CS394N Final Project: Optimizing inputs for Stock Market Prediction

Woody Austin and Joel Iventosch

December 15, 2014

Abstract

Stock market prediction a highly sought after goal for many for obvious reasons. This paper proposes a hybrid statistical-connectionist-evolutionary model to predict the daily closing stock prices of a given set of companies. The proposed model uses principal component analysis (PCA) to optimize time-series input data and a genetic algorithm (GA) to evolve the optimal set of external input features (i.e. data other than the time-series data). Both sets of input data are fed into a recurrent neural network (RNN) trained with a variant of the backpropagation learning algorithm. We test our model with 503 days of data collected for the 30 companies in the Dow Jones Industrial Average (DJIA) Index. Initial results show that while PCA does improve the predictive power of the model, most of external input features that were considered do not improve the performance of the model.

1 Introduction

In today's increasingly global economy, the stock market is used as a money-making vehicle for people across the globe. The world of finance is becoming ever-more interconnected as technology enables people to participate in the global marketplace. Nearly ubiquitous internet access in developed nations has revolutionized access to the stock market.

In light of all this, the task of stock market prediction is an exciting and relevant problem to study. While much previous research has been done on this topic, no perfect solution to the prediction problem has yet been published [5]. Techniques drawing inspiration from traditional time-series prediction, machine learning, and statistics have been proposed, yielding varied results. Statistical methods, symbolic methods, and numerous others including many hybrid methodologies have been applied to the problem [8] [3]. Neural networks have also been widely implemented for the task, and this is where we choose to start. Supervised neural networks are a particularly good fit for stock market prediction given the plethora of pre-labeled training data available (i.e. historical stock market data) and their inherent ability to learn nonlinear and chaotic patterns in very large data sets.

We look at two particular approaches to solving the problem that both center around optimizing input features to the network. We attempt to gain more information from time-series data by performing principal component analysis (PCA) on it as well as evolve a set of non-time-series data features as external inputs to the network using a genetic algorithm (GA). Networks using external features are often referred to as hybridized models in the literature, while networks whose inputs are statistically optimized are referred to as hybrid models [1] [3]. As such, our model is a hybridized hybrid model, which is a strange moniker. We will use the one of the two terms interchangeably to describe our model for the remainder of the paper.

The remainder of this paper will be organized as follows: Section 2 will introduce some previously established knowledge and knowledge gaps in the applied field, as well as discuss our motivation for this research; Section 3 will detail the specifics of our

model and approach; Section 4 will review the results of our experiments; Section 5 will discuss the significance of our results and potential future work; Section 6 will conclude.

2 Background and Motivation

Due to the large amount of previous work done in this domain, much is already known about using neural networks and hybridized models for stock market prediction. It has been widely demonstrated that basic feedforward multi-perceptron neural networks and recurrent neural networks perform well for stock market and general time-series prediction [5] [3]. The literature also reveals that backpropagation is one of the most successful, and most widely implemented, learning algorithms in time-series applications. In fact, it is estimated that over 80% of all financial time-series prediction models use some form of backpropagation as their primary learning algorithm [4]. We use the resilient backpropagation algorithm RProp- defined in [7]. Ayodele et al. demonstrated that including additional, related input data (e.g. data on corporate fundamentals or corporate sentiment) can also lead to improved performance [1]. Despite the extensive research on the topic, room for improvement still remains.

Four previous works inspired the design of our project. First, the use of PCA in time-series prediction has been applied to temporal link prediction [2]. We use the basic principles drawn from the part of that paper concerning two-dimensional inputs as inspiration for our preprocessing of inputs. The work of Ayodele et al. using neural networks with additional external input data (e.g. corporate fundamentals data) to predict stock prices [1] also informed our project. We used ideas from Taghi et al. and their work using a design of experiments to determine the optimal set of influential financial and economic variables to use as input features for their neural network stock index prediction model [6]. Finally, Mansour Sheikhan and Behzad Movaghar's work using neuroevolution to evolve the optimal network architecture and parameters for their evolutionary-connectionist model to predict foreign exchange rates is the last paper that we drew upon [8].

To evaluate the effectiveness of our hybrid model we collected 503 days of time-series and external input data for the 30 companies in the Dow Jones Industrial Average Index (DJIA), from November 12th, 2012 to November 10th, 2014 (excludes weekends and stock market holidays). We then ran a series of tests in which we trained and tested our recurrent neural network (RNN) without the use of PCA or evolution in order to establish baselines to compare our full-fledged hybrid model's results. To test our full model we ran 18 different evolutionary experiments in which we varied parameters of both the RNN and the GA. Our results showed that the best-performing network utilized PCA, but used very few of the 378 possible external input features.

3 Approach

3.1 Defining the Problem

The stock market is very complex by nature, with many different factors affecting the performance of a particular company's stock price on any given day. While traditional time-series forecasting simply uses the historical stock price data for a given company, more sophisticated models will often consider a variety of external factors that might affect the performance of a stock.

We aim to optimize both time-series and external data inputs to maximize predictive power. We have selected the 30 companies in the DJIA as the test set for our model.

As Figure 1 indicates, we have classified our input data into two general categories: time-series and external data. The time-series data refers to the daily closing stock price of each company in the set of companies being analyzed by the model (the 30 companies in the DJIA in this case). This type of data is also referred to as micro-technical analysis data [1] [5]. The external data refers to data that is *not* historical stock data (i.e. technical analysis data) for any of the 30 companies. To provide more specific classification, the external data is further broken down into five sub-categories: micro-fundamentals,

micro-sentiment, macro-technical analysis, macro-fundamentals, and macro-sentiment. Each type of "micro" input data is related to a particular company, while each type of "macro" data is derived from the economic landscape as a whole. Given n days, for each company we have n time-series inputs, $10n$ micro-fundamental inputs, and $2n$ micro-sentiment inputs. We also have 6 macro-technical inputs, 10 macro-fundamental inputs, and 2 macro-sentiment inputs. Given that we have 30 companies, this yields 408 inputs per day, 30 of which we will consider ***time-series data*** and the remaining 378 we will refer to as ***external data***.

3.2 Hypothesis

By using PCA to optimize the time-series input data and a genetic algorithm to evolve the optimal subset of selected external input data, we can produce a RNN model that predicts the next day closing stock price of each of the 30 companies in the DJIA more accurately than without these input optimizations.

3.3 Methodology: RNN + PCA + Evolution

We use PCA and a GA to optimize the input features of the RNN seen in Figure 2. Given the obvious importance of the time-series data to the forecasting mechanism, our model always uses the time-series data as inputs when testing with the GA. Since the relative importance of each of the external data inputs is largely unknown, the genetic algorithm is used to evolve the optimal subset of possible external inputs. The following subsections discuss each of the key components of our methodology in further detail.

3.4 Recurrent Nueral Network

The network used in our model is a four layer RNN with two hidden layers, as shown in Figure 2. The first level of the network is a linear layer and is split into two distinct input layers – one corresponding to time-series input variables (Time-Series Input Layer

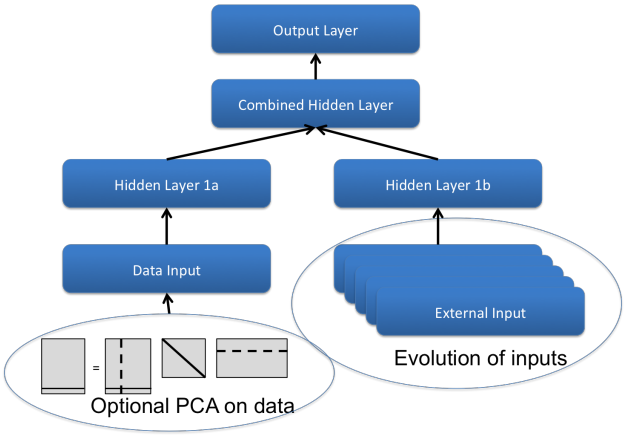

Figure 2: Design of the proposed hybrid Statistical-Connectionist-Evolutionary model. You can see the separate input and hidden layers for each type of data in the figure as well as the combined hidden layer and output layer. On the left branch, there is an optional data preprocessing step that uses PCA and on the right we use a GA to evolve to correct feature set.

in Figure 2), and another corresponding to the external input features selected by the GA for inclusion in the network (External Data Input Layer in Figure 2).

The second level of the network is also a split layer – one layer corresponding to each of the input layers (Time-Series Hidden Layer and External Data Hidden Layer in Figure 2). Both second-level layers are hidden nonlinear layers with sigmoidal activation functions. Each input layer is fully connected to its corresponding second-level layer – but completely independent (i.e. not at all connected) of the opposite second layer. Both second-level layers are also designed with fully-connected recurrent connections to provide the model with memory.

The Time-Series Hidden Layer contains six nodes, while the External Data Hidden Layer contains a variable number of nodes, ranging from 3 at the minimum to a maximum of 30. The number of nodes in the External Data Hidden Layer is determined by the following equation: $min(30, max(3, (ext\_inputs/5)))$ where ext_inputs = the number of external data inputs selected by the GA. While this might appear somewhat arbitrary, our reasoning behind using this

	Input Data					
Category:	Time Series Data	External Data				
Subcategory:	Micro-Technical	Micro-Fundamentals	Micro-Sentiment	Macro-Technical	Macro-Fundamentals	Macro-Sentiment
Input Variables	1. Daily closing price of each of 30 companies in the DJIA	1. Basic Shares 2. Net Income 3. Earning per Share (EPS) 4. Net Profit Margin 5. Return on Equity (ROE) 6. Return on Assets (ROA) 7. Price to Book ratio (PB) 8. Dividend per Share (DPS) 9. Dividend Yield (DPS/Price) 10. Price to Earnings (PE) (Price*Shares/NetIncome) 11. PE Ratio (Price/EPS)	1. Google Trends Rankings	1. NASDAQ Composite 2. S&P 500 3. Global Dow 4. S&P Global 1200 5. S&P 500 Volatility 6. NASDAQ 100 Volatility	1. Crude Oil 2. Gold 3. Silver 4. US 10 yr Note 5. US Federal Funds Rate 6. US GDP Growth Rate 7. US Inflation Rate 8. US Labor Participation Rate 9. US Unemployment Rt 10. Housing Starts	1. Consumer Sentiment 2. Retail Sales MoM

Figure 1: Breakdown of Input Data

range was to avoid an explosion of training time given the fully connected nature of the layer. Furthermore, we wanted to ensure there were at least 3 nodes to allow the network to perform learning at that layer.

The outputs of both second-level hidden layers feed into a single combined third layer (Combined Hidden Layer in Figure 2), which is also a nonlinear layer utilizing a sigmoidal activation function. All second-level outputs are fully connected with the inputs of the Combined Hidden Layer. The third layer contains 10 nodes and is fully connected with the output layer (Output Layer in Figure 2). The output layer is a linear layer, containing 30 nodes, which corresponds to one node for every company in the set of companies under analysis (the 30 companies in the DJIA in this case).

The network parameters for our RNN were selected initially and then held constant throughout each experiment. Originally, we wanted to perform each set of GA experiments with three learning rates: 0.1, 0.25, and 0.5. Unfortunately, Condor cluster problems constrained our ability to do so. We have run the full set of GA experiments with a learning rate of 0.1 both with and without PCA, and the same set only with PCA at a learning rate of 0.25. Our initial experiments without GA showed that learning rate decay did not perform any better than no decay, so we did not vary this in our GA experiments. We discuss this further in Section 5.

3.5 Principal Component Analysis

Principal component analysis (PCA) is an important statistical technique used in many machine learning applications. PCA finds any underlying structure present in the data being used by performing linear transformations upon it. These tranformations arrange the data such that the first principal component is aligned along the axis of most variance in the data, the second is aligned orthogonally to the first along the axis of second-most variance, and so on for each component. Each principal component (in two-dimensions) is a rank-one matrix that is scaled by a singular value. In fact, PCA is equivalent to the Singular Value Decomposition (SVD) [9].

For each company, we gather the closing value for each day from November 12, 2012 through November 10, 2014. We use thirty companies to form our time-series input, $T \in \mathbb{R}^{t \times c}$ where c is the number of companies and t is the number of days.

Each of these datasets are affected by the same macro-technical inputs discussed in Section 3.1 and general trends in the market. Therefore the companies' time-series data should be highly correlated. To exploit this correlation, we perform the truncated-SVD on T to get matrices, U, Σ, and V^T where $U, V \in \mathbb{R}^{t \times c}$ are orthonormal and $\Sigma \in \mathbb{R}^{c \times c}$ is a diagonal matrix of the singular values of T.

Originally, we thought that using a low-rank approximation to the data by taking the first r columns

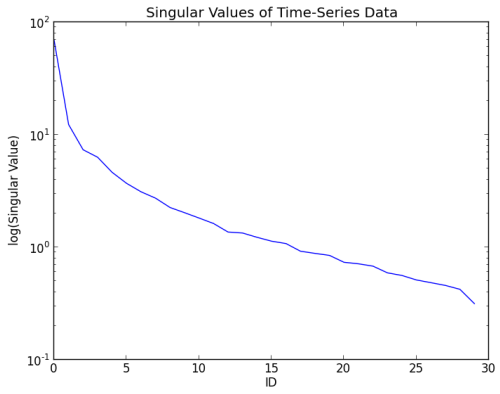

Figure 3: Singular values of T. Note that even in the log-scale only the last singular value even appears like it might be after a corner. Our quick algorithm did not find any obvious cutoff point.

of U would allow us to gain enough information and allow our network to converge more quickly by having fewer inputs. Upon analysis of the time-series data for the companies and times given, we found that the smallest singular-value was still large and no clear drop-off was observed. We ran a quick-and-dirty corner finding algorithm and only found the first obvious corner corresponding to the drop after the first singular value. This can be seen in Figure 3.

To use PCA in practice, we perform time-series predictions on U in the same manner as we would on T. Note that since we are not using a low-rank approximation, U has the same dimensions as T. If we had used such an approximation, U would have the same number of rows, but fewer columns. We use U as the input to our neural network to obtain predictions for future rows of U and obtain U'. To obtain actual predictions for companies, we recombine U', Σ, and V such that $T' = U'\Sigma V^T$. The resulting T' will have as many rows as U' and will therefore have as many time-step predictions as there are extra rows compared to T.

3.6 Evolution

Evolutionary algorithms are useful in situations where the error surface of the function being optimized is highly erratic and mostly unknown. Such characteristics are exemplified by the stochastic and chaotic nature of the stock market. Such algorithms allow for quick search that can avoid local optima much better than traditional search methods [8]. Another advantage of evolutionary and genetic algorithms is that they are highly parallelizable, lending themselves to problems with large search spaces. Given the extremeley large number of possible external data inputs that our proposed model considers, evolution was a natural fit for determining the optimal subset of external input features.

Our model considers 30 different types of external input data – 12 of which are considered "micro" and 18 of which are considered "macro" in scope (see Figure 1). This yields 378 possible external inputs since each of the 12 "micro" external inputs is multiplied by n different companies. Because our evolutionary process considers a micro-external input to be either **activated or not for *all* companies** in the DJIA, our search space comprises only 30 external inputs – not all 378. There are still 2^{30} possible selections; we leave the exploration of the full 2^{378} to future work. Given the significant computational expense (and time) it would take to run even the 2^{30} different RNN experiements, we elect to use evolution to search for the optimal subset of external input features.

We use a genetic algorithm (GA) to evolve a population of RNNs, in which the network architecture and parameters are held constant (within each evolutionary experiment), while the external data feature set is optimized. Each indivdual within the starting population is initialized with a randomly generated binary phenotype, encoding the subset of external input features to be passed along to its corresponding genotype.

Each genotype is thus the second of the two input layers to our RNN described in Section 3.4 as specified by its phenotype. Every RNN in the population is then trained on a set of labeled training data using the RProp- learning algorithm, and then tested on a

set of validation data. The RNN's mean squared error (MSE) on the validaiton data is then returned to the GA and used a the measure of that individual's fitness. Thus, the performance of a given RNN can be viewed as our GA's fitness function. Once each individual in the population has been evaluated (in parellel), the current population is used to spawn the memebers of the next population.

Our GA uses a combination of crossover, mutation, and random regeneration to produce the population for the next generation. More specifically, the next generation is defined as follows:

- elite children = 10-20%
- mutation children = 20-40%
- crossover children = 40-50%
- random children = 10%

Elite children are the top performing 10-20% of the inidividuals from the prevous generation, and they are passed along to the next generation without any alteration. *Mutation children* are derived from the next performance echelon in the previous population. Mutation children do not undergo any crossover, but are mutated at a much higher rate than crossover children. *Crossover children* are produced based on the "top percent (x,y)" selection operator and the "uniform" crossover operator. In our implementation of top percent selection, *Parent A* is randomly selected from the top x% of the previous generation and *Parent B* is randomly selected from the top y%, where $x \geq y$ and $x, y \geq$ top 50% (parameters were varied throughout different experiments; see Table 1). Parent A and B then perform uniform crossover, in which their offspring has an equal chance of inheriting each gene from either parent. Each set of parents produces one offspring, and then a new set of parents is selected in the same fashion to produce another offpsring. This process is repeated until all designated crossover children have been produced. Once crossover is complete, all crossover children are subject to the chance of mutation. Lastly, each generation includes 10% randomly generated *random children* to maintain diversity within the population.

We implement a "random-random" mutation strategey, in which the mutation children and crossover children are randomly selected for mutation (at different rates, however). If selected, an indivudual's genes are mutated uniform randomly.

Our GA was designed to promote efficient convergence, while maintaing enough diversity – particularly early in the search – to avoid convergence on local minimums. Table 1 provides further detail on the operators and parameters used in our GA.

GA Parameter	**Value(s)**
Population Composition:	**Opt1 Opt2 Opt3**
Elite Children (E.C.) %	10% | 20% | 10%
Random Children (R.C.) %	10% | 10% | 10%
Mutation Children (M.C.) %	40% | 20% | 30%
Crossover Children (C.C.) %	40% | 50% | 50%
	100% 100% 100%
Population Size	20
Generations	30
Top Percent: $\boldsymbol{x}$%	10% or 20%
Top Percent: $\boldsymbol{y}$%	25% or 50%
Parent A inheritance prob.	50%
Parent B inheritance prob.	50%
E.C. mutation rate	0%
R.C. mutation rate	0%
M.C. mutation rate	20% or 5%
C.C. mutation rate	10% or 2%
Gene-level mutation rate	15%

Table 1: GA parameters for our experiments. In the top portion of the taple, the parameters were varied together along options 1, 2, and 3.

4 Results and Discussion

Overall, our results only partially support our hypothesis in that PCA is better than not, but external features do not seem to offer a benefit to the prediction.

We first ran several tests to explore the best kind of data normalization for the time-series inputs. After trying entrywise l2, column-wise l2, min-max, and log min-max normalization schemes, we settled on min-max normalization. Many of the external inputs had

similar formats to the time-series data and we also used min-max on those. One thing that we could have improved was trying different normalization schemes for this data as well.

We report two sets of results, those that use the GA and those that do not. For those that do not we varied learning rate among 0.1, 0.25, and 0.5, learning decay rate among 1.0 and 0.88, chose whether or not to use (all of the) external data, and chose whether or not to use PCA. This lead to 24 experiments. We only split the data into training and testing sets for these tests. The lowest mean squared errors (MSEs) among these experiments all occured when using no external data and also using PCA. This *was* statistically significant. The second best set of experiments also included no external data. Overall, using learning rates of 0.1 and 0.5 were statistically better than learning rates of 0.25. No learning decay beat out learning decay in both the 0.1 and the 0.5 categories. These were all verified with Welch's t-tests. This lead us to believe that much of the external data would not be useful, but hoped that the GA would be able to salvage some of it. We took the parameters learned from these preliminary experiments and applied them to our GA trials.

For the GA runs, we split the data into three sets, a training set, a validation set, and a testing set. We did this because the GA needed to use the information from the validation set in order to tune the model and we did not want to contaminate our results involving the predictive power of the model. We set up three GA parameter options shown in Table 1. We planned to vary learning rate among 0.1, 0.25, and 0.5, but issues with getting Condor to launch tasks and then come out of hibernation when they were done set us back further than expected. As a result, we only have the full set of tests for a learning rate of 0.1, although we also ran tests with PCA and a learning rate of 0.5.

Preliminary experiments indicate that the best performing RNNs included few or no external input features. Closer review reveals that in the 18 evolutionary experiments we performed, only 7 of the 12 micro-fundamental external input features were ever included in the top performing networks. These were Return on Assets (ROA), Return on Equity (ROE), Price to Book ratio (PB ratio), Dividend Yield, Earnings per Share (EPS), Net Income, and Number of Shares. Among these 7, only 2 were included in more than two top finishing networks – ROE and ROA, both of which were included in 3 times. PB ratio, Dividend Yield, and Number of Shares each made 2 appearances in top performing networks. EPS and Net Income were selected once each. It is important to note that Dividend Yield and EPS both change daily, and thus have periodicities aligned with the output data. Additionally both EPS and Dividend Yield are metrics that intrinsically take into account the daily closing price of a company's stock. As such, we have to question whether it was the metric as a whole that the RNN garnered value from, or if the network was simply extracting the "time-series" element from these two metrics.

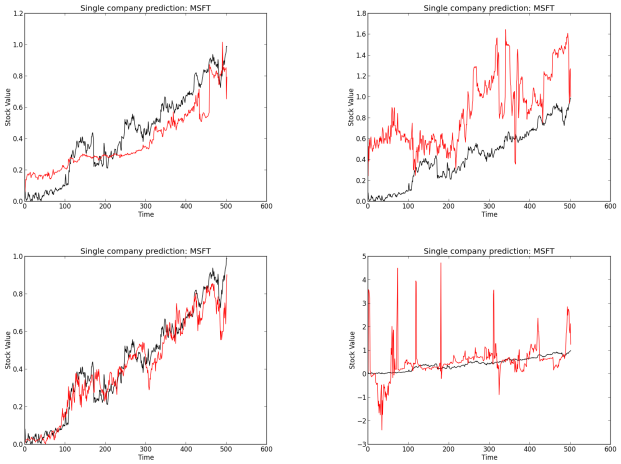

Figure 4: Comparison of multiple options on the same company's predictions. The **top-left** figure shows predictions generated by a GA run's phenotype with no PCA. The **top-right** figure shows predictions generated by a GA run's phenotype with PCA that has *not* been fixed. The **bottom-left** figure shows predictions without external data, for fixed PCA. The **bottom-right** figure shows predictions for all external data and no PCA.

Two addition points are worth noting: (1) ROA, ROE, Net Income, and Number of Shares all have periodicities misaligned with both the time-series data

and output predictions. Furthermore, none of these metrics instrically accounts for the stock price. (2) ROA and ROE are very similar metrics and were the two external input features that appeared the most in top performing networks. Both of these points strengthen the argument that these external input variables may have some relevance and add some value to an adjusted model – particularly a model in which the predicted output values are monthly or quarterly predictions instead of daily ones. See Section 5 for more on this potential future work.

Although our results do suggest that several of the external input features might strengthen the predictive power of the model, most of the external inputs proved to be of no value – or even weaken the results. We hypothesize that this may be due in large part to the varying and misaligned periodicities of much of the external input data.

The networks that were solved without PCA involved performed better at predicting the test data at first. It is interesting to note that the network predictions for single companies using PCA in the original manner all predict a similar shape that is only perturbed by a small amount between the companies. This shape varies greatly between various network runs and phenotype configurations but remains constant among the companies otherwise. This led us to believe that the largest principal component is always picked up, but is made much more dominant by the scaling factor of the singular values when recombined. This was a large lapse of judgement in our experimental design. We should have absorbed the scalings of the singular values into our U matrix (Section 3.5) before passing it into the network as input. After quickly re-running a few of the experiments at the last minute, this mini-hypothesis was verified and PCA networks did *appear* to perform better on average than the non-PCA networks. This was not verified by rigorous statistical tests.

In the original case, our PCA trials acheived significantly better MSE values for the given training and validation sets, but much worse values once converted back to the full representation. Had we performed a full data analysis earlier rather than taking these MSE values of the non-reassembled PCA predictions as proof of concept, we might have had time to fix the issue in all of our tests and include the reassembly in the GA process. A representative selection of predictions can be seen in Figure 4.

A discussion of training error and number of iterations follows as part of the next section.

5 Future Work

Varying Network Parameters:
We suggest running experiments with additional learning rate values, as well as varying the use of learning rate decay. Future work should also include experimentation with other network parameters such as momentum and network topology. One could even evolve the features themselves through the GA.

Varying GA Parameters:
Future work should include many additional evolutionary experiments to allow for the manipulation and optimization of the numerous GA parameters. See Table 1 for details on which parameters were varied in our experiments and which might be good candidates for manipulation in future work.

Expanding the Search Space:
The GA used in our experiments encodes all 378 external input features for any given individual, from an implementation perspective. However, because our evolutionary process considers a micro-external input to be either activated or not for *all* companies under consideration, our search space only comprises 30 external inputs in actuality – not all 378. This yields 2^{30} possible subsets of external input features. We could foresee using all 2^{378} features as input to the GA.

More Data: More Companies, More Days:
Given our relatively small sample data size (30 companies, 503 days), PCA did not yield a low rank approximation of the time-series data. Future work should explore the use of PCA with a substantially larger sample data set to determine if it leads to a low rank approximation under such conditions – and if so, if a low rank approximation does in fact lead to better results.

Monthly or Quarterly Predictions:
The time-series data changes daily; however much of the external data only changes on a weekly, monthly,

or quarterly basis, and is therefore held at constant levels for many consecutive time steps. Future work should include experimentation with different output and/or input strategies to deal with this inconsistency.

Handling Training Error:

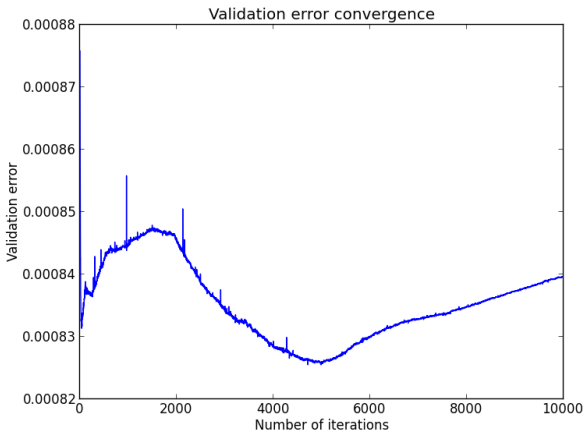

Figure 5: There is a clear minimum in the validation error curve for this experiment.

We did not use PyBrain's built in trainUntilConvergence() method because it did not have a good way to perform error reporting within it, nor was it able to handle training/validation/testing splits the way we wanted. Instead, we used a static number of epochs. Due to the staggering amount of data generated by our experiments, we only saved the final network weights. There is a clear minimum of prediction error shown in the validation error curves and it is not always at our prescribed number of iterations. You can see this in Figure 5. A method with better reporting should be developed that trains until convergence.

Altering Model Predictions:

While this list is not comprehensive, the last avenue of future work we will suggest is to experiment the effects of changing the model to predict the change in stock price for each company, rather than the actual price. We hypothesize that this would yield more statistically significant results.

6 Conclusion

Although we did not find clear connections between external features and better predictive performance, we were able to implement a recurrent neural network, PCA on time-series data, and a GA for evolving feature sets in a short amount of time. The GA was also able to determine that fewer features *are* better given the possibilities fed into it. By taking inspiration from multiple sources, adapting the ideas, and applying them to our networks, we have set up a good framework with which to pursue further questions about the nature of inputs to stock predicting networks. The preliminary results with the corrected version of the PCA input manipulation look very promising. Implementing these in concert with the GA and allowing for full phenotype expression with more time to explore mutation rates could yield better performance of our model.

The training times of these networks, especially when care is taken to avoid overtraining, is not very long once features are known. If one were able to do some offline calculation to evolve the feature set and preprocess large amounts of data with PCA, the general framework that we put forth could prove very useful in practice.

References

[1] A. A. Ayodele, A. K. Charles, A. O. Marion, and O. O. Sunday. Stock price prediction using neural network with hybridized market indicators. *Journal of Emerging Trends in Computing and Information Sciences*, 3(1):1–9, January 2012.

[2] D. M. Dunlavy, T. G. Kolda, and E. Acar. Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, 5(2):10:1–10:27, February 2011.

[3] E. Guresen, G. Kayakutlu, and T. U. Daim. Using artificial neural network models in stock market index prediction. *Expert Systems with Applications*, 38.

[4] I. Kaastra and M. Boyd. Designing a neural net-

work for forecasting financial and economic time series. *Neurocomputings*, 10:215–236, 1996.

[5] R. Lawrence. Using neural networks to forecast stock market prices. December 1997.

[6] S. A. Niaki and S. Hoseinzade. Forecasting s&p 500 index using artificial neural networks and design of experiments. *Journal of Industrial Engineering International*, 9(1):1–9, February 2013.

[7] M. Riedmiller. Advanced supervised learning in multi-layer perceptrons - from backpropagation to adaptive learning algorithms, 1994.

[8] M. Sheikhan and B. Movaghar. Exchange rate prediction using an evolutionary connectionist model. *World Applied Sciences Journal*, 7:8–16, 2009.

[9] L. N. Trefethen and David Bau III. *Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997.