, properly trained, should be able to capture many nonlinear functions of the features as Hornik et al (1989) posited.

In order to fully define the network, two more parameters need to be defined: the number of nodes in the hidden layer, or the layer width, and the activation function. These are two of the hyperparameters that I include in random search. For activation, I consider three commonly used functions: (1) the rectifier (or as units with this function are known as rectified linear unit (ReLU)) which is probably the most commonly used activation function currently, (2) one of its modified versions, leaky ReLU, and (3) the hyperbolic tangent function (tanh). For layer width, I consider a random draw from the 30 - 300 range, similar to Messmer (2017) . For reference, the number of outputs is 3 and the number of features is 91.

$LayerWidth \sim$int(exp(η)), where $\eta \sim$ Uniform(log(A), log(B)) where A is 30 and B is 300.

Beyond the shape of the network and the activation functions, the optimal weights and biases for each node need to be determined through training. I provide a cursory description of this optimization here. First, weights and biases are initialized to some values. I use xavier initialization for weights, which essentially, ensures that the weights are not too small or too large, and zeros for biases. Next, the weights are optimized in an iterative fashion. Now that the network is completely defined (shape, activations, and initial values for node weights and biases), the inputs can be forward propogated to the outputs through the hidden layer to make a prediction.

We'd like the predictions to be as close to the actual results. To measure the network's performance, we define a cost function, in this case, softmax cross entropy cost function. Gradient descent is an algorithm that's very commonly used to iteratively update the weights and biases in a way that reduces the cost (which, essentially implies that the predictions are as close to targets as possible). This is done by taking the partial derivative of the cost with respect to weights and biases, multiplying the derivatives with the learning rate (also referred to as alpha) and subtracting those quantities for the current values to compute the updated weights and biases. The learning rate controls the size of the updation step: large steps may miss the cost minima and small steps may result in a very gradual and slow optimizations. It is thus an important hyperparameter, possibly the most important hyperparameter to optimize. This updation of weights backwards is known as backpropogation.

There are two important features of the gradient descent that I incorporate into random search. First, I use Adam optimizer, which improves upon the gradient descent by updating each weight and bias using a separate learning rate tied to alpha, and by changing the speed (momentum) of updation based on an autoregressive weight scheme that takes more recent changes into more consideration. This algorithm has three other parameters beyond alpha; I use the tensorflow defaults for them. Second, I use stochastic gradient descent, which, instead of forward propogating through the whole training set ("one epoch") at once, splits the training set into minibatches or small training subsets, and updates parameters iteratively after each minibatch. Minibatch size is a hyperparameter that I perform a random search on.

Here is a summary of network parameters.

Parameter	Choices
Number of hidden layers	1 (fixed)
Number of nodes in a layer ("layer width")	$\sim$int(exp(η)), where $\eta \sim$ Uniform(log(A), log(B)) where A is 30 and B is 300.
Activation functions	relu, leaky relu, tanh
Learning rate	random lognormal from 0.00001, 0.0001, 0.001, 0.01, 0.1
Minibatch size	32, 64, 128, 256, 512, 1024
Optimizer	Adam (fixed)

I use two other methods to help improve the performance of the random search. First, I use L2-regularization. This adds additional restrictions on the cost function such that larger parameter values are penalized. This helps avoid overfitting. Also, I use dropout, which randomly sets some (for instance 50%) of network weights and biases at each iteration. Dropout also helps avoid overfitting.

Finally, I use early stopping to conclude each network training trial.[7] After each epoch, I check whether the validation set error has stopped falling over the last several epochs. If the error has started rising, I stop training and set the network parameters to those associated with the lowest validation set error obtained.

I trained five hundred neural networks within this framework in two stages. The first stage (about 220 trials) searched the parameter space outlined above with a "patience" of 250 (if the validation set error doesn't fall over the last 250 epochs, stop training). Given the large patience value, this stage essentially doeesn't use early stopping. I focused on the model's performance over the Up portfolio more than its overall performance because this would yield a network that could be used to predict which stocks would go up over the next month. This network could then be used to implement a long only strategy (buy Up predictions) as well as a long-short portfolio (buy Up, short sell Down). I required a precision of at least 35% for Up (which is better than that of random guessing benchmark) and selected the models with highest Recall for Up at each epoch.

The results from each trial (classification report, confusion matrix, number of epochs, best epoch, etc) get written into a log in jupyter. I pulled that log into the Atom editor and reshaped this output into tabular form that I then pulled the output into Google Shets and then Tableau. I used Tableau to visually inspect model performance (specifically, f1-score for Up/2) and noted that leaky ReLU activation, middle-range number of nodes, larger minibatch sizes and smaller learning rates all appeared to help model performance. I then used a patience of 20 and trained 280 networks with leaky ReLU, and minibatch sizes of 128 or 256, keeping other hyperparameter ranges as before. This second round random search resulted in the following network that had the highest recall for Up: activation: leaky ReLU, minibatch_size: 128, learning rate: 0.00001, number of hidden layer nodes: 104.

Testing the model and generating the results over the test dataset

Program: GenerateTestResults_2018April_SP500ReturnPredictionProject_V4_MA.ipynb

Once the network parameters are set, I get the final results progressively over time. I first set the earlier 80% of dates for training and validation (latest 20% of these dates being the validation set) and keep the rest of the data for testing. I train a network as before, using the network parameters chosen above. I then test the network's performance on the earliest test date and record the results. Once those results are recorded, that test date gets added to the training/validation set, so that the network is trained using this newly expanded set. For each training iteration after the very first, network parameters are initialized to the values from the previous session. I repeat this process for all the test dates. The results reported in this paper are based on the test set performance of these networks. The program produces an output, SP500_Results_V4.CSV, that has, for each date+ticker in the test set, the model's predictions as well as benchmark predictions. It also produces a log that has classification reports and confusion matrices, which was used to generate further insight into the workings of the model.

Analyzing the results

Program: AnalyzeResults_2018April_SP500ReturnPredictionProject_V4_MA.ipynb

I use this program to analyze model performance and generate tables and graphs that are in the paper. For one of the tables, I parse the output from classification reports and confusion matrices. Some of the code in this program is data exploration and not all graphs are in the paper. Also, I perform t-test using a small R program (TestMeans.R) as part of data analysis. Finally, I use Tableau for some data exploration and one graph.

[7]My implementation is similar to that outlined in Goodfellow et al (2016) on page 240.