Financial Market Time Series Prediction with Recurrent Neural Networks

Armando Bernal, Sam Fok, Rohit Pidaparthi

December 14, 2012

Abstract

We used echo state networks, a subclass of recurrent neural networks, to predict stock prices of the S&P 500. Our network outperformed a Kalman filter, predicting more of the higher frequency fluctuations in stock price.

The Challenge of Time Series Prediction

Learning from past history is a fudamentality ill-posed. A model may fit past data well but not perform well when presented with new inputs. With recurrent neural networks (RNNs), we leverage the modeling abilities of neural networks (NNs) for time series forecastings. Feedforward NNs have done well in classification tasks such as handwriting recognition, however in dynamical environments, we need techniques that account for history. In RNNs, signals passing through recurrent connections constitute an effective memory for the network, which can then use information in memory to better predict future time series values.

Unfortunately, RNNs are difficult to train. Traditional techniques used with feedforward NNs such as backpropagation fail to yield acceptable performance. However, subsets of RNNs that are more amenable to training have been developed in the emerging field known as reservoir computing. In reservoir computing, the recurrent connections of the network are viewed as a fixed reservoir used to map inputs into a high dimensional, dynamical space–a similar idea to the support vector machine. With a sufficiently high dimensional space, a simple linear decode can be used to approximate any function varying with time.

Two reservoir networks known as Echo State Networks (ESNs) and Liquid State Machines (LSMs) have met with success in modeling nonlinear dynamical systems [2, 4]. We focus on the former, ESN, in this project and use it to predict stock prices and compare its performance to a Kalman filter. In an ESN, only the output weights are trained (see Figure 1).

Echo State Network Implementation

The state vector, $\mathbf{x}(t)$, of the network is governed by

$$\mathbf{x}(t+1) \quad = \quad f\left(W^{in}\mathbf{u}(t) + W\mathbf{x}(t) + W^{fb}\mathbf{y}(t)\right), \tag{1}$$

where $f(\cdot) = \tanh(\cdot)$, W^{in} describes the weights connecting the inputs to the network, $\mathbf{u}(t)$ is the input vector, W describes the recurrent weights, W^{fb} describes the feedback weights connecting the outputs back to the network, and $\mathbf{y}(t)$ are the outputs. The output $y(t)$ is governed by

$$\mathbf{y}(t) \quad = \quad W^{out}\mathbf{z}(t),$$

where $\mathbf{z}(t) = [\mathbf{x}(t), \mathbf{u}(t)]$ is the extended state. By including the input vector, the extended state allows the network to use a linear combination of the inputs in addition to the state to form the output.

ESN creation follows the procedure outlined in [3]. Briefly,

1. Initialize network of N reservoir units with random W^{in}, W, and W^{fb}.

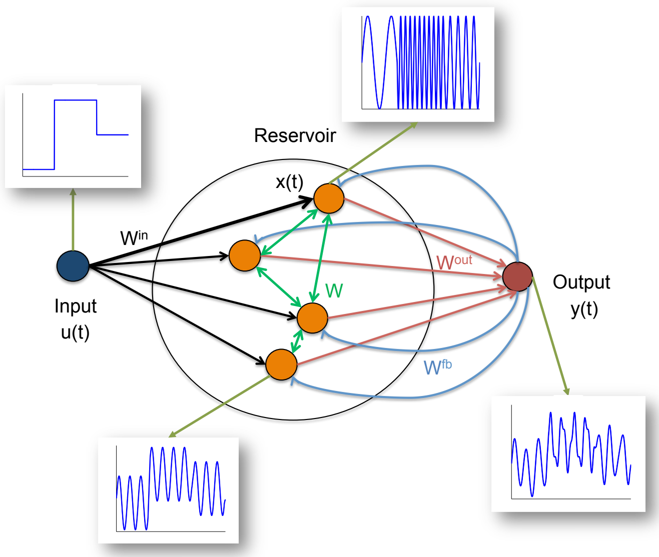

Figure 1: The ESN consists of inputs units (dark blue), reservoir units (orange), and output units (red). The input weights W^{in}, recurrent weights W, and feedback weights W^{fb} are fixed. Only the output weights W^{out} are trained. Through recurrent connections and nonlinear activation functions of the reservoir units, the ESN can model arbitrary, dynamical systems with a simple linear decode.

2. Sparsify W by setting *sp* fraction of the entries to 0.
3. Set $W := \frac{\alpha}{\sigma_{max}(W)} W$ so spectral radius of W is α where $|\alpha| < 1$.
4. Train network:
   (a) Input training sequence and assign output $\mathbf{y}(t)$ to take on the value of the training output.
   (b) Record extended state $\mathbf{z}$ over training period.
   (c) Find W^{out} to minimize $\|\mathbf{y}_{train} - W^{out}\mathbf{z}\|_2^2$ over the training period.

Setting the spectral radius to less than 1 ensures that any state of the network eventually decays to 0, which preserves the stability of the network.

Data, Feature, and Parameter Selection

We sourced our data from Yahoo! Finance. We collected daily stock prices dating back from late 2004 to early 2009. Our feature vector included the current and 5-day history of the stock price, the 5, 10, 15, and 20 day moving averages, volume, and the S&P500 index. These features are typically used in graph analysis to buy, hold, or sell a stock. We only used features updated daily so did not include indicators such as the Price-to-Earnings, which is released quarterly. The S&P500 index was chosen as a feature since the index represents the 500 biggest companies in the largest economy in the world. The stock volume was used as an indicator of the activity of the stock.

In our ESN implementation, $N = 100$, $\alpha = 0.8$, and $sp = 0.05$. We wanted to gauge how many training samples we need to train. We found that flushing out the initial network state by driving the network with training input for a period before recording the extended network state for training the output weights improved the network performance (see Figure 2).

We examined how the testing error behaved with respect to the number of training examples (see Figure 3). Error did not decrease monotonically with number of training examples. A possible explanation for this could be that during the high test error training periods, the stock was subject to different perturbations than during testing.

To compare the contribution of each feature to the model, we performed a leave-one-out cross-validation analysis to check which features result in the lowest error (see Figure 4). The ESN performed worst when the price was not included as a feature, and the ESN network performed its best when the feature set included the current price, trading volume, and the S&P500 price.

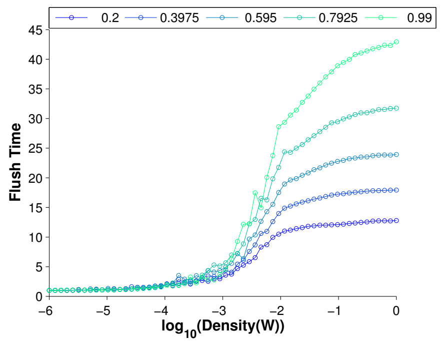

Figure 2: Flush time for the $\|\mathbf{x}\|_2$ to decay to less than 10^{-12} from initial random state as a function of density (fraction of nonzero entries) of W, and spectral radius (legend). Each data point averaged from simulation of 300 random initial states. For the parameters of our ESN, we found that about 25 time steps was necessary to flush out the initial state.

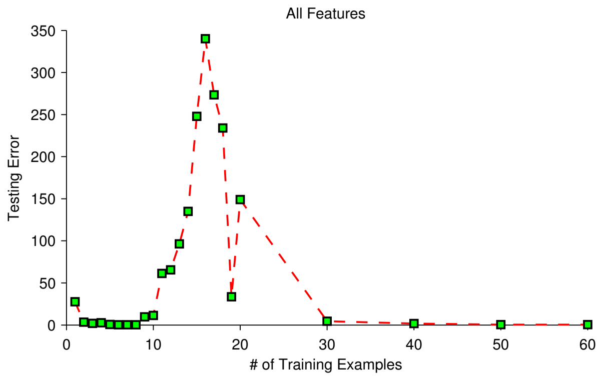

Figure 3: Performance of ESN Using all 12 features on a test set of 500 days as duration of training is varied. Test error is defined as $\frac{(y-y_{target})^2}{\sigma_{y^2_{target}}}$.

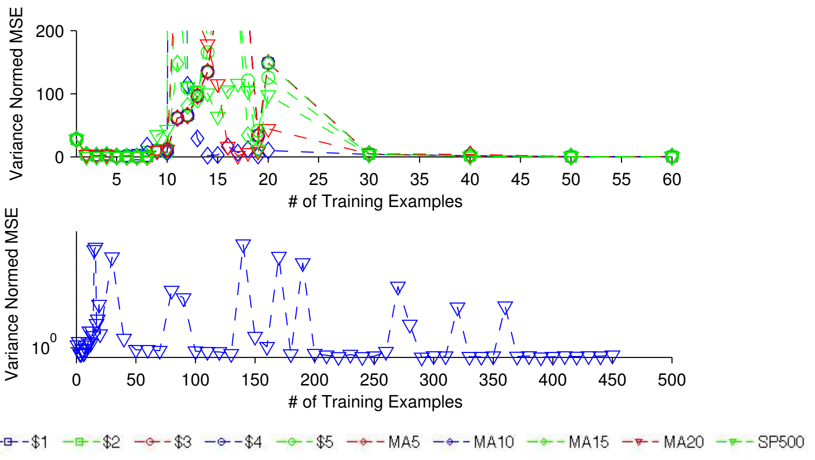

Figure 4: Top: leave-one-out analysis for all features exluding volume. Bottom: analysis for volume. Note the magnitude of error introduced when leaving out volume.

Kalman Filter Performance Comparison

As a baseline to compare ESN performance, we implemented a Kalman filter. The Kalman filter uses an underlying linear dynamical model recursively estimates the state of a process by reducing the mean squared error. We used a simple model to predict the next day's price. We modeled the system by the following:

$$x_{k+1} = x_k + \varepsilon \tag{2}$$

$$y_k = x_k + \tau \tag{3}$$

where x_k represents the price at time k and y_k represents the observation made at time k. The process and measurement noise ε and τ are assumed to be zero mean Gaussian random variables where the variance were measured using the historical data. Figure 5 compares the filter with the ESN.

Results

ESN was tested on Google's stock price in comparison to the Kalman filter (see Figure 5). The ESN captures quick changes in the stock price whereas the simple Kalman filter cannot.

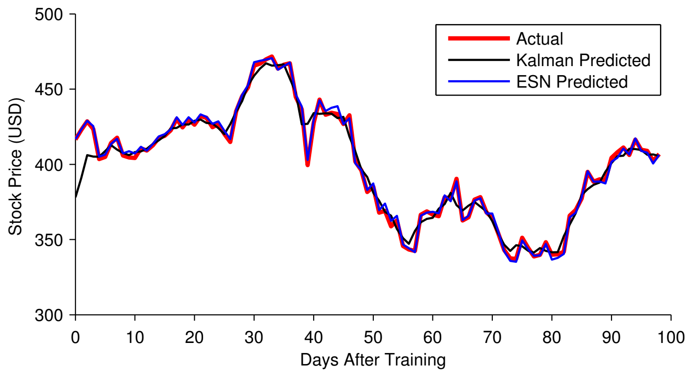

Figure 5: Google stock price prediction for ESN and Kalman filter. ESN predicts rapid changes in stock price wheras Kalman filter tends to smooth rapid changes. Test error, $\frac{(y - y_{target})^2}{\sigma_{y^2_{target}}}$, for the ESN is 0.0027. In contrast, test error for the Kalman filter is 0.2135.

The ESN was also tested using 50 randomly selected stocks from the S&P500 to see if it generalize well to other stocks (6). It is interesting to note that we don't see an increase in error as we forecast more and more into the future. However, it makes sense that the error does not increase over time so long as the underlying dynamics of the stock during the testing period is the same as during the training period. If there were a major variation in our testing period not accounted for in our training period than we would expect big errors.

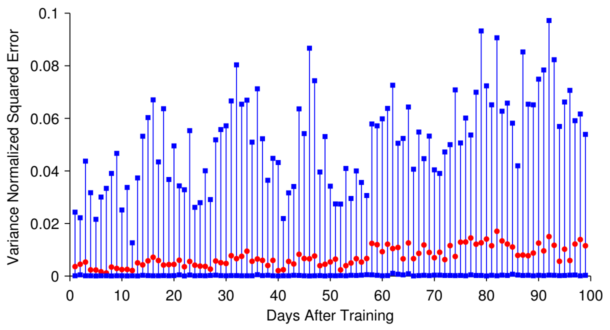

Figure 6: ESN performance on 50 randomly selected stocks from the S&P 500. Red marks indicate median and blue bars extend between the 10-90 percentiles of the data.

Discussion

Time series analysis gained attention when George Box and Gwilym Jenkins published their ARMA and ARIMA models for time series forecasting [1]. ESNs provide a powerful black box method for modeling time dependent phenomena. Black box methods are appealing because they require no prior assumptions or knowledge of the underlying dynamics of the time series data as with the Kalman filter. The Kalman filter does not have enough features to predict prices and capture rapid movement in the stock price. One would have to build a more complex model, but in that case one needs to estimate the state matrix, i.e. one needs to understand the dynamics between future prices and past historical data.

The power of reservoir computing and the relative dearth of theory and understanding indicates that reservoir computing is a rich field opportunity for study. When we predicted the future prices of 50 stocks with our ESN we also observed higher error in periods with more volatility shown by a higher volume of trading. This was likely exacerbated by the fact that we trained and tested on the same time periods for all the stocks which means that our training set may not have captured the phenomena that occurred in the market during the time of our testing set. In order to mitigate this, we would like to train and test the 50 stocks at different time periods so that all of them do not suffer from volatility at the same time which makes our predictions inaccurate.

Due to time contstraints, we were only able to play with a few of the parameters of the ESN. In our future research we would like to explore what the highest frequency signal we can match with our ESN. According to [2, 3][2, 3] this involves a complex interplay between the spectral radius of W and the number of reservoir neurons in our network. Additionally we would like to explore how changing the feedback weights $W^{fb}$and the input weights W^{in}affects the network.

References

[1] G. E. P. Box, G. M. Jenkins, and G. C. Reinsel. *Time series analysis: forecasting and control*, volume 734. Wiley, 2011.

[2] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Tecnical report GMD report*, 148, 2001.

[3] H. Jaeger. *Tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the" echo state network" approach.* GMD-Forschungszentrum Informationstechnik, 2002.

[4] Wolfgang Maass, Thomas Natschlager, and Henry Markram. Real-time computing without stable states: A new framework for neural computation based on perturbations. *Neural Computation*, 14(11):2531–2560, November 2002.