MDPI

Article

Are Markets Truly Efficient? Experiments using Deep Learning for Market Movement Prediction

Sanjiv Das [1,†], Karthik Mokashi [2,†] and Robbie Culkin [3,†]

[1] Santa Clara University; srdas@scu.edu
[2] Santa Clara University; kmokashi@scu.edu
[3] Santa Clara University; rculkin@scu.edu
* Correspondence: srdas@scu.edu; Tel.: +1-408-554-2776
† Current address: 500 El Camino Real, Santa Clara, CA 95053.

Academic Editor: name
Version April 29, 2018 submitted to Algorithms

Abstract: We examine the use of deep learning (neural networks) to predict the movement of the S&P 500 Index using past returns of all the stocks in the index. Our analysis finds that the future direction of the S&P 500 index can be weakly predicted by the prior movements of the underlying stocks in the index. Decomposition of the prediction error indicates that most of the lack of predictability comes from randomness and only a little from nonstationarity. We believe this is the first test of S&P500 market efficiency that uses a very large information set, and it extends the domain of weak-form market efficiency tests.

Keywords: Deep neural nets; market efficiency; market prediction

1. Introduction

There is a long history of research on market efficiency, beginning with [1]. In a follow up article more than three decades later, [2] revisited the evidence on market efficiency, and found in favor of its broad existence using effective methods, especially event studies.

Tests of market efficiency take many forms, such as tests of autocorrelation on univariate time series stock (or stock index) return data, as in [3]. These tests of the weak-form market efficiency hypothesis take a narrow view of the information set on which the tests are conditioned, i.e., the past history of the time series being predicted. Event study tests, beginning with [4] also use information from the same time series, and complement this data with the return on the market index. These tests in general, also seem to support overall market efficiency, though of course, some anomalies do exist in the short-term. Tied up with an assessment of affirmative market efficiency is a corresponding proposition of an absence of stock predictability. In this paper, we revisit whether markets are predictable, in particular, if the direction of the broad market, not just one stock, might be predicted with higher than random chance.

What is different about this assessment compared to the long history of research showing stock market unpredictability? First, whereas weak-form tests have relied upon the past history of a *single* time series of returns, we will assess the predictability of the daily direction of S&P500 index using historical data on *all* stocks in the index for a preceding period of time. Therefore, this is a large-scale generalization of the information set used in testing market efficiency.

Second, models for market efficiency tests have usually been based on linear statistical specifications, such as multivariate regression. Here, we will use highly nonlinear deep neural networks to specify the functional relation between the sign of the move in the S&P500 on day $(t+1)$ and the T-day history of all stocks in the index until day t. The hope is that what linear models cannot

pick up, we may train nonlinear models to learn. Further, by expanding the information set by a huge order of magnitude, we are able to re-examine weak form efficiency more comprehensively.

Market efficiency, evidence by a lack of predictability is usually supported by two features of the data. First, *randomness*. A purely random series is, by definition, unpredictable. Second, *non-stationarity*. This belies predictability because the fitted parameters from historical data have limited shelf-life, and become invalid in forecasting the future evolution of returns. Our results are two-fold. First, we find that predictability of the S&P return using all 500 stocks in the index exists, but is weak, and too small to be profitable after taking transactions costs into account. Second, we find evidence that both randomness and non-stationarity make prediction difficult, but randomness plays a bigger role in making markets efficient.

One may argue that tests of market efficiency that assess whether mutual funds can beat the market (see [5]) are in effect using much larger conditioning data sets than single stock series. Once again of course, these are time series tests with a single series of the fund at a given point in time. And in addition, these tests are linear, and do not admit nonlinearity in their specifications. Therefore, we believe that the use of deep learning neural nets to develop a stock index predictability model are novel, and despite the long history of empirical support for market efficiency, such models, in theory, may be using information sets in a different way, one that has not been assessed before. Whereas it may be quite impossible for a human to detect predictive patterns in reams of stock data, it may be within the realm of possibility for machine intelligence. And, though the S&P index seems only weakly predictable, other asset classes, such as small stocks, may prove to be less efficient in the face of large information sets.

In the following sections, we explore this new approach to testing market efficiency. Section 2 recaps the architecture of neural nets, including a discussion of the backpropogation technique. Section 3 describes the data engineering used in this paper, and has some interesting ways in which we handle the data in order to render it suitable for consumption by a deep learning model. Section 4 describes the experimental structure used, and Section 5 describes the results. Section 6 provides concluding discussion.

2. Neural Networks

Neural networks (NNs), the primary methodological tool used in our analysis, have been in existence since the 1940s, beginning with the early work of [6], but have become immensely popular in the last decade due to the increase in computing power and the explosion of available data. Neural networks are set up to imitate the decision making process of the human brain. An example of a Neural network is shown in Figure 1. For our stock prediction algorithm, we would inject the information set (in our case, data histories on all stocks in the index) into the (left side) of the neural network. Each layer in the network comprises nodes, i.e., a set of mathematical functions that transform the data, which is then fed into the nodes (functions) in the next layer, until finally the output layer is reached. The output function in our case generates a value in the interval $(0, 1)$, i.e., the probability that the market will go up in the forecast period. Intuitively, the neural network is a giant nonlinear function that takes in a large number of inputs and generates a prediction about the likelihood that the market will rise in the ensuing period of time.

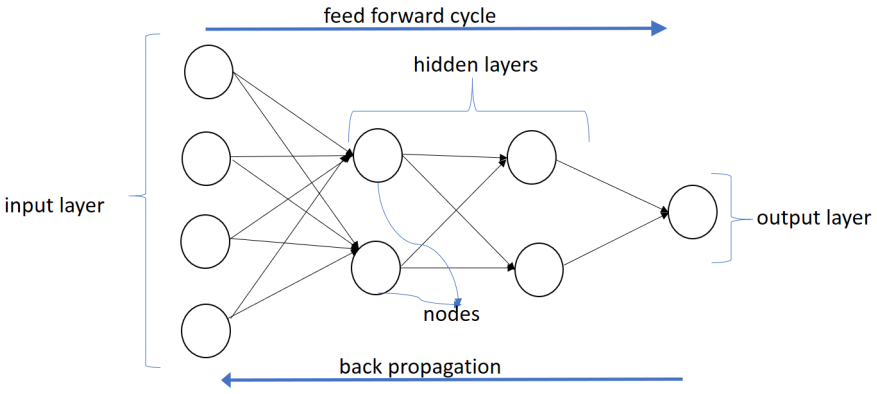

Figure 1. A simple neural network with two hidden layers of two nodes each, four inputs, and a single output node.

The first layer with input values (usually denoted x) is called the input layer, the middle layers with function values (usually denoted y) are called the hidden layers and the last layer with z values is called the output layer. The functions at each node are called "activation" functions, borrowing from neuroscience terminology where neurons are said to fire or activate. There may be several hidden layers and multiple perceptrons (nodes) in each hidden layer. Neural nets with multiple layers (sometimes more than 100 layers) have resulted in the nomenclature of "deep learning" neural nets. Higher number of layers or nodes naturally lead to longer processing times. The output layer can either have multiple outputs or a single output (binary classifier). In the case of a binary output, the output layer is implemented with squashing functions that emit an output value between 0 and 1. A typical binary output node is implemented using the "sigmoid" function, also known as the "output activation" function.

$$z(a) = \frac{1}{1+\exp(-a)} \in (0,1) \tag{1}$$

where a is called the "net input" to the sigmoid function, and is a weighted sum of inputs from the preceding hidden layer. The neural network we use in our paper is a fully connected network i.e., each input layer node is connected to each node in the first hidden layer. Each node in the first hidden layer is connected to every node in the subsequent layer, and so on, till we reach the output node(s).

2.1. Working of a Neural Network

The easiest way to understand how a neural network works is to note that each of the inputs (the x values) are provided as weighted inputs for each hidden layer (y values). The weights vary for each node in every hidden layer and the output layer. The output of a hidden layer is passed through a function, and weighted as an input to the next hidden layer and so on.

In our example, the inputs x are passed to the first hidden layer y, and at each node in the first hidden layer, the inputs are weighted to arrive at what is called the "net input". So for the network shown in Figure 1, there are two nodes in the hidden layer and the net input for each node is as follows, given the four input values x_1, x_2, x_3, x_4.

$$a_1^{(1)} = \sum_{i=1}^{4} w_{i1}^{(1)} x_i + b_1^{(1)} \tag{2}$$

$$a_2^{(1)} = \sum_{i=1}^{4} w_{i2}^{(1)} x_i + b_2^{(1)} \tag{3}$$

where $a_j^{(1)}, j = 1, 2$ are the two net input values, and the weights are $w_{ij}^{(1)}, i = 1, 2, 3, 4; j = 1, 2$, which are the parameters of the model that need to be calibrated. The constant terms $b_j^{(1)}, j = 1, 2$ are the "bias" terms and cause a linear shift the net input value. The superscripts denote the number of the hidden layer. The first hidden layer has ten parameters (eight w values and the two bias b values). The second hidden layer has six parameters. The two hidden layer nodes each generate a single output through a nonlinear transformation of the net input, i.e., if we use sigmoid functions, then these would be as follows.

$$y_j^{(1)} = \frac{1}{1 + \exp(-a_j^{(1)})}, \quad j = 1, 2 \tag{4}$$

These values are then passed to the second hidden layer, where the "net input" equations are as follows:

$$a_1^{(2)} = \sum_{j=1}^{2} w_{j1}^{(2)} y_j^{(1)} + b_1^{(2)} \tag{5}$$

$$a_2^{(2)} = \sum_{j=1}^{2} w_{j2}^{(2)} y_j^{(1)} + b_2^{(2)} \tag{6}$$

And of course, after activation, the output of the second hidden layer is two values:

$$y_j^{(2)} = \frac{1}{1 + \exp(-a_j^{(2)})}, \quad j = 1, 2 \tag{7}$$

Now, the output node's net input would be

$$a = \sum_{j=1}^{2} w_j y_j^{(2)} + b \tag{8}$$

with final activation value given by equation (1).

$$y = \frac{1}{1 + \exp(-a)} \in (0, 1) \tag{9}$$

This model example in Figure 1 has 19 parameters, five for each node in the first hidden layer, and three for each node in the second hidden layer, and three for the output node.

When each input layer is connected to each hidden layer and the values generated flow forward, as in Figure 1, the neural network is denoted as a "fully connected, feed-forward" network. The weights $\{w_{ij}^{(r)}, b_j^{(r)}, w, b\}$, for all i, j, and all hidden layers r, are chosen to minimize the prediction error of the neural network on training data. The error is combined into a loss function across all training data, usually by choosing the loss function to be the sum of squared prediction error, or some other loss function such as entropy. The calibrated weights will determine the accuracy of the network in predicting the output, i.e., the movement of the S&P index in our tests of market efficiency.

The biggest challenge in calibrating an accurate neural network, is calculating the weights to be used at each layer. It is easy to imagine that calculating the optimal weights one at a time at each layer is computationally complex and inefficient. In the picture shown above, we have to calculate only 19 weights. But in the prediction model we develop in this paper, we will be fitting models with hundreds of thousands of parameters! This complexity was in fact the primary reason that neural networks did not grow in popularity and complexity when they were invented. The onerous computational requirement to fit large numbers of weights (w, b) resulted in long model training times. Today this challenge has been surmounted by special purpose hardware, and commoditized programming tools,

such as Google's TensorFlow[1] and other popular open source implementations of neural networks such as MXNet[2], and h2o.[3] In conjunction with better hardware and software, one single mathematical innovation from more than three decades ago is solely responsible for the incredible efficacy of deep learning neural networks—backpropagation, often shortened to "BackProp."

2.2. Calibration via Backpropagation

We may think of the neural network as a large-scale function approximation, where the network is calibrated to as to best fit outputs (using the inputs) to be as close to the true output as possible. Therefore, calibrating a neural network involves optimization, where a loss function $L(\mathbf{w}, \mathbf{b})$ is minimized, by choosing the weights $\mathbf{w}, \mathbf{b}$ of the model as best as possible. There are many loss functionals that may be used, each of them expressing the aggregate difference between the true output and model output in different ways.

Finding the best parameters by minimizing the loss function provides a best effort result at uncovering a function that approximates the mapping from inputs to outputs. Because neural net architectures can be as large and complex as the modeler likes, these nonlinear functions are often called "universal approximators"—they can approximate any function. Our goal is to find the best prediction function that approximates how the history of stocks' returns determines the future direction of the stock index. This goal is supported by the Universal Approximation Theorem, which asserts that a neural net with a single hidden layer, a large number of nodes, with continuous, bounded, and non-constant activation functions can approximate any function arbitrarily closely ([7]), though this has been debated by [8].

Optimization is undertaken by adjusting the weights in the model by moving each weight in the direction that reduces the loss function. This direction is given by the gradients of the loss function $\frac{\partial L}{\partial \mathbf{w}}$, $\frac{\partial L}{\partial \mathbf{b}}$ with respect to the weights. This fitting approach is known as "gradient descent". Given the large number of parameters (weights) in the huge neural nets in vogue today, gradient computation must be efficient. Preferably, it should be analytical.

The backpropagation algorithm does exactly that, i.e., it provides an analytical scheme for calculating all gradients. The evolution of the backpropagation algorithm covers the past 50 years, through seminal work by [9]; [10]; [11]; [12], the latter being the implementation that is widely used. It solves the issue of slow training by guessing a list of weights initially and propagating the activation values forward through the network. Then at the last node, the gradient for the weights is computed analytically, and the weight is adjusted to a more ideal value and then the gradients for all preceding nodes may also be calculated using analytic functions. Then, all the weights of all the nodes in the previous layers are adjusted accordingly. This results in an improvement in the loss function. One such forward pass followed by a backpropagation pass is called an "epoch" and is simply one iteration to minimize the loss function.

The modeler decides how many epochs to use. Calibration time is linear in the number of epochs. Of course, one may use a large number of epochs and drive down the loss function aggressively, but then the model may be overfitted on the training data, resulting in poor out-of-sample performance. Other approaches are used to manage overfitting, such as training the model on smaller random batch sizes of the data ("stochastic batch gradient descent"), or randomly dropping some nodes in each epoch (known as the "dropout" technique). We will employ these techniques in the empirical work that follows.

Thus, there is a massive, though manageable amount of computation, but with one pass forward and one pass backward all weights are adjusted. This allows for speeding up in training times.

---

[1] https://www.tensorflow.org/
[2] http://mxnet.io/
[3] https://www.h2o.ai/

Furthermore the use of parallel GPUs to perform these computations have allowed the split of the computation across multiple cores to further enhance performance. Backpropagation is a standard algorithm available in all of the popular packages today. Whereas this section only offered a sketch of the calibration process in deep learning neural nets, there are many references that provide detailed expositions to which the reader may refer.[4]

We move on now to discuss the data used in our prediction experiment to test for market efficiency.

3. Data

The primary source of data is the daily returns of the S&P index and its underlying stocks extracted from the Wharton Research Data Services (WRDS) database, from April 1963 to December 2016. This gives us approximately 54 years of daily return data for our analysis, across thousands of stocks. Note that there are many tickers involved as they move in and out of the index.

We built the data for the daily returns of the component stocks as follows. Using the Compustat database a query was run to collect all tickers of the S&P 500 between April 1963 and December 2016. The unique identifiers from the resulting file were then run against the WRDS database to collect historical security prices for all the securities in the time period of our analysis. This resulted in a "long" formatted data frame. This was further reshaped into the "wide" format for our analysis. A long format data frame has a list of dates, stock tickers and daily returns in sequence, i.e., a three-column panel data set. This format while required for plotting, is not ideal for feeding as an input to an algorithm. The long format is simply a typical panel data set.

In contrast a wide format data frame has one column of dates, and each of the stocks has one column which contains the daily returns of the stock on that particular date. If the stock was not part of the S&P Index on that date, the entry contains a NA value.

Reshaping the data into the wide format allows us to setup the daily returns of all component stocks that were a part of the index between April 1963 and December 2016. Finally the wide data set containing stock returns is merged with the S&P index returns data set to generate one large data set for the analysis. This data set now contains the date, S&P return, and the return of all stocks that were part of the index. Since there are many additions and deletions of stocks from the index, we end up with 5370 different stock tickers over the sample period, and these account for several additions to, and deletions from, the index, over a period of 54 years (13,532 trading days) covered by our sample. This data set contains many missing values, as only 500 stocks at a time exist in the index, and is therefore a semi-sparse data matrix. The degree of sparsity is 68.54%. We also counted the proportion of days on which the index rose versus fell, which is $\sim 52\%$ of the time.

3.1. Feature Engineering

We carried out feature engineering to prepare the data for calibration with a deep learning neural net. This feature creation essentially lines up L days of stock history from day $(t-1)$ back to day $(t-L)$, placed side by side in the same row for a given date. Therefore, there will be a block of columns for $(t-1)$, another block for day $(t-2)$, and so on, until a final block of columns for day $(t-L)$. This results in a very large feature set. In order to keep each block of columns manageable, we transformed the data in a special way, discussed next.

Since over the past 54 years many stocks have entered and exited the S&P 500 index, we had to account for the fact that a security would have influenced the movement of the index only during a certain period of time. Also, even for just the L-day history in the feature set, some columns may have missing data as stocks may have left the index and others may have entered it. We apply an elegant and simple idea to resolving the problem of missing data in the feature set.

---

[4] For an accessible, technical introduction to neural nets, see http://srdas.github.io/DLBook. Excellent books on the subject are by [13]; [14]. An excellent online resource is Michael Nielsen's book: http://neuralnetworksanddeeplearning.com/.

We approach this problem by using percentiles. We convert the daily returns of the underlying component stocks to percentiles. The percentiles considered are at the 1, 2, 3, 5, 10, 15, 20, 30, 40, 50, 60, 70, 80, 85, 90, 95, 97, 98, and 99 percentile levels. This results in a data set that has the daily return of the S&P as the dependent variable, and 19 percentiles as the explanatory variables per day. These percentiles allow us to capture the entire distribution through a discrete approximation, and places each day's data into a standard data structure for the probability distribution of returns. This reorganization of the data allows us to abstract from the fact that the underlying stocks in the index are changing, yet consider the influence of their returns in predicting the forward movement of the S&P. It allows us to model the return of the index based on its 500 component stocks at any point in time. It also results in a tidier data set for the prediction exercise that is the main focus of our analysis.

3.2. Statistical Features of the Data Set

We begin by reviewing some attributes of the data. The distribution of the S&P returns is shown in Figure 2. As seen from the histogram returns are tightly distributed with most of the daily returns between -3.5% and $+3.5\%$. The distribution has mild negative skewness and high kurtosis, both known features of index return data.

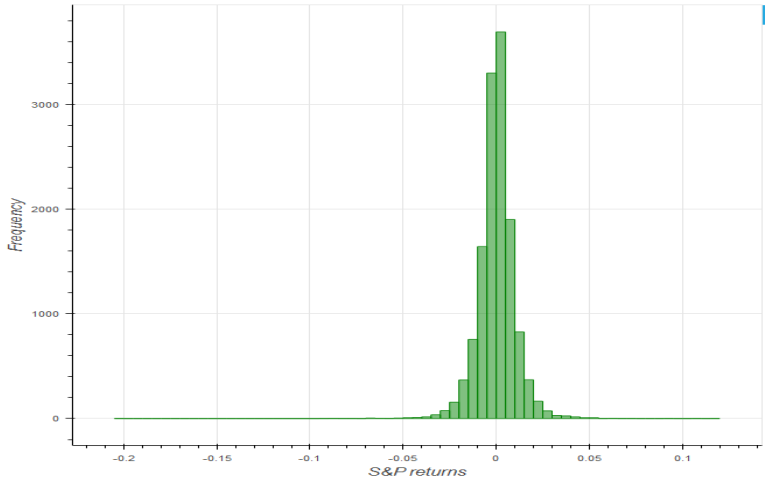

Figure 2. The distribution of daily S&P 500 index returns from 1963-2016. The mean return is 0.00031 and the standard deviation is 0.01. The skewness and kurtosis are -0.62 and 20.68, respectively.

A plot of the daily returns confirms the presence of large outliers, leading to kurtosis, and periods of high and low volatility, see Figure 3.

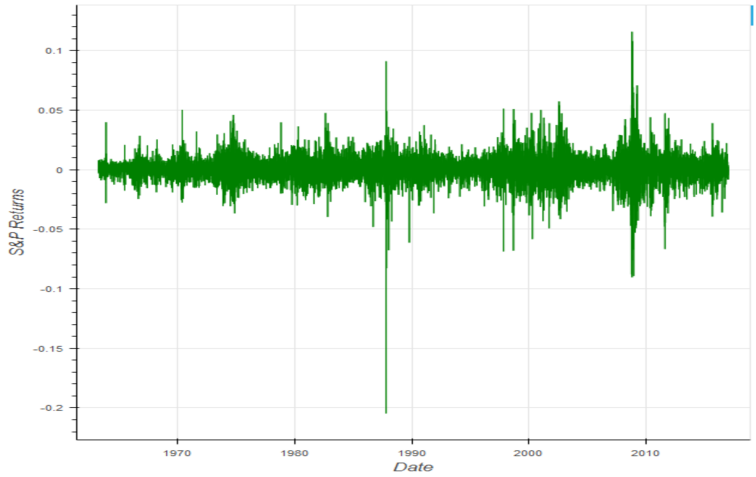

Figure 3. Daily S&P 500 index returns from 1963-2016.

4. Experiments

We use the data structure we created as described above to forecast the direction of movement $M = \{+, -\}$ in the S&P500 index for any given day d as a function of data over a lookback period of L days, i.e., over a set of days $d = \{d-1, d-2, \ldots, d-L\}$. The number of days of data used for training is N (not to be confused with the look back period L defined earlier in Section 3, and here in this section). Therefore, we may think of the fitted deep learning model (m_N) as the following function:

$$M(d) \equiv m_N(d-1, d-2, ..., d-L) \tag{10}$$

That is, we pass into the function the last L days of data to predict the movement on the next day, or subsequent days. For each day in the L-day history, we may have C columns of data. Thus, the model will require $C \times L$ inputs to generate a binary classification M. For training purposes, N such inputs will be used, i.e., N rows of features, each of size $C \cdot L$. We note that $N \gg L$.

After fitting the deep learning model, we then hold the model fixed and use it to forecast index direction over an out-of-sample forecast period (F), i.e. for days $d, d+1, d+2, \ldots, d+F-1$. We then compute the accuracy of our prediction, i.e., the percentage of days in the forecast period F for which the model was able to correctly predict the sign of movement of the index.

N is the number of observations used to train the model. This is a sliding window, and each forward forecasted period F is modeled on the previous N observations. For our analysis we have considered $N = 10,000$, each forecast period F is predicted based on a training set of the previous $10,000$ days. This means that we train several models on a rolling basis, each with $10,000$ data observations. Each row of these N observations contains $19 \cdot L$ days of history. Even though we use preceding periods' data to predict the index move for the forecast period, our data structure does not require it, as each row in the data set contains the target variable and all the preceding days' data that is needed. However, note that the structure of the feature set retains the time sequence of the data. The advantage is that we may sample data randomly as well from the entire data set when constructing training and validation samples.

L is the lookback period, the number of previous days' percentiles which have been considered in predicting the next day's S&P return. For example if $L = 30$, that means today's return was considered to be dependent on the previous 30 days' percentiles of the underlying stocks. We have considered $L = 30, 60$, and 90 days in our analysis. Finally, F is the forecast period. In our analysis we consider various forward forecast periods of 10 and 30 days. The combination of various values of L and F results give multiple sets of parameter choices for our prediction experiment.

4.1. Prediction Models

We model the direction of returns as binary. We consider any day where the daily return is positive as having a value of 1, and any day when the return was negative or zero is given a value of 0. Our implementation attempts to predict the direction of the movement, rather than the magnitude of the movement itself.

We fit a deep learning neural net (DLNN) model to the data. The model has the following structure, i.e., hyperparameters. The input layer comprises $C \cdot L$ inputs. The number of hidden layers is 3 and each layer contains 200 nodes. The activation function used is Rectifier with Dropout, and Batch Gradient Descent is used with a dropout ratio of 20%, and the crossentropy loss function is used with 50 epochs of training. The model is implemented in the `R` progamming language using the deep learning package from `h2o.ai`.[5] The DLNN is a fully connected, feedforward network with no CNN (convolutional neural net) or RNN (recurrent neural net) features. Standard backpropagation is applied.

We model various scenarios, in each case using $N = 10,000$ observations to train our model and building a model with a 30, 60, and 90 lookback period and a forward forecast of 10 and 30 days. We report the results for the $L = 30$ case, as this is not dissimilar to the $L = 60, 90$ cases. As the prediction period F increases, the accuracy of the model decreases. We report the results for the $F = 30$ days case.

We measure accuracy in two ways:

1. *Forecast period average accuracy*: In this measure we calculate the ratio of the number of days per forecast period that our model gets the index movement correct (i.e, accurately predicted that the index went up when it actually went up and vice versa) to the total number of forecast period observations. Next, we take the ratio of the number of periods that our model was over 50% accurate to the total number of forecast periods. For example, suppose over three forecast periods our model was accurate 60%, 60% and 30% of the time. Then the forecast period average accuracy would be 66.66%, as our model gave a prediction of over 50% accuracy twice in the three forecast periods. This measure gives us a point in time measure of how our model is performing and is labeled as average accuracy in each of the runs discussed below.
2. *Overall accuracy*: In this measure we focus on the daily measures of accuracy. We take individual day level predictions and compare it to the actuals to generate a confusion matrix below. This measure of accuracy gives us a continuous historical measure of the performance of our model.This measure is just the number of correctly predicted days across all days in all experiments. Of course, this accuracy number will be lower than average forecast accuracy.

Figure 4 shows the results for the case when the look back period is 30 days and the forecast period is either 10 or 30 days. In the former we obtain many more non-overlapping rolling periods. This results in 348 non-overlapping forecast periods for $F = 10$, and 115 non-overlapping periods for $F = 30$. Forecast period average accuracy is 64% for $F = 10$ days and 77% for $F = 30$ days,[6] both well over 50% as can be seen from the number of points in Figure 4 that lie above the 0.5 level on the y-axis. The overall accuracy is slightly over 50%, i.e., 53.9% for $F = 10$ days and 52.7% for $F = 30$ days. Given that the percentage of days that the market went up in the sample is 52.7%, this suggests very weak predictability. Hence, using short-term forecasts, we see that markets are efficient when conditioned on the information in large data sets and modern deep learning tools. Even if the forecast period accuracy were exploitable in some way, the levels of accuracy are average but may not be enough (especially after transactions costs) to suggest that market efficiency may not be supported when a larger universe of information is considered in modified weak-form efficiency tests.

---

[5] See http://h2o.ai. We also implemented the model using TensorFlow using Python, and obtained similar results.

[6] Even when we check how many blocks we had accuracy greater than 52.7%, we arrive at the same forecast period average accuracy of 77%.

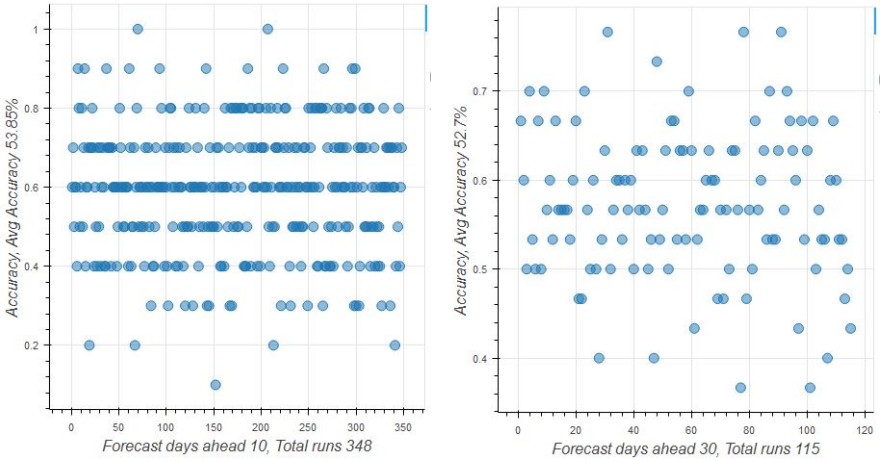

Figure 4. Forecast period average accuracy, where the look back period is $L = 30$ days and the forecast period is $F = 10, 30$ days. Forecast period average accuracy is 64% for $F = 10$ days and 77% for $F = 30$ days, both well over 50% as can be seen from the number of points in Figure 4 that lie above the 0.5 level on the y-axis. The overall accuracy is slightly over 50%, i.e., 53.9% for $F = 10$ days and 52.7% for $F = 30$ days.

4.2. Identifying the sources of market efficiency

Given that our evidence suggests that markets are more or less efficient, we may proceed to use the technology we have to uncover whether the lack of predictability comes from one of two possible sources; (i) random movements in the stock index, or (ii) non-stationarity in the distribution of stock index returns. In this section, we describe a non-parametric approach to ascribing the lack of predictability to each of these sources.

We set the lookback period to $L = 30$ days and the forecast period to $F = 10, 30, 1000$ days. The size of the training data (rows of data) will be $N = 5,000$. We also fix the number of epochs in the fitting exercise to be 50 as before. We compute overall accuracy from the following three experiments.

1. *In-sample*: Using 5,000 observations, starting from the first day of the sample, we train the neural net, and then test it on three sets: (i) a randomly chosen set of 10 observations from the training sample, (ii) a randomly chosen set of 30 observations from the training sample, (iii) a randomly chosen set of 1000 observations from the training sample, and (iv) the entire training sample is treated as the test sample. We then roll forward 20 days and repeat this experiment. This will give us 423 such experiments, each with four accuracy values (one from each test set) for "overall accuracy" and for "forecast period average accuracy."
2. *Stationary out-of-sample*: Here we try to maintain stationarity in the sample by bifurcating the sample into separate training and test groups but from the same sample period using a block of consecutive 5,000 observations. (i) Randomly select 4990 observations for training and keep the remaining 10 for testing; (ii) randomly select 4970 observations for training and keep the remaining 30 for testing; (iii) randomly select 4000 observations for training and keep the remaining 1000 for testing. as in the previous case, roll forward 20 days and repeat this experiment.
3. *Non-stationary out-of-sample*: Starting from the first observation, pick a block of consecutive 5,000 observations, and train the deep learning model. Then, (i) test the model on the next 10 observations; (ii) test the model on the next 30 observations; (iii) test the model on the next 1000 observations. Roll forward 20 days and repeat this experiment.

Each of these experiments gives an "overall accuracy" that is denoted A_i where $i = 1, 2, 3$ for each of the classes above. By comparing these cases, we can determine how much prediction error

Table 1. Accuracy levels for three cases of experiments on stock market predictability. We report the overall accuracy (OA) and forecast period average accuracy ($FPAA$). The first number in each cell is the OA and the second number is the $FPAA$, computed at a threshold of 50%, i.e., $FPAA = 1$ if the percentage of correct forecasts in the prediction period (F) is greater than 0.5, else $FPAA = 0$. We may also compute FPAA for a threshold of 0.527, i.e., the baseline percentage of times the stock market rises, but we get identical results for $F = 10, 30$ and slightly lower values for $F = 1000$.

Experimental case	$F = 10$ (OA, FPAA)	$F = 30$ (OA, FPAA)	$F = 1000$ (OA, FPAA)	$F = 5000$ (OA, FPAA)
A_1. In-sample (IS)	(0.604, 0.662)	(0.565, 0.693)	(0.525, 0.905)	(0.522, 0.929)
A_2. Stationary (OS)	(0.593, 0.624)	(0.551, 0.652)	(0.526, 0.896)	—
A_3. Nonstationary (NS)	(0.594, 0.631)	(0.558, 0.674)	(0.535, 0.888)	—

is supported by randomness versus lack of stationarity. In the following section, we quantify these accuracy values to determine the sources of lack of predictability.

5. Empirical Results

In this section, we denote the three cases as follows: In sample (IS), Stationary out-of-sample (SO), and non-stationary out-of-sample (NS). For each of these cases we report the forecast period average accuracy ($FPAA$) and the overall accuracy (OA). The results are reported in Table 1.

We first examine the four experiments we conducted in the In-sample (IS) case. The overall accuracy declines as we predict larger subsets of the data for testing, albeit in-sample. However, the chance that we do better than a coin toss keeps increasing. Ultimately if the test data set is the same as the training data, the small level of predictability is amplified to deliver very high levels of average accuracy. The histograms of all experiments are shown in Figure 5.

Next, we look at the case where we use out-of-sample values from within the same period, i.e., the Stationary (OS) case. The histograms of all experiments for the OS case are shown in Figure 6. As expected, the accuracy, both OA and $FPAA$, are smaller than for the In-sample case. The difference in accuracy between the IS and OS cases may be attributed to randomness because in both cases the training and the testing sample are drawn from within the same continuous period of time. In the third set of results for the Nonstationary (NS) case, we see almost similar results to the Stationary case. The histograms of all experiments for the NS case are shown in Figure 7.

The patterns in Table 1 show that overall accuracy (OA) declines as the forecast period lengthens for all three cases of forecast experiments. The baseline is the percentage of days for which the market moved up, which for the current data sample is 52.7%. The case of $F = 1000$ days look ahead quickly dissipates to the baseline level expected, as the forecast period is very large, and is to be expected if the market is efficient, which there is evidence of (in [3]). Therefore, it is more interesting to examine how the OA declines as the forecast period is extended from 10 days to 30 days.

First, in all three experimental conditions, overall accuracy declines as we extend the forecast period from 10 to 30 days. This suggests that recent history does matter in prediction, and is also intuitive, as the joint statistical distribution of market stocks is likely to be stationary for small look ahead windows.

Second, we see that the difference in prediction accuracy ($OA \approx 0.6$) between the in-sample case, stationary case, and non-stationary case is small for the case when $F = 10$ days. The suggests that the lack of predictability here ($1 - OA = 0.4$) comes from randomness and not nonstationarity.

Third, for the $F = 30$ days case, the difference between the three cases is also small, but the prediction error has increased to 0.45 (from 0.4 in the $F = 10$ case). We may attribute this to the effect of non-stationarity.

Fourth, we see from the $F = 1000$ days case, that the overall accuracy has dropped back to the baseline of 0.527, as the full effect of nonstationarity and randomness erase all predictability.

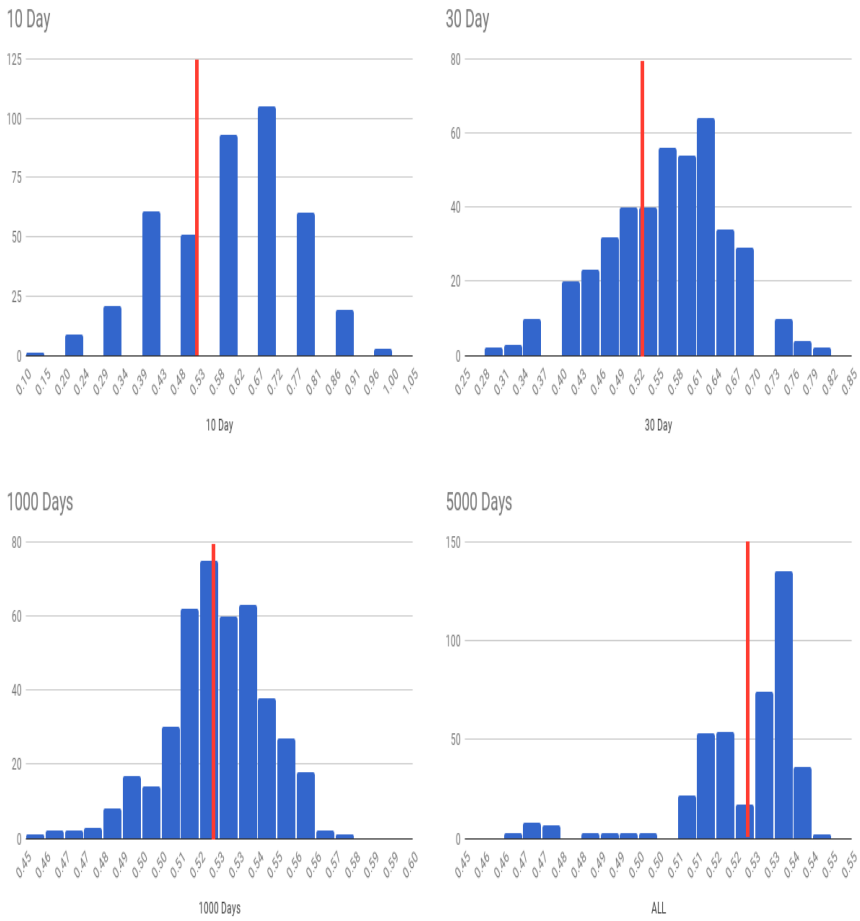

Figure 5. Histograms of the results from all experiments, where the test data was in-sample. The four plots are, reading left to right, top to bottom, the cases when $F = 10, 30, 1000, 5000$ days, respectively. The red line is at the 52.7% accuracy cut off.

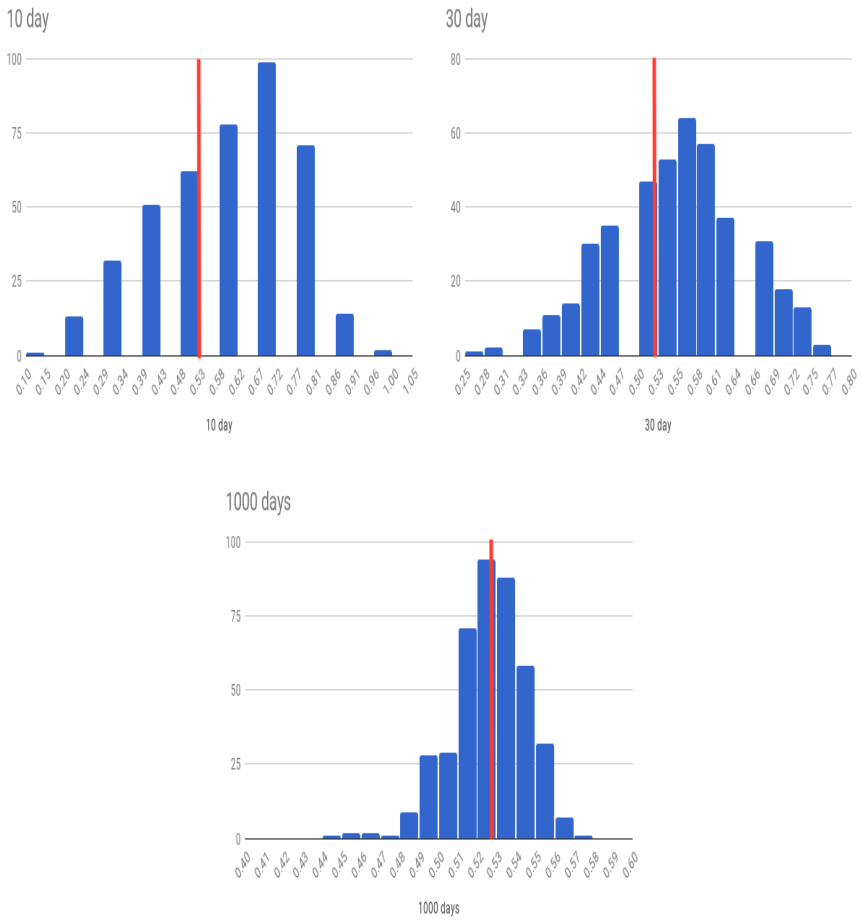

Figure 6. Histograms of the results from all experiments, where the test data was for the Stationary out-of-sample experiment (case *OS*). The three plots are the cases when $F = 10, 30, 1000$ days, respectively. The red line is at the 52.7% accuracy cut off.

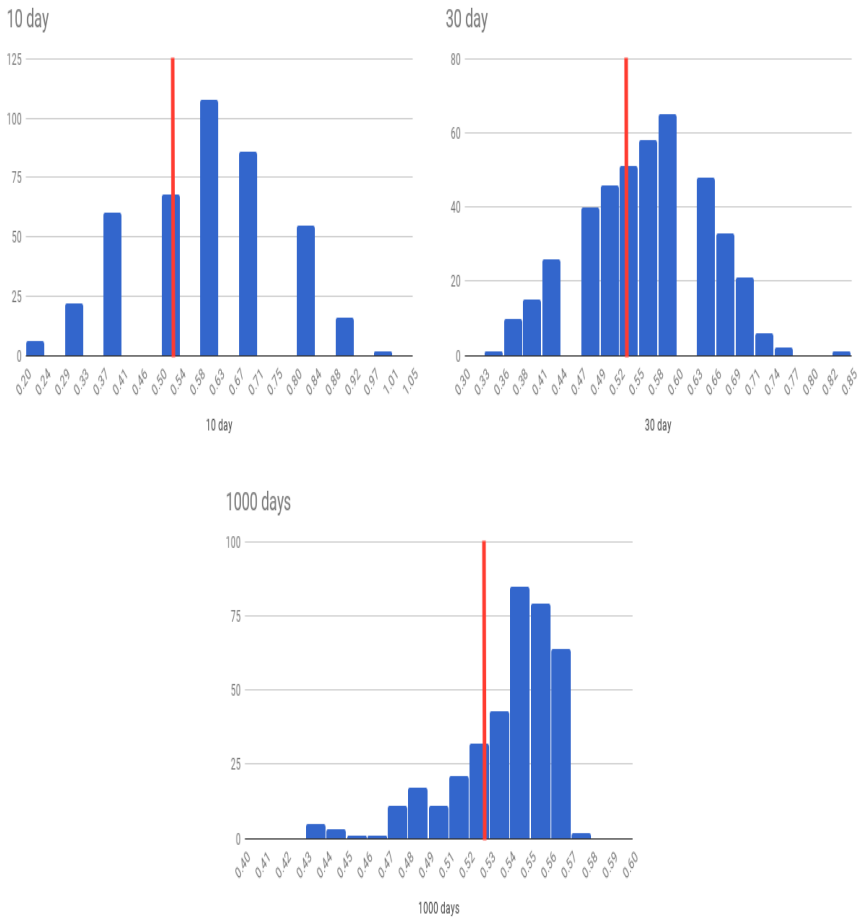

Figure 7. Histograms of the results from all experiments, where the test data was for the Non-stationary out-of-sample experiment (case NS). The three plots are the cases when $F = 10, 30, 1000$ days, respectively. The red line is at the 52.7% accuracy cut off.

Fifth, in both, the $F = 10, 30$ cases, the accuracy level is higher than the baseline percentage number of days for which the market goes up (52.7%). Yet, it is hard to argue that the predictability is sufficient to generate low-risk profit after transactions costs. Hence, these experiments confirm, with large-scale testing, that markets are efficient for an investigation of predictability of the S&P500 index.

6. Concluding Discussion

This paper offers a small experiment to assess the potential of deep learning algorithms to predict markets. Using data on all stocks on the S&P500 index from 1963 to 2016, we train a fully-connected, feed-forward deep learning network with three hidden layers of 200 nodes each to predict the direction of the S&P500 index on a daily basis. Unlike existing tests of weak-form market efficiency, which use a single return series to test for autocorrelation, this new approach expands the information set by a huge order of magnitude, i.e., by using return data from all stocks in the index for the past 30 days. The trained prediction model is then used for look-ahead periods ranging from 10 to 30 days, and more.

Over the entire sample period, the percentage of days for which the S&P index moves up is 52.7%. Therefore, our prediction algorithm needs to predict up moves better than this percentage and not just do better than 50%. (Note that a naive algorithm that predicts markets always move up will be correct 52.7% of the time.) With look-ahead periods of 10 days (out-of-sample), we find that the model gets the predicted movement correct about 59% of the time. For a look-ahead period of 30 days the accuracy is 55%. While this is greater than 52.7%, it is not statistically strong enough to claim that deep learning is able to predict the markets. We may therefore conclude that our tests, which use vastly greater information sets than hitherto used in weak-form tests of market efficiency, does not uncover strong evidence of market inefficiency. However, the shapes of the histograms in Figures 5, 6, and 7 suggest that the model is performing well enough that advanced tuning may lead to better predictability. This is reflected in the FPAA metric shown in Table 1. We close by recommending further exploration into neural net architectures better tailored and tuned for this specific prediction problem.

Acknowledgments: We thank participants for their comments in seminars at the Stockholm School of Economics, TwoSigma, Research Affiliates, CFA San Francisco.

1. Fama, E. The Behavior of Stock Market Prices. *Journal of Business* **1965**, *38*, 34–105.
2. Fama, E. Efficient Capital Markets: II. *Journal of Finance* **1991**, *46*, 1575–1617.
3. Lo, A.; MacKinlay, C. Stock Market Prices do not Follow Random Walks: Evidence from a Simple Specification Test. *Review of Financial Studies* **1988**, *1*, 41–66.
4. Fama, E.; Fisher, L.; Jensen, M.C.; Roll, R. The Adjustment of Stock Prices to New Information. *International Economic Review* **1969**, *10*, 1–21.
5. Elton, E.; Gruber, M.; Das, S.R.; Hlavka, M. Efficiency with Costly Information: A Reinterpretation of Evidence from Managed Portfolios. *Review of Financial Studies* **1993**, *6*, 1–22.
6. McCulloch, W.; Pitts, W. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics* **1943**, *5*, 115–133.
7. Hornik, K. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Networks* **1991**, *4*, 251–257.
8. Telegarsky, M. Benefits of Depth in Neural Networks. *JMLR: Workshop and Conference Proceedings* **2016**, *49*, 1–23.
9. Kelley, H.J. Gradient theory of optimal flight paths. *Ars Journal* **1960**, *30*, 947–954.
10. Bryson, A.E. A gradient method for optimizing multi-stage allocation processes. Proceedings of the Harvard Univ. Symposium on digital computers and their applications, 1961.
11. Dreyfus, S. The computational solution of optimal control problems with time lag. *IEEE Transactions on Automatic Control* **1973**, *18*, 383–385.
12. Rumelhart, D.E.; Hinton, G.E.; Williams, R.J. Learning representations by back-propagating errors. *Nature* **1986**, *323*, 533–536.

13. Goodfellow, I.; Bengio, Y.; Courville, A. *Deep Learning*; MIT Press: Cambridge MA, 2016.
14. Geron, A. *Hands-On Machine Learning with Scikit-Learn and Tensorflow*; O'Reilly Media Inc: Sebastopol CA, 2017.