Feedforward and Recurrent Neural Networks and Genetic Programs for Stock Market and Time Series Forecasting

Peter C. McCluskey

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-93-36
September 1993

Feedforward and Recurrent Neural Networks and Genetic Programs for Stock Market and Time Series Forecasting

by

Peter C. McCluskey
B. S., Yale University, 1978
Sc. M., Brown University, 1993

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University.

June 1993

This research project by Peter C. McCluskey is accepted in its present form
by the Department of Computer Science at Brown University
in partial fulfillment of the requirements for the Degree of Master of Science.

Date ______________ ____

Leslie Pack Kaelbling

Contents

Abstract

Adding recurrence to neural networks improves their time series forecasts. Well chosen inputs such as a window of time-delayed inputs, or intelligently preprocessed inputs, are more important than recurrence. Neural networks do well on moderately noisy and chaotic time series' such as sunspot data. A single neural network or genetic program generalizes poorly on weekly stock market indices due to the low signal to noise ratio. When the responses of a number of networks are averaged, the resulting forecast shows substantial profits on historical data.

keywords: Neural Networks, Time series forecasting, Recurrence, Cascade Correlation, Cascade 2, RTRL, Simple Recurrent Networks, Sequential Cascaded Network

Chapter 1

Introduction

I have studied the ability of neural networks to forecast the stock market (using the Standard & Poor's 500 Index), the annual sunspot data, and the Mackey-Glass time series and compared the results of a number of neural network training algorithms, both feed-forward and recurrent. For the stock market data, I have also compared genetic programming and hand-coded approaches.

The sunspot data and the stock market are interesting problems because they involve real-world data in large enough quantity that they are challenging to analyze, but still not enough data that experts can agree on a theory which explains them satisfactorily. The stock market is particularly interesting because there is serious disagreement about whether better-than-random predictions of stock prices are possible [11].

There are three ways that a neural network can forecast a time series. It can be provided with inputs which enable it to find rules relating the current state of the system being predicted to future states. It can have a window of inputs describing a fixed set of recent past states and relate those to future states. Or, it can be designed with internal state to enable it to learn the relationship of an indefinitely large set of past inputs to future states, which can be accomplished via recurrent connections.

These three methods require decreasing sophistication in the choice of inputs, but the training is increasingly difficult.

Chapter 2

Algorithms Used

2.1 Feed-forward Networks

1. Linear Associator (using the Widrow-Hoff rule) (WH)

   A single layer linear network.

2. Backpropagation (BP)

   A popular algorithm which uses one or more hidden layers.

3. Cascade Correlation (CC) [6]

   An algorithm which adds hidden units one at a time, training several candidate units at each step and choosing the one most correlated with the error.

4. Cascade 2 (C2) [8]

   Like cascade correlation, but the candidate units are trained to minimize the sum squared difference between the unit output and the error of the output layer.

   I also tried:

5. RAN (Resource Allocating Network) [13]

   This builds up a network of radial basis units one unit at a time.

   While I got the RAN to do some gradient descent, it did not work nearly as well as the papers indicated it should, suggesting there is still a bug in my code. I suspect from my results and from the Kadirkamanathan and Niranjan paper that the algorithm is sensitive to the threshold used to control when new units are added.

2.2 Recurrent Networks

1. Recurrent Cascade Correlation (RCC) [7] (see figure 2.1)

   Similar to the cascade correlation algorithm, but with the output of each hidden unit fed back as an input to itself.

2. Simple Recurrent Networks (SRN)[4](see figure 2.2)

   Like backpropagation, but with the outputs of the hidden layer fed back as inputs to that layer.

3. Real-Time Recurrent Learning (RTRL) [23, 24] (see figure 2.3)

   A single layer network, with the outputs of all units (of which there may be more than there are external outputs) fed back as input to all the units.

4. Sequential Cascaded Network (SEQ) [16] (see figure 2.4)

   Connects the inputs to the outputs by a single layer, whose weights are set dynamically by a matrix of context weights based on the previous output.

I also did some work with a BPTT variant (Back-Propagation Through Time) [22]. I did not get this to work. Williams and Peng only described how to train units that were connected to an output unit, although it appears that hiddens units could be created by adding a feedforward layer to the output of the recurrent layer, and training the second layer as in backpropagation, producing a network similar to Elman's SRN.

I attempted to implement Pearlmutter's Time-Dependent Recurrent Back-Propagation. I was unable to produce anything that looked like gradient descent. Since I did not find any indication that it has been used for anything more difficult than the (very simple) purposes to which Pearlmutter put it, and because of the following: "We replicated this result, but the original algorithm was very sensitive to the choice of parameters and initial conditions" [9], I decided it was not promising.

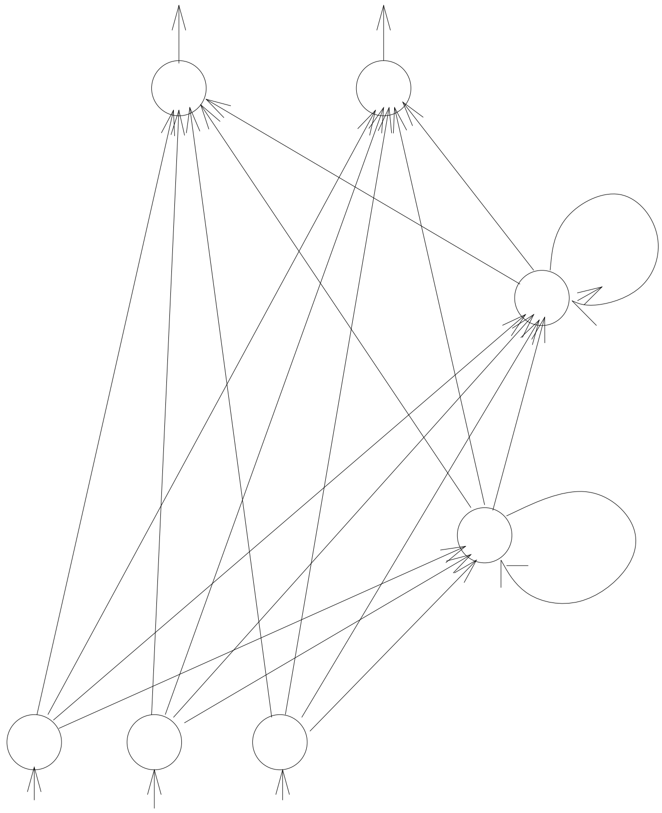

Figure 2.1: Recurrent Casacade Correlation

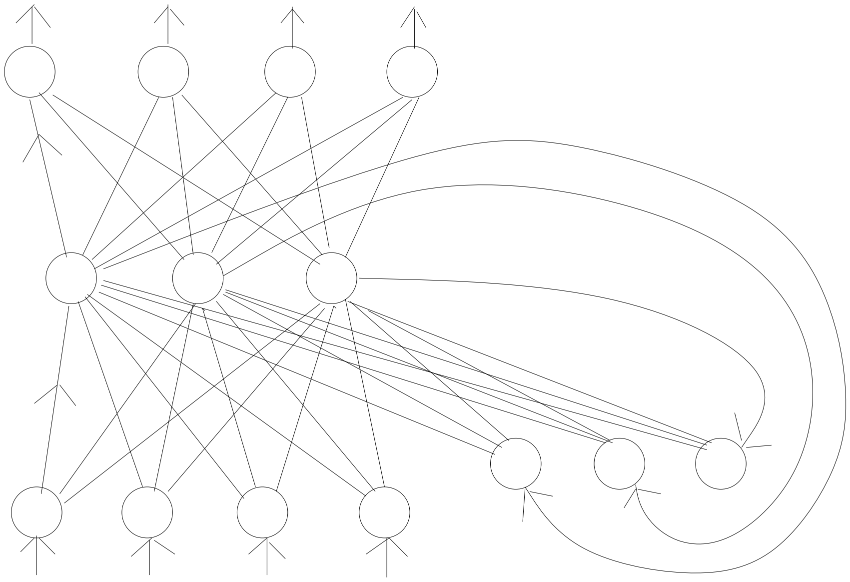

Figure 2.2: Simple Recurrent Network

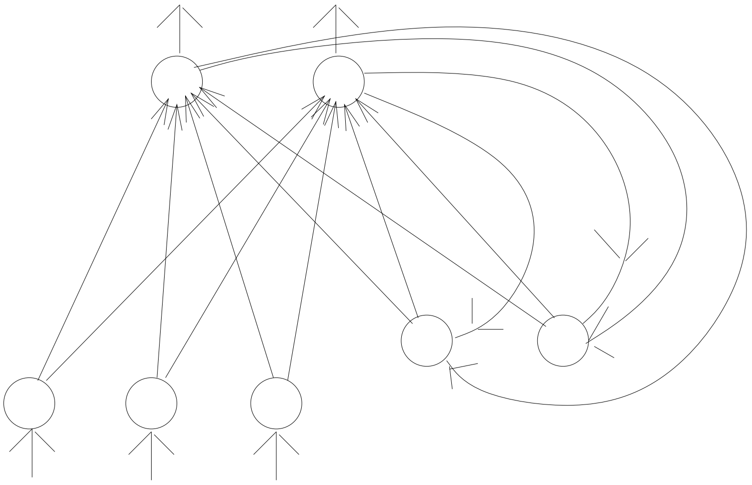

Figure 2.3: Real-Time Recurrent Learning

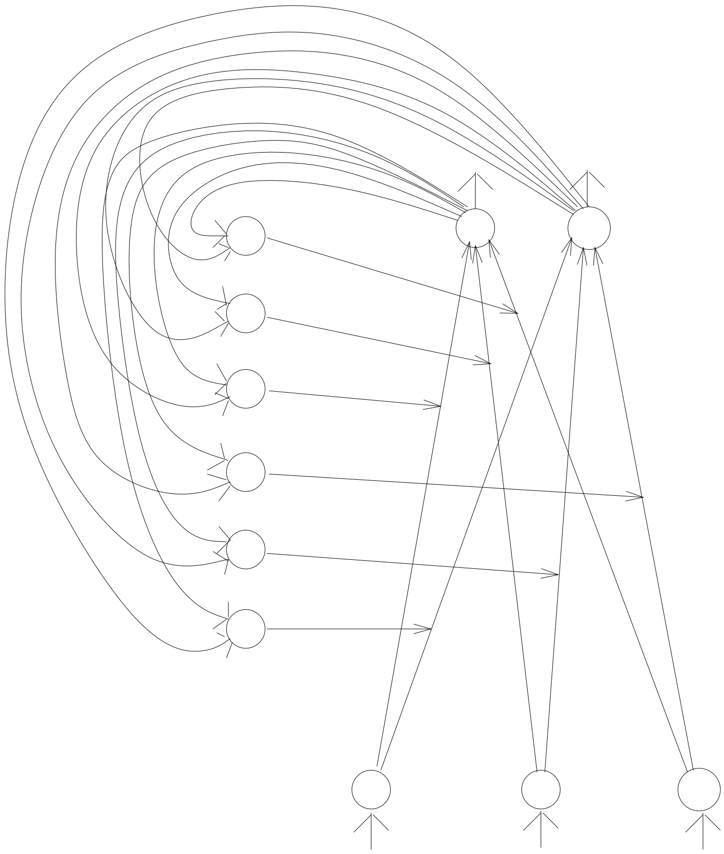

Figure 2.4: Sequential Cascaded Network

2.3 Genetic Programming (GP)

I start with several seed expressions (detailed in the appendix), and initialize the remaining individuals by mutating a randomly selected seed expression. When a mutation at this point produces an individual with a fitness of 0.0 (fitness values are limited to the range 0.0 (worst) through 1.0 (best)), that individual is discarded and replaced with a new mutated individual, up to 100 times if needed.

Mutation consists of the following steps (depending on the node type):

1. Raw input replacement for nodes which referred to input files:

   an input file (containing data as it was available from a primary source) is selected at random to replace an existing input file.

2. Operator replacement:

   an operator taking the same number of arguments is selected at random.

3. Constant optimization algorithm:

   Iteratively test different values and move in the direction of greater fitness.

   The constant is initially changed by $(1 + abs(value\_of\_constant))/2$, and the new fitness measured. This is repeated up to ten times. Whenever the fitness decreases, the next change to the constant is -0.95 times the previous change, otherwise the next change is 0.95 times the previous change or
   $(original\_change/100) * (change\_in\_fitness/prior\_change\_in\_fitness)$,
   whichever is less ($prior\_change\_in\_fitness$ is treated as 0.1 in the first iteration). This process terminates before ten iterations if the change to be added to the constant drops below 0.01 times its initial value, or at any time after four iterations if the change in fitness drops below 10^{-6} or the fitness is below 0.01.

If there are constants in the expression, with 50% probability the node to be mutated will be selected from the set of constant nodes, otherwise it will be selected from the entire set of nodes in the expression.

Crossover occurs by cutting edges in the expression tree, and swapping the resulting pieces.

The constant optimization should produce some of the advantages of the gradient descent that is available for neural networks while retaining the abilities of genetic programming to use a more powerful variety of operators than the add-and-multiply of a neural network, and to optimize reinforcement (it is difficult if not impossible for a neural network to learn functions that optimize reinforcement when a target output is unavailable and the output must be more complex than binary data).

2.4 Miscellaneous Implementation Details

All of the algorithms are implemented in C++. I have started with straight-forward implementations based on classes including vector and matrix implementations which produced a fair amount of overhead, primarily due to memory allocation. I then optimized the most frequently executed loops to the point where I believe that the overhead is small in comparison to the required operations, except that for the sequential cascaded network I have left in some duplication and overhead which may slow it down by a factor of 2 or 3, and for the evaluation part of the genetic program, I have not checked the efficiency as carefully.

For the cascade correlation and derived algorithms, I have copied a fair amount of code from several programs developed by Scott E. Fahlman, R. Scott Crowder III, Conor Doherty, and P. Michael Kingsley.

The carefully optimized features of the cascade correlation software make the comparison with the other algorithms (which I implemented from the basic theory with only a modest effort at optimization) somewhat biased.

Most calculations were done with 32 bit floating point numbers. Some testing was done earlier with 64 bit floating point numbers. There were no noticeable differences.

2.5 Handling of Empty Data

For many of the stock market inputs, data is not available for all time periods. To minimize the effect of this on the networks, I have coded the values as empty (represented as 10^{30}), and propagated the values as follows:

- If either number in a multiplication is missing, the result is treated as missing.
- If one number in an addition or subtraction is missing, treat that number as zero. If both are missing treat the result as missing.

This prevents empty values from causing any change in the weights. In hindsight, this could have been done more efficiently by representing the empty values by zero during neural net training, but there were many places in the preprocessing phase where it was important to avoid treating empty data as zero (most obviously when dividing by that number).

While it might be possible to improve on these results by adding a separate input associated with each of the existing inputs with a binary value indicating whether or not a valid data value exists, this approach would significantly slow down the training. I doubt that it would have enough impact on the results to justify the time required. If the network had trouble finding any rules which predict the training set, then I would have tried this approach, but I expect that with networks that find more correlations between single input - output pairs in the training set than are actually useful for describing the prediction set, that it would overtrain on the single input - output pair correlations faster than it would learn the "and" relationship between two inputs and an output, especially when that "and" relationship is obscured by a good deal of noise.

For all inputs and outputs, I have normalized the data so that the range is within the interval [-1,+1] by dividing the data by the maximum absolute value for that particular input. While not strictly necessary, this makes the choice of the best training rate easier. Most neural network training algorithms include a weight change which is based on something using

$training\_rate * (desired - actualoutput)$.

This can be shown to produce gradient descent when the training rate is infinitesimal, but training can be unstable if that expression is large compared to the weights, and standardizing the input and output ranges helps to simplify the choice of training rates needed to accomplish this. Also, some combinations of ranges could produce numerical stability problems, although I do not believe this would have been a problem with the data that I used.

Chapter 3

Sunspot Tests

I used the average annual sunspot data (approximately a count of the number of sunspots, with adjustments for sunspot groups and differences in telescopes), taken from [17]. I took the data from 1700 to 1920 for a training set, and from 1921 to 1979 for a prediction set. In addition to calculating the error for the whole prediction period, I have calculated it for what [21] calls the "early" period (1921 through 1955) and the "late" period (1956 through 1979).

I used the average relative variance:

$arv = \sum_k (target_k - prediction_k)^2 / \sum_k (target_k - mean)^2$,

where $mean$ is the mean of the complete data set, as my measure of the results, in order to compare them to the results in [21].

3.1 Neural Network Parameters

I scaled the data to a range of .1 to .9 for both the inputs and the outputs.

In all cases, there is 1 external output, 1 external input in the non-windowed cases, 12 external inputs representing the most recent 12 years of data in the windowed cases (or "empty" if the window extended back before the beginning of the data), and a bias input set to 0.5.

All tests used a logistic squashing function, with beta = 1. No momentum was used. The weights were updated incrementally except with cascade correlation, where batch updates were used. Weights were initialized to random values uniformly distributed between 0 and 0.1, unless noted otherwise.

SRN: unwindowed:

10 recurrent connections, 10 hidden units, training rate .06.

windowed version:

4 recurrent connections, 4 hidden units, training rate .06.

Sequential Cascade: unwindowed:

15 units plus a bias output to the context weights, training rate 0.3.

windowed version:

2 units plus a bias output to the context weights, training rate 0.01.

RTRL unwindowed:

6 recurrent connections, training rate .01 at the start, reduced 0.15% per epoch until it reached .0004.

windowed version:

2 recurrent connections, training rate .025 at the start, reduced 0.2% per epoch until it reached .004.

Cascade Correlation (both recurrent and feedforward):

3 hidden units for the windowed versions, 8 for unwindowed recurrent version
32 candidate units, 200 epochs per candidate unit, 200 epochs of training for the output connections after each hidden unit added.
patience = 15 (how many epochs with little change justifies quitting),
epsilon = 1.0 (used in quickprop, similar to training rate),
weight decay = 0,
threshold = .01 (minimum change used with patience),
mu = 2.0 (training rate used in quickprop)
Weights for the candidate units in the recurrent version were initialized to random values uniformly distributed over -1.0 to 1.0 (I have no good reason for using a different range here; I adopted it while debugging and never changed back).

Cascade 2 :

The same parameters as for cascade correlation (with the candidate output parameters the same as candidate input parameters), for the sunspot tests only.

BackPropagation :

8 hidden units, plus a bias to both layers, training rate .05.

Widrow-Hoff :

bias input, training rate $0.04/\sqrt{epochnumber}$.

The Cascade Correlation parameters are based largely on the defaults and examples used in the software which was made available by Scott Fahlman. My tests indicated that the algorithm has very little sensitivity to small changes in them.

The training rates for the remaining algorithms were set by testing with a large value and reducing it in subsequent tests until the error no longer "blew up" (increased suddenly and remained high) during training.

The number of epochs was determined by looking at how long it took for the prediction error to turn up during the trial runs I did to decide on the training rate and the network size. Ideally, this decision should have been made on the basis of the training set alone or a validation set [21].

3.2 Sunspot Results

name	epochs	training set ARV	training set RMSE	prediction set ARV	prediction set RMSE	early ARV	late ARV	CPU time (minutes)
		no window						
RCC		0.110	0.334	0.136	0.385	0.116	0.149	14
SEQ	5000	0.183	0.430	0.240	0.511	0.207	0.261	159
RTRL	5000	0.249	0.501	0.202	0.468	0.206	0.200	333
SRN	1000	0.190	0.438	0.202	0.468	0.195	0.206	6
		with window						
C2		0.118	0.345	0.104	0.336	0.078	0.120	6
CC		0.112	0.336	0.119	0.358	0.095	0.133	8
RCC		0.112	0.336	0.107	0.341	0.082	0.122	8
SEQ	5000	0.126	0.357	0.142	0.393	0.077	0.182	60
RTRL	5000	0.183	0.431	0.133	0.380	0.126	0.138	53
SRN	5000	0.102	0.321	0.152	0.407	0.091	0.190	18
BP	5000	0.197	0.441	0.268	0.536	0.202	0.308	26
WH	5000	0.169	0.414	0.129	0.374	0.110	0.141	6

ARV stands for average relative variance.

RMSE stands for the normalized root means square error (the rms error divided by the standard deviation).

All times are from a lightly loaded Sparc 1.

The results are the average of 5 runs.

3.3 Analysis of Sunspot Results

The choice of window size, and the number of hidden units for backpropagation, was influenced by a desire to compare my results with results for backpropagation and threshold autoregression as reported by Weigend, Rumelhart, and Huberman in [19, 21]

With the training rate of .1 which they used for backpropagation, I got very unstable fluctuations in the error, but reducing the training rate produced results similar to theirs.

For all networks, windowing was more important than the recurrent connections. The window size of 12 apparently produces all the information that any known time series method can make significant use of, and direct connections are easier to learn than multi-step uses of recurrence.

The good results for the Widrow-Hoff associator indicate confirm that it is hard to add useful information to the 12 input window, as it is one of the least powerful algorithms.

Cascade correlation with a window produced much better results than other methods. Its results after adding one hidden unit were often better than what other networks could ever produce. However, the primary effect of adding more units was to memorize the training set, with only slight effects on the prediction set. Very little fiddling with the defaults provided with the sample code that I ftp'd was needed to produce the results that are shown.

Cascade 2 and the windowed recurrent cascade correlation behaved in a manner indistinguishable from cascade correlation, probably because the hidden units had little importance.

Elman's SRN behaved much like the best results from backpropagation, although with the non-windowed case the prediction error would sometimes bottom out after a few hundred epochs, rise slowly for over a thousand epochs, then slowly decline to slightly higher than the earlier minimum. It is unclear whether it would have improved upon the early minimum if I had allowed it to run indefinitely.

The differences between the error rates for a given set of parameters was typically less than 10%, except for the backpropagation results, for which the was a difference of a factor of about 2 between the best and worst results. Backpropagation appears to be somewhat more sensitive to the initial weights than other algorithms, although the Mackey-Glass results indicate that this difference is not as large as the sunspot results suggest.

The Sequential Cascaded Network is one of the simplest recurrent networks. Its results are mediocre.

RTRL was very sensitive to the choice of training rate. It required substantial trial and error to find a way to produce reasonable results with training times of less than a day of Sparc 1 CPU time.

I am unable to explain the cases where lower errors were reported on the prediction set than on the training set. This occurred consistently with the RTRL and with the Widrow-Hoff associator, and sporadically under other conditions.

RTRL requires $O(o^3 i)$ time to train, and $O(o^2 i)$ space during training, for a given number of training epochs, where i is the number of inputs (including recurrent inputs), and o is the number of outputs (including recurrent), with $i \geq o$, and it appears that the number of epochs needed increases more rapidly as a function of the network size than with most other network algorithms.

Assuming that the number of recurrent connections in RTRL and the number of hidden units in networks such as backpropagation and cascade correlation scale up as $O(max(i, o))$, then the comparable limit is $O(max(i^2, o^2))$ for the multi-layer feedforward networks.

The Sequential Cascaded Network requires $O(o^2 i)$ time and space during training, and unlike RTRL its poor scale-up applies even when training is complete. It is hard to say precisely how it scales up because I have very little intuition for how many context units are needed for a given problem.

3.4 Mackey-Glass Results

This series is computed by integrating:

$dx/dt = a * x[t - \tau]/(1 + x[t - \tau]^{10}) - b * x[t]$,

with a = 0.1, b = 0.2, and τ = 17. I used 5000 points for the training set and 1000 points for the prediction set.

name	number hidden units	training rate	epochs	training ARV average	best	worst	prediction ARV average	best	worst
	using a single (unwindowed) input:								
SRN	20	0.04	3000	0.00098	0.0005	0.015	0.0608	0.0025	0.1278
SRN	25	0.04	3000	0.00074	0.0003	0.0013	0.0092	0.0024	0.0315
SRN	30	0.04	2500	0.0048	0.0020	0.0071	0.0190	0.0047	0.0600
RCC	50			0.00134	0.00102	0.00190	0.00324	0.00147	0.00435
	using a window of inputs delayed by 1, 6, 12, and 18								
SRN	20	0.04	3000	0.0004	0.0002	0.0005	0.0060	0.0015	0.0132
SRN	25	0.04	2500	0.00074	0.0003	0.0013	0.0092	0.0024	0.0315
SRN	30	0.03	2000	0.00124	0.0007	0.0017	0.0058	0.0023	0.0164
BP	20	0.05	3000	0.0087	0.0042	0.0165	0.088	0.0043	0.0170
BP	25	0.05	3000	0.0220	0.0131	0.0489	0.0225	0.0135	0.0498
BP	30	0.05	3000	0.0154	0.0046	0.0214	0.0158	0.0047	0.0220
SEQ	28X8	0.02	3000	0.0220	0.0134	0.0323	0.0326	0.0164	0.0559
CC	50			0.00072	0.00049	0.00100	0.00085	0.00051	0.00141
C2	50			0.00034	0.00030	0.00038	0.00052	0.00044	0.00057
RCC	50			0.00031	0.00019	0.00054	0.00040	0.00019	0.00092

ARV stands for average relative variance.

The results are the average of 5 runs.

The cascade 2 parameters were changed to: 16 candidate units, all patiences = 20, candidate input epsilon = 100, candidate output epsilon = 10, candidate weight decay (input and output) = 0.0000000001, and output epsilon = 0.1.

3.5 Analysis of Mackey-Glass Results

Recurrence is more important here than in the sunspot tests, probably because the input window was less effectively chosen (I followed [13]).

There was substantially greater variation in the results for a given algorithm here.

Recurrent connections are more important here than with the sunspot data, causing windowed recurrent cascade correlation to be the best.

Cascade 2 is better than cascade correlation for this problem (and reportedly for most problems needing real valued outputs), because the training of its units produces a better estimate of the magnitudes of the appropriate hidden unit output when the error is small.

Chapter 4

Stock Market Forecasting

4.1 Organization of Networks

I have classified the types of inputs by the degree of preprocessing. The lowest level is the raw data (normalized), the intermediate level (derived data) combines one to three types of raw data with a few arithmetic operations intended to replicate the most basic types of indicators used on Wall Street, and the high level indicators classify the data into five broad approaches.

A. Single network

1. High level indicators as input (5 categories)
2. Derived values as input (31 different indicators)
3. Raw values as input (31 types as taken from primary sources plus 16 seasonal bits)
4. windowed S&P500 historical price data only

B. Multiple networks of the same type, averaged

I trained up to 64 networks, each with the same parameters, and averaged their responses to produce the final output.

C. Two layers of networks

The second layer is a single network whose inputs are the outputs of the networks in the first layer.

1. 1st layer of networks similar to derived values

2. 1st layer of networks similar to high level indicators

D. No neural net (hand-coded)

A weighted average of the derived values that were used as input in A.2, with the weights determined by a trial and error process consisting of a single pass through all the inputs and testing various weights starting with 1.0 and varying them by 0.5 in both directions until a locally optimal result was found.

E. Genetic Programming

Starting with a population of expressions similar to the derived values in C.2, the genetic algorithm employed crossovers at the edges of the parse trees, mutations (replacing one operator with another of the same arity), and gradient descent on the real-valued constants in the expressions.

My evaluation function is:

$max(0, 1 - 0.8 * sp\_profit/gp\_profit)$,

where sp_profit is the profit from a strategy of being 100% invested over the training period, and gp_profit is the profit from using the genetic programming expression to create a forecast of the S&P 500 weekly change, and calculating an investment level as described below for neural net outputs. Its value approaches 1 as the profitability of the expression approaches infinity, and a value approaching zero represents a strategy somewhat worse than a buy and hold strategy. The factor of 0.8 is somewhat arbitrary and I didn't test alternatives rigorously. Substantial increases would limit the number of possible non-zero seed values. Decreasing the factor would reduce the difference between good and bad individuals, which would slow down the rate of fitness increase. The resulting increased diversity might improve the results, but the time required for the tests that I did was quite large already. A better type of evaluation function would approach zero asymptotically as the strategy approached a total loss of investment over the training period.

I initialized the population by starting with 3 expressions (listed in appendix F) that I selected based on my hand-coded tests, and then mutating randomly selected expressions from those seeds to produce the remaining individuals.

I believe (although I have not verified) that it is important for this application to be carefully initialized because the evaluation function gives a fitness of zero to many individuals because the number of strategies which outperform the market is a tiny fraction of the representation space, so that random initialization could easily produce an entire population with zero fitness.

Most of the genetic programs were run for 3 generations of 50 individuals with a crossover probability of .9, a mutation probability of .4, and the 2 best individuals tenured. The last set of tests listed used 10 generations of 100 individuals with a crossover probability of .9, a mutation probability of .2, and no individuals tenured.

4.2 Output Postprocessing

I trained the networks to predict the change in S&P 500 closing prices over the either a single week or for next 1, 2, and 4 weeks. I also tried target output look-ahead combinations of 1, 2, 4, and 8 weeks and 1, 4, 13, and 52 weeks, but the results from these were not worth detailed reporting.

I then converted the expected change for a single week into an investment level (-1, 0 or +1, representing 100% short, 100% in cash (very rare) or 100% invested), depending on the sign of the expected change. If the commission costs needed to change to this investment level exceed the expected weekly gain from this investment level, the change is limited as follows:

$invest = last\_invest + expect\_gain / commission\_cost * (invest - last\_invest)$;

where $expect\_gain = fact * expect\_change * invest$;

After trying several methods of initializing $fact$ based on the average absolute change that was forecast, I settled on fact = 1 as the best value that I could find.

The conditions under which this limiting procedure affected the investment level ended up being rather rare.

I also trained a linear associator to map the different forecast intervals that were output from the network into a single forecast for 1 week change, and evaluated the resulting forecast as above.

The weights in this secondary network almost always ended up with the weight for the 1 week input somewhat greater than 0.5, the 2 week input weight (if present) was somewhat less than 0.5, and the weights associated with longer look-aheads significantly lower, often slightly negative, indicating that the longer-term forecasts of the primary network are of no use.

I tried two other postprocessing approaches, primarily to reduce the negative effects of commission costs.

In place of the limit on investment changes based on expected commission costs, I tried filtering the investment level as follows:

$a * invest + (1 - a) * last\_invest, 0 < a < 1.$

Unfortunately, reducing a to the point where it noticeably reduced the expense of making frequent changes in investment also slowed down the reaction to signs that a major change (particularly a crash) was imminent.

Another approach was to train a network to predict the reward given as inputs the investment level, one or more time periods worth of forecasts, and possibly the change in investment and the commission costs. This could then have been used to find an optimum level of investment for given forecasts by using trial and error to find the optimum expected reward. This approach failed because the sign of the weights connecting the investment level to the reward would have to have been positive when the correct forecast was positive, and negative if the correct forecast was negative. This is equivalent to an XOR problem, except that the minimum that I want the network to find is just barely better than the simpler local minimum (positive weights indicating a strategy of buy and hold) and covers a much smaller fraction of the total weight space than the simple local minimum, with the result that this approach invariably produced a buy and hold strategy.

I used the closing price on the last trading day of each week as the price for that week, and limiting all actions and profit calculations to those prices. I modeled transaction costs by a commission that is a fraction of any change in the level of investment. I measured the profits using 0, 0.1%, and 1.0% commission rates.

I assumed that the data for each week is available (or can be estimated closely enough) to allow predictions to be made before the close of trading. For most of the data, I believe that this is a reasonable assumption. I checked some of the tests to verify that small changes in inputs don't change the investment level at important times, especially October 1987.

I have also assumed that trades can be made at the closing price, when in fact all that can be expected is that trades can be executed within a few minutes of the close. When the S&P 500 could not be purchased as a basket via the futures market, it would not always have been possible to buy the individual stocks near the close because trading in individual stocks may have been halted.

4.3 Division into training and prediction sets

The simplest approach would be to use several of the most recent years for the prediction set, and the rest for training. This has the disadvantages of preventing the use of some types of raw data which are only available for the past few years, and of testing on a prediction set that may reflect some phenomena (such as investor attitudes) which may persist for several years but which are still atypical.

The other approach is to select several years from different portions of the available data as the training set. This has the disadvantages of obscuring longer term trends and cycles (although I don't have much hope of recognizing these anyway), and of reducing the independence of the training and prediction data (i.e. avoiding memorizing specific patterns that reflect fads or moods that last on the order of a year).

I divided up the data into two periods:

- prior to November 16, 1979, and
- November 19, 1979 to April 2, 1993.

The second is the period for which I have nearly complete data, while I am missing a lot of data for the first. I used January 1 1928, January 1 1945, and January 1 1960 as different starting points for the first period, corresponding to starting points of the many important inputs.

I then divided each of these into five periods of equal time, and trained the network(s) on four of the five smaller period within either the second or both of the larger periods, using the remaining fifth of the data as a prediction set.

It is impossible to fully evaluate the ability of any theory to predict the stock market due to the limitations described above and due to the following: the indicators used may have been selected from a large enough set of possible indicators that they "work" only because they are lucky (there is an indicator based on the Superbowl whose excellent track record must be an example of this) and because the widespread use of an indicator reduces its effectiveness.

4.4 Stock Market Test Results

TRAINING ON FULL 1928–1993 PERIOD

ANNUALIZED RETURN ON INVESTMENT (percent)

Net	input	XV	period a		period b		period c		period d		period e	
type	type		train	test	train	test	train	test	train	test	train	test
CC	d 4a	10.41	26.65	1.85	26.68	-1.62	20.67	6.16	19.83	23.06	23.61	22.62
CC	d 4a w	11.05	25.85	-1.02	26.38	-0.69	22.44	9.58	19.16	21.01	25.29	26.35
CC	d 4b	11.49	24.81	9.41	30.76	-0.53	27.50	3.49	19.29	22.87	19.37	22.23
CC	d 4b w	10.49	23.79	1.65	30.24	-0.23	28.11	3.80	19.72	21.55	19.63	25.69
RCC	d	10.50	26.22	0.36	27.68	2.16	27.03	4.02	28.93	21.81	22.15	24.16
RCC	d w	11.60	25.89	-2.84	26.82	1.83	23.71	13.86	27.75	21.21	22.57	23.92
GP		11.74	19.70	0.14	17.63	13.29	17.01	17.50	16.37	14.58	15.31	13.21
hand-coded		18.39	16.44	25.37	18.99	15.02	17.27	21.71	20.00	11.32	18.02	18.54
SP500		9.84	12.12	0.06	8.60	13.81	7.76	18.21	9.29	11.00	10.67	5.94
cash		3.91	3.94	3.74	4.35	2.10	4.19	2.74	3.71	4.67	3.31	6.29

Periods used for prediction sets (the remainder of 1928–1993 is used for training):

period a =	Jan 1 1928 – May 18 1938	Nov 19 1979 – Jul 23 1982
period b =	May 19 1938 – Oct 1 1948	Jul 24 1982 – Mar 25 1985
period c =	Oct 2 1948 – Feb 15 1959	Mar 26 1985 – Nov 26 1987
period d =	Feb 16 1959 – Jul 2 1969	Nov 27 1987 – Jul 30 1990
period e =	Jul 3 1969 – Nov 16 1979	Jul 31 1990 – Apr 2 1993

The XV column is the average of the 5 columns of test set annual returns.

A 4a in the input type column means that the network was trained to forecast the 1, 4, 13, and 52 week change. A 4b in that column means that 1, 2, 4, and 8 week change forecasting was used.

All other neural net tests in this and the next table were run with 1, 2, and 4 week change forecasting. The tests with 1960–1993 data were run with one week forecasting only (except for one test with two week forecasting only).

There are more notes at end of next table.

TRAINING ON 1979–1993 PERIOD ONLY

ANNUALIZED RETURN ON INVESTMENT (percent)

Net type	input type	XV	period f train	period f test	period g train	period g test	period h train	period h test	period i train	period i test	period j train	period j test
WH -	h	15.34	37.60	15.27	37.90	31.13	33.66	0.30	37.89	11.30	43.70	18.68
WH -	h w	16.28	37.51	12.45	35.11	31.07	34.96	4.24	38.78	16.66	42.04	16.98
BP 4	d	10.52	40.18	9.53	44.42	-0.69	43.45	38.18	9.21	-9.39	54.31	14.97
BP 4	d w	7.32	39.60	9.32	43.23	0.65	37.21	19.60	28.99	-7.94	51.64	14.97
BP 3	d	14.21	29.32	14.72	36.54	8.31	33.09	22.52	23.53	10.53	34.68	14.97
BP 3	d w	17.74	30.50	22.27	36.54	12.63	31.91	28.31	23.53	10.51	35.80	14.97
BP 2	d	14.58	32.52	14.78	29.51	32.36	20.57	16.37	17.34	-3.52	23.80	12.91
BP 2	d w	14.43	33.13	13.84	29.51	32.36	20.57	16.37	21.33	-3.55	23.90	13.14
BP 3	h	16.84	18.03	32.45	26.82	29.13	21.40	32.56	22.80	0.07	22.18	-10.01
BP 3	h w	15.23	22.37	19.96	24.91	35.08	17.45	20.41	27.04	0.36	25.56	0.36
SRN	h	20.51	25.59	26.43	26.27	32.66	23.21	25.22	29.52	8.65	30.92	9.59
SRN	hw	22.83	21.78	14.93	18.52	45.73	24.57	17.49	22.17	16.26	24.61	19.75
SRN	h	18.92	26.18	13.05	26.87	35.94	20.19	25.64	27.95	11.09	29.34	8.85
SRN	hw	24.55	20.44	14.37	21.40	40.65	23.52	21.16	22.42	23.97	21.69	22.58
SRN 3	d	12.21	37.09	23.63	45.46	1.94	40.69	7.55	44.74	9.66	54.34	17.27
SRN 3	d w	11.19	37.77	24.91	42.86	0.30	34.22	-0.57	44.18	9.90	49.97	21.42
SRN 2	d	15.96	34.75	30.11	36.66	29.58	32.77	0.53	44.86	2.92	42.83	16.65
SRN 2	d w	16.10	33.78	33.92	38.25	26.48	34.46	2.55	41.72	2.56	42.68	14.98

Numbers immediately after the network type indicate the number of hidden units.

A d in the input type column means that “derived values” were used as input.
An h means that “high-level indicators” were used.
A D means that two layers with the first similar to the derived values were used.
An H means that two layers with the first similar to the high-level indicators were used.

If followed by a w, then the results are from 3 outputs combined via a linear associator, otherwise the results are from a single output.

Periods f through j are the subsets of periods a through e which exclude dates prior to November 19, 1979.

(Table continued on next page.)

Net type	input type	XV	period f train	period f test	period g train	period g test	period h train	period h test	period i train	period i test	period j train	period j test
CC	d	15.79	42.38	23.88	46.22	17.53	41.84	12.59	32.49	6.66	39.26	18.27
CC	d w	18.15	39.79	23.91	49.35	17.53	41.61	23.40	34.91	10.60	36.13	15.29
CC	h1	21.97	33.11	33.66	32.61	20.11	27.07	21.71	24.72	11.72	27.49	22.65
CC	h1 w	17.54	32.43	25.41	28.21	22.47	26.12	14.76	24.44	12.58	25.19	12.50
CC	h2	14.26	36.59	22.68	29.74	18.99	31.05	8.17	28.78	3.32	29.14	18.16
CC	h2 w	12.77	31.46	24.65	26.71	11.48	31.77	3.31	25.10	1.65	22.14	22.75
CC	D	11.84	42.74	20.46	40.15	29.22	50.43	-6.06	49.64	-4.65	53.04	20.23
CC	D w	13.07	43.39	11.99	40.52	29.38	46.46	2.40	45.96	0.73	44.64	20.85
CC	H	13.64	41.73	16.47	42.26	18.60	47.38	10.32	42.98	10.63	39.49	12.15
CC	H w	12.07	38.66	4.18	42.09	16.20	46.03	13.13	44.03	12.61	42.69	14.25
CC	d	12.76	45.94	26.04	36.48	20.72	37.43	6.33	41.20	9.46	31.91	1.25
CC	d	14.32	40.04	11.06	34.71	20.72	37.24	3.66	39.10	14.36	41.39	21.82
RCC	d	15.72	42.29	22.13	44.38	16.80	37.13	6.83	37.26	7.53	44.90	25.31
RCC	d w	16.74	41.12	28.93	40.20	17.52	35.02	3.70	38.28	12.21	48.86	21.34
RCC	h	14.59	21.83	9.52	17.41	40.82	28.76	3.89	26.66	14.97	26.84	3.76
RCC	h w	16.85	20.76	9.28	16.57	36.78	17.68	19.82	25.57	13.66	22.83	4.72
GP		18.36	21.80	10.56	25.51	36.07	28.37	20.18	30.87	12.79	31.83	12.21
GP		16.04	28.69	8.84	28.27	29.34	28.47	20.61	34.33	6.35	31.72	15.06
hand-coded		28.03	25.91	33.49	23.69	43.37	26.68	29.96	30.86	14.08	29.16	19.27
SP500		15.75	17.57	6.47	12.46	27.19	15.56	15.24	14.03	18.79	16.72	11.08
cash		8.38	7.09	13.53	8.12	9.26	8.82	6.48	8.48	7.78	9.22	4.84

TRAINING ON 1960–1993 PERIOD

Net	input	hidden	#	XV	period k		period l		period m		period n		period o	
type	type	units	nets		train	test	train	test	train	test	train	test	train	test
C2	h	2	64	14.87	22.77	15.24	20.93	21.10	20.50	13.12	20.05	16.19	23.75	8.70
C2	h	3	16	14.46	22.34	14.46	20.83	19.17	22.46	13.83	20.16	16.19	26.65	8.66
C2	d	2	16	22.14	33.32	24.19	31.74	20.12	30.27	35.02	32.02	14.33	34.25	17.04
C2	d	2	64	21.82	33.62	24.32	33.64	22.80	31.57	28.09	31.66	14.63	35.92	19.26
C2 2w	d	2	16	20.71	30.61	24.04	33.01	21.60	27.26	24.62	33.93	10.07	33.27	23.20
C2	d	3	8	20.20	33.83	26.54	32.05	20.46	33.12	25.61	35.98	14.92	42.94	13.48
C2	d	3	16	19.97	33.21	23.61	33.65	20.05	32.33	26.18	32.17	14.46	35.78	15.54
C2	d	3	32	21.44	34.03	27.33	35.73	23.20	31.67	26.90	35.69	15.13	37.97	14.65
C2	d	4	16	19.94	32.46	22.33	35.00	20.39	31.43	24.63	35.77	15.91	37.62	16.43
C2	dw5	3	16	44.92	69.92	47.73	71.05	17.86	69.20	57.56	63.82	64.68	66.96	36.76
C2	dw5	5	1	50.84	63.50	52.14	80.29	22.86	73.65	60.87	73.41	75.07	81.95	43.27
C2	dw5	5	4	53.69	71.78	57.02	82.33	23.60	74.30	65.50	73.28	79.65	78.59	42.70
C2	dw5	5	8	52.98	70.48	52.88	80.60	21.28	75.78	70.23	68.18	75.69	80.38	44.82
C2	dw5	5	16	52.42	70.86	55.11	79.86	23.93	74.75	67.30	69.28	78.21	80.35	37.53
C2	dw5	5	32	52.45	70.85	58.75	77.98	22.29	73.94	63.70	70.98	75.39	82.91	42.14
C2	dw4	5	1	45.00	65.87	45.03	71.39	38.18	67.65	44.92	62.60	61.73	67.65	35.13
C2	dw4	5	4	43.49	60.81	40.63	70.53	32.21	66.92	53.49	64.21	51.84	69.80	39.27
C2	dw4	5	8	44.92	62.15	48.31	70.48	29.50	68.18	54.76	64.18	56.66	67.56	35.39
C2	dw4	5	16	46.96	64.33	45.14	69.25	38.67	69.44	54.61	63.64	56.84	69.82	39.56
C2	dw4	5	32	44.77	65.74	43.66	71.19	33.03	65.57	60.07	60.95	49.84	67.63	37.23
C2	dw4	10	16	45.04	68.22	42.65	75.62	26.99	71.14	55.06	68.41	63.22	76.27	37.26
CC	d	3	64	21.69	35.96	27.82	32.66	21.64	29.15	30.04	33.22	13.94	36.80	15.00
CC	h	3	16	13.69	31.06	22.58	31.39	10.48	35.11	16.03	31.55	9.31	37.26	10.04
CC	dw5	5	16	46.24	67.79	49.09	75.34	22.53	71.05	53.81	66.81	68.33	76.57	37.43
RCC	d	2	16	22.86	31.60	31.36	32.92	20.69	32.06	25.92	34.14	18.58	35.81	17.78
RCC	h	2	64	20.16	30.13	13.47	26.74	26.86	26.80	16.58	24.31	31.56	29.41	12.41
RCC	h	3	64	20.83	30.67	13.08	31.24	21.44	30.55	21.93	31.02	35.88	31.25	11.81
RCC	d	3	16	23.84	60.00	36.29	68.35	27.96	63.09	26.74	58.28	18.01	59.70	10.27
RCC	d	4	8	28.76	69.15	38.47	72.83	27.43	61.02	36.22	66.23	18.53	68.47	23.14
RCC	d	4	16	25.50	64.10	34.36	50.92	17.78	68.01	25.07	65.79	23.99	70.24	26.32
RCC	d	5	16	28.64	70.48	30.44	77.65	27.21	72.51	38.39	71.78	22.78	76.80	24.37
RCC	d	6	8	27.82	75.11	35.18	80.11	27.97	73.96	31.42	74.43	24.24	78.01	20.31
RCC	d	6	16	29.43	75.42	36.13	85.86	33.42	80.68	37.72	72.46	19.59	83.54	20.30
RCC	d	7	8	28.84	77.05	33.14	84.51	29.33	83.37	37.95	75.21	24.31	86.74	19.46
RCC	d	7	16	31.00	78.21	35.05	87.06	25.64	80.82	42.38	74.63	26.22	87.78	25.71
RCC	d	10	16	32.96	91.59	38.63	99.64	31.89	95.56	43.79	82.67	29.08	98.06	21.41
RCC	dw4	5	16	43.93	109.83	53.10	116.99	22.19	109.30	53.69	100.85	61.54	114.99	29.14
RCC	dw5	5	16	50.78	113.22	64.56	117.40	27.50	111.34	57.16	104.91	96.70	26.52	8.00

Net type	input type	hidden units	# nets	XV	period k		period l		period m		period n		period o	
					train	test	train	test	train	test	train	test	train	test
SRN	h	3	1	21.29	21.14	18.79	18.43	25.25	19.41	24.18	17.60	26.35	26.04	11.86
SRN	hw	3	1	18.83	20.09	10.76	18.33	22.92	19.43	24.67	15.10	28.41	23.88	7.38
SRN	h	3	32	18.34	21.58	12.88	20.58	24.26	18.40	11.38	15.90	31.92	24.04	11.25
BP	d	2	64	21.74	29.85	24.62	27.58	20.11	22.39	20.31	22.78	22.48	28.17	21.05
WH	d		64	21.43	32.66	29.66	32.82	21.92	30.33	25.75	34.62	15.49	38.42	14.31
hand-coded				20.27	19.66	21.10	19.11	23.30	19.81	19.17	17.37	29.34	22.75	8.45
sp500				10.15	10.56	8.33	8.58	16.60	10.59	8.88	10.44	7.32	10.11	9.63
cash				6.55	6.42	7.03	6.62	6.25	6.64	6.17	6.40	7.11	6.62	6.18

2w – means the network was trained to predict the 2 week change in the S&P 500.

dw4 – means the input used a window of inputs delayed by 1, 4, 10, and 28 weeks
dw5 – means the input used a window of inputs delayed by 1, 2, 3, 5, and 9 weeks

Periods k through o are the subsets of periods a through e which exclude dates prior to January 1, 1960.

The backpropagation and Widrow-Hoff networks in this table were trained for 2000 epochs each.

The cascade algorithms in this table used 8 candidate units rather than 32 when more than one network and the derived inputs were used.

The recurrent cascade correlation had an output epsilon of 0.1 and 1000 output epochs when windowed inputs were used.

The cascade 2 parameters differed from the sunspot test parameters as follows: candidate input epsilon = 100, candidate output epsilon = 10, all patiences = 15, candidate weight decay (input and output) = 0.0, and output epsilon = 0.1.

In order to test the sensitivity of these results to individual inputs and to attempt to detect any faulty data that might be affecting the results significantly, I reran the 16 network, 10 hidden unit recurrent cascade correlation test with one of the indicators replaced by zeros in the input. Since it showed an unusual dependence on the djua_vs_djia indicator, I also ran a test using cascade2 with the 5 time period windowed derived inputs, 5 hidden units and 4 networks, without that indicator. The results are shown in the following table:

Indicator Ommitted	XV	period k train	period k test	period l train	period l test	period m train	period m test	period n train	period n test	period o train	period o test
freereserves	32.59	91.11	39.84	98.37	29.92	93.96	39.01	88.44	28.41	98.86	25.77
yieldcurve	26.52	89.88	32.41	99.73	17.11	94.92	40.38	86.29	23.75	93.60	18.97
realtbill	30.86	89.65	37.60	99.04	25.88	92.07	44.93	85.85	20.08	97.22	25.79
realtbond	30.86	89.90	39.43	100.52	28.69	95.87	38.76	87.60	23.59	97.95	23.84
disc_trend	29.35	92.19	35.52	100.39	28.01	92.17	39.28	86.43	23.18	97.29	20.76
disc_tbill	28.52	91.75	27.55	96.86	30.33	92.05	35.57	87.00	23.25	93.13	25.90
disc_change	30.23	92.53	39.85	97.39	31.48	94.33	30.78	83.17	26.01	97.34	23.01
tbill_trend	27.79	90.60	37.69	86.78	22.10	91.83	40.66	84.04	22.87	95.95	15.61
installdebt	34.05	90.08	32.01	98.66	31.58	92.86	42.22	88.87	39.59	98.85	24.83
specshort_ratio	30.42	91.67	34.01	101.18	31.24	91.49	44.08	85.33	25.70	93.68	17.09
secondary_offers	28.07	89.60	34.40	95.74	25.38	94.42	43.55	87.89	19.67	96.45	17.34
book_to_djia	31.47	90.11	38.98	102.21	29.75	95.50	34.88	86.88	31.00	94.98	22.73
book_osc	30.81	88.61	31.56	99.29	29.29	91.78	43.00	84.59	31.58	94.95	18.60
tbill_sp500yield	31.53	89.08	39.16	100.56	30.30	90.96	37.12	86.64	24.31	97.26	26.75
sp500_vs_200day	31.64	93.06	39.25	102.61	30.44	95.94	43.92	88.71	24.52	98.58	20.07
new_hi_vs_new_lo	29.63	91.88	37.23	100.13	31.50	91.80	41.20	82.58	22.64	86.92	15.60
advance_minus_decline	32.16	90.18	42.77	96.95	25.25	93.93	42.34	86.74	29.32	98.43	21.14
advance_decline_10day	32.17	91.35	40.97	96.23	28.35	92.50	50.89	82.27	24.71	94.84	15.92
updown_ratio	30.34	91.14	40.10	99.00	29.66	92.96	37.68	84.98	19.85	95.18	24.41
updn9to1	31.03	89.98	33.54	102.06	28.02	93.75	48.08	83.77	25.53	97.27	19.99
djua_vs_djia	11.19	88.82	16.33	97.29	17.14	88.91	6.48	81.03	5.73	91.75	10.28
churn	31.55	95.03	39.34	102.54	30.77	95.82	45.08	88.20	24.25	96.19	18.32
expdecay hilo	29.27	91.09	41.13	101.39	26.96	94.47	40.88	86.39	16.15	95.17	21.24
mvg avg 10 hilo	29.42	90.17	38.73	98.57	30.45	92.86	36.31	88.44	22.99	95.49	18.64
neg_vol_index	31.59	89.39	40.16	97.94	26.85	94.17	41.41	83.98	26.02	98.76	23.52
ntrin	28.70	88.59	29.23	95.21	29.16	92.63	40.28	86.97	21.20	96.59	23.64
mcclellan_osc	33.41	87.63	38.39	98.87	31.55	96.55	49.59	85.00	26.55	93.15	20.99
mcclellan_sum	31.55	94.76	38.34	99.02	32.81	96.12	39.24	84.15	23.61	97.91	23.77
on_balance_volume	30.81	88.44	40.02	99.55	28.14	94.13	43.18	82.89	24.46	96.64	18.27
stix	26.57	91.46	32.91	82.19	17.26	93.74	41.14	87.06	21.88	95.23	19.68
C2, no djua_vs_djia	44.07	74.23	55.14	76.04	17.72	71.84	47.01	70.18	76.72	81.78	23.78

All the results listed in these tables ignore commission costs. The results for the run which produced 53.69% return showed a 34.84% return with 0.1% commissions and a 10.07% return with 1.0% commissions. The effects of commissions decreased as the zero-commission returns decreased. The runs which showed little profit at the zero commission rate often showed slightly higher profits when tested with nonzero commissions, probably because those runs were changing investment levels at random, and when the commissions reduced the extent to which they responded to noise, they ended up either coming closer to a buy and hold strategy (which was usually better than the strategy they were following) or changed investment levels only when their expected gain was high and thereby filtered out some effects of overtraining.

In addition, I did tests where the input consisted only of windowed S&P500 values, and of the raw input files. In all cases, these approaches produce a buy and hold strategy, indicating that they were unable to see any pattern other than the long-term trend of rising stock prices.

I also did some tests with varying weight decay using cascade 2, and did not find any improvement.

4.5 Analysis of Results

While it was fairly easy to produce profits that exceeded a buy and hold strategy, it took substantial effort to exceed the hand-coded results. Since the successful network configurations used inputs that contained much of the information in the hand-coded approach, this was disappointing. It should be noted, however, that the hand coded approach was implemented using all the data as if it were a training set, with no attempt to test the results on an independent data set, since I had no reliable way of keeping my knowledge of various indicators' past performance from influencing my judgement. It is therefore unclear whether the hand-coded approach should be compared with the networks' performance on prediction sets or on training sets.

The stock market comes close to being an unpredictable time series, because the existence of predictability provides financial incentives to exploit that predictability, which changes the behavior in such a way as to reduce the magnitude of the predictable changes. Thus, my models which show good results on historical data might fail in the future because other people investing on the basis of similar approaches might eliminate the profits that had existed in the past.

There are several possible causes of market inefficiency:

- Emotions may prevent objective analysis.
- The complexity of the models needed for forecasting may exceed human understanding.
- Transaction costs.
- Some relevant information is not publicly available or requires substantial effort to find.

I had expected the first two of these to provide opportunities for a neural net to show small but definite profits. The fact that I got poor results until I discovered the importance of averaging the outputs and windowing the inputs indicate that the opportunities available from the first are quite limited. The fact that I have no practical way of determining what patterns the better configurations are finding in the inputs implies that the second effect is what my networks are exploiting.

The use of the average response of a number of networks was very important to overcoming the overfitting problem. Until I discovered that, I was unable to find the importance of windowing the inputs.

The fact that the good results appeared gradually as I approached the best configurations suggests that the networks were finding patterns in the data which are real (as opposed to the results of bugs in the software) and sufficiently unobvious that it is believable that other investors would have overlooked them.

The results of the "leave one indicator out" tests in the last table rule out the possibility that errors in any data other than the S&P 500 are causing the good results. The strong dependence on the djua_vs_djia indicator (which I can't explain) indicates that some of the profits shown are dependent on the accuracy of the DJIA and DJUA data, but the final cascade 2 test that was prompted by those concerns shows impressive results even without those data.

In spite of the fact that I have used a large set of data to train the networks, there is not enough data for the approaches that I initially decided upon to distinguish between many different sets of rules which are consistent with the data, many of which work only by accident.

“Lemma: Given any function f in a hypothesis class of r hypotheses, the probability that any hypothesis with error larger than epsilon is consistent with a sample of f of size m is less than $r * (1 - \epsilon)^m$.” [1].

While the exponential m in this expression may superficially appear to insure that a sample size in the hundreds or thousands will eliminate most accidentally correct hypotheses, the hypothesis space is also exponential in the complexity of the hypothesis (number of weights), and when mapping real-valued inputs to real-valued outputs, the number of distinct values that a weight can take have to create a new hypothesis can be large enough to make the hypothesis class quite large.

I initially expected that using the entire range of data available for some of the inputs (1928–1993) would be the best approach, since some interesting phenomena occur infrequently enough that shorter time periods will miss them (i.e. the 1929–1932 bear market was the largest decline and unlike most bear markets, interest rates declined sharply near the beginning rather than the end). However, after a fair amount of testing I decided that the large amount of missing data was having a negative effect (although the comparisons that I have done with time periods for which I have nearly complete data are inconclusive). If the network learns to depend on an input over the portion of the training set for which it is available, that may reduce its use of other inputs which are providing similar information, so that for the time period for which the former input is not available, the latter input provides an inadequate contribution to the final output, when it might have provided a larger contribution had the network needed to rely on it over the whole time period.

Alternatively, it may simply be that the data from the early parts of the period were too incomplete for the prediction set that was chosen from that range to have enough input data to make useful forecasts with any set of weights.

The results for the full 1928 to 1993 period show a strong positive correlation between the data available and the profits in the prediction periods. Only 10 of the raw inputs are available before 1945 (which includes all of the first period a prediction set and more than half of the first period b prediction set) versus 20 from 1960 on (which includes nearly all of the period d prediction sets and all of the period e prediction sets).

Here is the average annualized return on investment (percent) for all neural net tests on full 1928–1993 period in excess of the hand-coded results (prediction period only):

	period a	period b	period c	period d	period e
including GP	-24.0	-13.0	-13.4	+9.6	+4.1
Neural Nets only	-23.8	-14.9	-14.9	+10.6	+5.6
GP only	-25.2	-1.7	-4.2	+3.4	-5.3

The greater correlation of the neural net results with the availability of the data suggests that the former explanation is at least part of the explanation, as the *average* operator contained in the genetic program seed expressions caused the effective contribution of the available data to increase when some inputs were unavailable, because the average was taken of only the non-empty values for any given date.

The best results came from tests on the period starting in 1960. The few tests that I did with the period starting at 1945 (not shown) were slightly worse.

4.6 Specific Neural Net Algorithms

With averaging of the outputs and/or the best configuration of inputs, cascade 2 and recurrent cascade correlation showed the same superiority that they did with the Mackey-Glass and sunspot tests. Under less favorable conditions, the differences between algorithms reflected either random factors or differences in how readily the algorithms overfitted.

The ability of the linear associator to produce results very similar to backpropagation strongly suggests that hidden layers were not being used constructively in the tests done before I started averaging the outputs, and that the primary effect of training was to find direct correlations between individual inputs and outputs.

I have been unable to explain the superior performance of the SRN with the single network and high level indicator input, which is surprising in view of its more ordinary performance on the sunspot data, and in view of this quote:
"However, if I were interested in predicting earthquakes, stock market prices, or the weather, it strikes me as misguided to use an SRN. I'd go for the most powerful machine available." [5].

My best guess is that I have stopped it before it overtrained, thereby getting something closer to an arbitrary weighting of the preprocessed inputs than the most other algorithms. The fact that the training set error for these runs is lower than in many of the other tests tends to confirm this, although my measurements of the prediction error during training of the other algorithms do not show the increase that this hypothesis would suggest.

The 2-layer network organization did not help. In order for it to be advantageous, the networks in the first layer would have to have been able to pick out useful features in the subset of data available to them. Since little of the training in the comparable single-network configurations found useful features, it is not surprising that nothing useful came from the layered approach. I did not do any tests with 2-layer networks after discovering the advantages of averaging the outputs and windowing the inputs.

4.7 Genetic Programs

The genetic programs are less effective at following gradients because of the large element of randomness in the reproduction / mutation rules. Since the tasks that I have chosen appear to offer gradual improvements as the optimum rule is approached, this puts them at a disadvantage. I wish that I had more time to compare the effects of larger population sizes, different operators, and the ability to change the arity of nodes which take an arbitrary number of arguments. I suspect that my use in the seed expressions of a node which averages a large number of expressions without providing the arity-changing capability is limiting the diversity of the population and thereby precluding some good expressions from being explored. However, with the shortest of the tests that I reported taking well over an hour, I was unable to try the size and variety that I had hoped for.

Chapter 5

Conclusions

Recurrence is of some use for time series forecasting with neural networks. A well chosen set of inputs such as an input window which contains all the data that the network can usefully relate to the output, or a more heavily preprocessed set of inputs, is a lot more important.

Cascade 2 and recurrent cascade correlation produced the best results of the algorithms studied.

Neural networks can produce some improvements in stock market forecasts if good inputs are chosen and if the responses of a number of networks are averaged together to reduce overfitting. It is unlikely that the future performance of these models will be as good as the tests on historical data suggest, due to executions that don't match the ideal of the historical closing prices, and due to increased competition from other investors' use of improved technology, but the profits suggested are large enough that these problems are not likely to wipe them out entirely.

Appendix A

Genetic Program operators

A.1 Binary operations:

Operator	Effect on d1,d2
+	d1 + d2
–	d1 – d2
*	multiplies d1,d2
/	d1 / d2
is_above	if d1 > d2 then 1 else –1
average2	(d1 + d2) / 2
delay	replace d1[i] with d1[i – d2], d2 > 0
abs_diff	absolute value of (d1 – d2)
min	lesser of d1,d2
max	greater of d1,d2
moving_average	(sum of latest d2 non-empty values of d1)/d2
fraction_of_sum	d1/(d1 + d2)
reduce_abs	make d1 closer to 0 by up to d2
exp_decay	result = d2 * result + (1–d2) * d1
add_extend_empty	(d1 + d2)
convert_to_interval_of	[changes frequency at which data stored]
convert_rate_to_interval_of	[like above, with multiplication by frequency change]
last_cross	[–1 or +1 indicating direction in which d1 last]
	[crossed d2, or 0 if d2 not yet crossed]

A.2 Unary operations:

Operator	Effect on d1
change	$(d1[i] - d1[i-1])/d1[i]$ (first derivative)
annual_rate	$(d1[i] - d1[i - 1year])/d1[i]$
day_interval	[changes frequency at which data stored]
week_interval	[changes frequency at which data stored]
month_interval	[changes frequency at which data stored]
year_interval	[changes frequency at which data stored]
quarter_interval	[changes frequency at which data stored]
moving_average_200	200 day moving average
quarterly_to_weekly	$d1[i] * 7/4/365$
annually_to_weekly	$d1[i] * 7/365$
fill_empty_values	fills empty dates with the most recent non-empty value
normalize	d1[i] / maximum absolute value for d1
yearly_average	replaces all values for each calendar year
	with the average value for that year
limit1	clips values to range [-1,1]
log10	$\log_{10}$ (d1)
is_month	if(month = d1) then 1 else 0
is_year_mod_4	if((year modulo 4) = d1) then 1 else 0
make_zeroes_empty	if(abs(d1) $< 10^{-6}$) then empty else d1

A.3 List operations:

Add	sum
Average	average of all non-empty values

Appendix B

Raw Financial Data Available

(all data available through approximately April 2, 1993):

NAME	DESCRIPTION	FREQUENCY	AVAILABLE STARTING ON	SOURCE
sp500	S&P 500 index	daily	1/3/28	1,2
djia	Dow Jones Industrial Average	weekly	1/2/20	1,2
djua	Dow Jones Utility Average	weekly	1/8/60	1,2
bookval	Dow Jones Industrial Average book value	yearly	1929	3
upvolume	Volume of rising issues on the NYSE	daily	*11/19/79	3
downvolume	Volume of falling issues on the NYSE	daily	*11/19/79	3
volume	Total trading volume on the NYSE	monthly	1/1928	3,6,7
advances	Number of Advancing issues on the NYSE	daily	1/8/60	1,3
declines	Number of Declining issues on the NYSE	daily	1/8/60	1,3
unchanged	Number of Unchanged issues on the NYSE	daily	1/8/60	1,3
new_highs	Number of NYSE stocks making new highs	daily	*11/19/79	3
new_lows	Number of NYSE stocks making new lows	daily	*11/19/79	3

MONETARY

m2	M2 (money supply)	monthly	1/47	1,3
m3	M3 (broader money supply)	monthly	1/20	3,11
cpi	Consumer Price Index	monthly	1/45	1,3

ppi	Producer Price Index	monthly	1/45	1,3
napm	NAPM index of economic activity	monthly	1/82	2,3
indprod	Industrial Production	monthly	10/56	1,3,8
INTEREST RATES				
discount	Federal Reserve Discount Rate	irregular	11/1/20	2,4,6
tbill	Yield on 3-month Treasury Bills	weekly	1/6/28	1,3,4
fedfunds	Federal Funds Interest Rate	weekly	7/6/54	3,4
tbond	Yield on long term Treasury Bonds	weekly	1/18/28	1,3,4
LIQUIDITY				
freeres	Free Reserves	weekly	7/5/85	3
freeres	Free Reserves	monthly	1/29	4
indebt	Consumer Installment Debt	monthly	1/45	1,3
margin	Margin requirements for stock purchase	irregular	1934	4,10
INVESTOR SENTIMENT				
specshort	Specialist Short Sales	weekly	11/9/79	3
pubshort	Public Short Sales	weekly	11/9/79	3
bulls	Percent of Advisory Services Bullish	weekly	5/15/81	2,9
bears	Percent of Advisory Services Bearish	weekly	5/15/81	2,9
secondary_offers	Number of Secondary Offerings	weekly	*11/23/79	3

* significant gaps in the data in 1986 and 1987

Sources:

1. Molly's Economic Database, Marketbase Inc. 1989.
2. *Investor's Business Daily* 10/87 - 4/93.
3. *Barron's* 11/29/79 to 4/4/93.
4. Board of Governors of the Federal Reserve System, *Banking & Monetary Statistics*

1941-1970.

5. Board of Governors of the Federal Reserve System, *Banking & Monetary Statistics, The National Capital Press 1943.*

6. U.S. Department of Commerce, *Business Statistics 1961-88*;

U.S. Department of Commerce, *Survey of Current Business* Vol. 70 no 7. 1990.

7. Wigmore, Barrie A., *The Crash And Its Aftermath*, Greenwood Press, 1985. 8. info.umd.edu:/info/EconData/ 9. *Investor's Intelligence*, published weekly by Chartcraft Inc., Larchmont N.Y.

10. Fosback, Norman G., *Stock Market Logic*, Institute for Econometric Research, 1976.

11. Friedman, Milton and Anna G. Schwartz, *A Monetary History of the United States 1867-1960*, Princeton University Press, 1963.

Appendix C

Derived Indicators (Low Level Indicators)

Indicators computed from 1–3 types of raw values:

Monetary

yieldcurve = tbonds − tbill

realtbill = ppi_annual_rate month_interval − tbill
realtbond = ppi_annual_rate month_interval − tbonds

These remove the effect of inflation on interest rates to measure the extent to which they compete with stocks as an investment.

disc_trend = exp_decay (change (discount) .99))
m_expand = normalize (− (maxm2m3 + (ind_prod_annual_rate cpi_annual_rate)))

This measures whether economic growth is accelerating or decelerating.

disc_tbill = discount − tbill
fedf_tbill = discount − fedfunds

disc_change = exp_decay (change (discount) .99)
tbill_trend = exp_decay (change (tbill) 0.99)

discount_3month = is_above (discount moving_average (tbill 3))

These measure the trend of interest rates. The theory is that the market reacts to changes with some delay.

Liquidity

freereserves = freeres
installdebt = annual_rate (indebt)
installdebt9 = is_above (installdebt 0.09)
margin_change = exp_decay (change (margin) 0.995)
net_free = freereserves − moving_average (freereserves 80)

Fundamental

book_to_djia = bookval / djia
book_osc = average2 (last_cross (book_to_djia 0.8) last_cross (book_to_djia 0.5))
tbill_sp500yield = sp500yield_weekly − tbill

Trend

Trends, when not taken to unusual extremes, are generally assumed to continue more often than random. Changes in advance-decline ratios, volume, and new high/new low ratios have historically preceded changes in the market.

adv_minus_dec = advances − declines
tot_issues = advances + declines + unchanged
total_volume = change (upvolume + downvolume)
new_hi_vs_new_lo = moving_average ((new_highs − new_lows) 10)
advance_minus_decline = exp_decay ((advances − declines) 0.92)
advance_decline_10day = log10 (moving_average (advances/declines 10))
updown_ratio = log10 (moving_average (10 upvolume/downvolume))
updn9to1 = exp_decay (reduce_abs (updown_ratio 0.954) 0.99)

Measures unusually strong and broad market changes.

djua_vs_djia = exp_decay ((1.5 * change (djua) − change (djia)) 0.95)

The interest rate sensitive utility average tends to start long-term moves before the rest of the market. The factor of 1.5 helps to adjust for the lower volatility of the utility average.

sp500_vs_200day = is_above (sp500 moving_average_200day)

This measures the whether the long-term trend of the market is up or down.

churn1 = / ((abs_diff (advances declines)) tot_issues)

A market in which few stocks are moving supposedly indicates that expert investors are unloading stock in response to a steady supply of complacent new, inexperienced buyers. The same phenomenon does not occur at market bottoms because investors are panicking and constrained by liquidity problems.

min_new = (min (new_highs new_lows))
hilow1 = (/ (min_new tot_issues))
hilow_logic = moving_average (10 hilow1)

updown_ratio1 = log10 (/ (upvolume downvolume))

volume_decrease = make_zeroes_empty (
    max (0 − (0 is_above (change (upvolume + downvolume) 0))))
neg_vol_index = fill_empty_values (normalize (
    exp_decay ((volume_decrease * sp500daily_change) 0.99)))

trin = / (/ (upvolume advances) / (downvolume declines))
ntrin = log10 (trin)

This measures the volume weighted by the direction of movement.

breadth_ad = / (moving_average (advances 10) moving_average (declines 10))

mcclellan_osc = − (exp_decay (adv_minus_dec .9) exp_decay (adv_minus_dec .95))

mcclellan_sum = exp_decay (mcclellan_osc 0.99)
mcclellan_oscillator = last_cross (mcclellan_osc 0.0)
mcclellan_summation = last_cross (mcclellan_sum 0.0)

These measure short term extremes of the 19-day exponential moving average relative to the 39-day exponential moving average, with the expectation that the longer term trend measured by the latter is something the market will return to after extremes of the former indicate that the market has moved too far and fast in a direction.

obv = * (+ (upvolume downvolume) is_above (sp500daily_change 0))
on_balance_volume = is_above (obv exp_decay (obv 0.99))

This is based on the theory that changes on heavy volume are more indicative of future trends than light volume changes.

stix = exp_decay (advances / (advances + declines) 0.91)

Sentiment

specshort_ratio = log10 (specshort / publicshort)
secondary_offers
bears = bears / 100
bulls = bulls / 100

Seasonal

year in relation to presidential election cycle
month of year

Appendix D

High Level Indicators

Monetary (state of the economy)

yieldcurve
− 0.5 * normalize (maxm2m3)
+ 0.5 * (1 − realtbond)
+ 0.5 * (1 − realtbill)
+ 0.5 * disc_trend
+ 2.0 * disc_tbill
+ 0.5 * (1 − disc_change)
+ tbill_trend

Liquidity (availability of credit; not well distinguished from monetary)

2.5 * net_free
+ 0.5 * exp_decay (freereserves 0.9)
− 0.5 * installdebt9

Trend (stock price momentum; derived from stock price and volume data)

2.0 * sp500_vs_200day
+ exp_decay ((advances − declines) 0.99)
+ 0.5 * normalize (exp_decay (churn1 0.99) − .001)

+ 2.0 * ntrin
+ 1.5 * mcclellan_oscillator
+ mcclellan_summation
+ 2.0 * on_balance_volume
+ 0.5 * new_hi_vs_new_lo
+ 0.5 * advance_decline_10day
+ 0.5 * updown_ratio
+ 0.5 * updn9to1
+ 0.5 * djua_vs_djia
+ 0.5 * normalize (exp_decay (log10 (hilo1) .95))
+ 0.5 * normalize (moving_average (10 hilo2))

Investor Sentiment (measures whether the average investor is too optimistic or pessimistic)

1.5 * specshort_ratio – 0.5 * secondary_offers

Fundamental (does the market rationally discount the value of future earnings?)

normalize (book_to_djia)
+ 0.5 * book_osc
+ 1.5 * normalize ((sp500divs / sp500weekly) – tbill)

Appendix E

Hand Coded Expression

```
avg = average_list
(
0.2
* ( 1.5 sp500_vs_200day )
normalize ( / ( bookval sp500weekly ) )
* ( 1.0 tbill_trend )
* ( 1.0 yieldcurve )
* ( 1.0 normalize ( week_interval ( exp_decay ( - ( advances declines )  0.99 ) ) ) )
* ( 0.5 normalize ( week_interval ( - ( exp_decay ( churn1 0.99 ) .001 ) ) ) )
* ( 2.0 disc_tbill )
* ( 0.5 week_interval ( - ( 1 disc_change ) ) )
* ( 0.5 week_interval ( - ( 1 realtbond ) ) )
* ( 0.5 week_interval ( - ( 0 installdebt9 ) ) )
* ( 0.5 week_interval ( - ( 0 normalize ( maxm2m3 ) ) ) )
* ( 1.5 normalize ( - ( / ( week_interval ( sp500divs ) sp500weekly ) tbill ) ) )
* ( 2.5 net_free )
* ( 0.5 normalize ( exp_decay ( freereserves 0.9 ) ) )
* ( -1.0  neg_vol_index )
* ( 1.5 ntrin )
* ( 1.0 mcclellan_oscillator )
* ( 0.5 mcclellan_summation )
* ( 1.5 on_balance_volume )
* ( -0.5 stix )
)
```

Appendix F

Genetic Program seed expressions

```
avg1 = average_list
(
normalize ( / ( bookval sp500weekly ) )
* ( 1.0 tbill_trend )
* ( 1.0 yieldcurve )
* ( 1.0 normalize ( week_interval ( exp_decay ( - ( advances declines )  0.99 ) ) ) )
* ( 0.5 normalize ( week_interval ( - ( exp_decay ( churn1 0.99 ) .001 ) ) ) )
* ( 2.0 disc_tbill )
* ( 0.5 week_interval ( - ( 1 disc_change ) ) )
* ( 0.5 week_interval ( - ( 1 realtbond ) ) )
* ( 0.5 week_interval ( - ( 0 installdebt9 ) ) )
* ( 0.5 week_interval ( - ( 0 normalize ( maxm2m3 ) ) ) )
* ( 1.5 normalize ( - ( / ( week_interval ( sp500divs ) sp500weekly ) tbill ) ) )
* ( 2.5 net_free )
* ( 0.5 normalize ( exp_decay ( freereserves 0.9 ) ) )
* ( -1.0  neg_vol_index )
* ( 1.5 ntrin )
* ( 1.0 mcclellan_oscillator )
* ( 0.5 mcclellan_summation )
```

```
* ( 1.5 on_balance_volume )
* ( -0.5 stix )
)

avg2 = average_list
(
* ( 1.0 tbill_trend )
* ( 1.0 yieldcurve )
* ( 1.0 normalize ( week_interval ( exp_decay ( - ( advances declines )  0.99 ) ) ) )
* ( 2.0 disc_tbill )
* ( 0.5 week_interval ( - ( 1 disc_change ) ) )
* ( 0.5 week_interval ( - ( 1 realtbond ) ) )
* ( 1.5 normalize ( - ( / ( week_interval ( sp500divs ) sp500weekly ) tbill ) ) )
* ( 2.5 net_free )
* ( 1.5 ntrin )
* ( 1.0 mcclellan_oscillator )
* ( 1.5 on_balance_volume )
* ( 1.0 margin_change )
)

avg3 = average_list
(
* ( 0.5 normalize ( week_interval ( - ( exp_decay ( churn1 0.99 ) .001 ) ) ) )
* ( 2.0 disc_tbill )
* ( 0.5 week_interval ( - ( 0 installdebt9 ) ) )
* ( 0.5 week_interval ( - ( 0 normalize ( maxm2m3 ) ) ) )
* ( 1.5 normalize ( - ( / ( week_interval ( sp500divs ) sp500weekly ) tbill ) ) )
* ( 2.5 net_free )
* ( 0.5 normalize ( exp_decay ( freereserves 0.9 ) ) )
* ( -1.0  neg_vol_index )
* ( 0.5 mcclellan_summation )
* ( -0.5 stix )
)
```

Bibliography

[1] Blumer, Anselm, Andrzej Ehrenfeucht, David Haussler, and Manfred K. Warmuth, "Occam's Razor", in *Information Processing Letters* 24 (1987) pp 377-380, also reprinted in *Readings in Machine Learning* by Jude W. Shavlik and Thomas G. Dietterich.

[2] Colby, Robert W., and Thomas A. Meyers, *The Encyclopedia of Technical Stock Market Indicators*, Dow Jones-Irwin, 1988.

[3] Crowder, R. Scott, "Predicting the Mackey-Glass Timeseries With Cascade-Correlation Learning" in David S. Touretzky (ed), *Connectionist Models: Proceedings of the 1990 Summer School.*

[4] Elman, Jeffrey L., "Distributed Representations, Simple Recurrent Networks, and Grammatical Structure", *Machine Learning* 7, p. 195 (1991).

[5] Elman, Jefferey L., post to comp.ai.neural-nets, August 6, 1992.

[6] Fahlman, Scott E., "The Cascade-Correlation Learning Architecture", in *Advances in Neural Information Processing Systems* 2, Morgan Kaufman (1990).

[7] Fahlman, Scott E., "The Recurrent Cascade-Correlation Architecture", in *Advances in Neural Information Processing Systems* 3, Morgan Kaufman (1991).

[8] Fahlman, Scott E., Personal Communication (1993).

[9] Fang, Yan and Terrence J. Sejnowski, "Faster Learning for Dynamic Recurrent Backpropagation", *Neural Computation* 2, 270-273 (1990).

[10] Fosback, Norman G., *Stock Market Logic*, Institute for Econometric Research, 1976.

[11] Guimares, Rui M.C., Brian G. Kingsman and Stephen J. Taylor, *A Reappraisal of the Efficiency of Financial Markets*, Springer-Verlag 1989.

[12] Hertz, John, Anders Krogh, and Richard G. Palmer, *Introduction to the Theory of Neural Computation*, Addison-Wesley 1991.

[13] Kadirkamanathan, Visakan and Mahesan Niranjan, "A Function Estimation Approach to Sequential Learning with Neural Networks" svr-ftp.eng.cam.ac.uk:/reports/kadirkamanathan_tr111.ps.Z (1992).

[14] Koza, John R., *Genetic Programming*, MIT Press 1992.

[15] Mandelman, Avner, "The Computer's Bullish!: A Money Manager's Love Affair with Neural Network Programs", Barron's, Dec 14, 1992.

[16] Pollack, Jordan B., "The Induction of Dynamical Recognizers", Machine Learning 7, p. 227 (1991).

[17] Tong, Howell, *Threshold Models in Non-linear Time Series Analysis*, Springer, 1983.

[18] Utans, Joachim, and John Moody, "Selecting Neural Network Architectures via the Prediction Risk: Application to Corporate Bond Rating Prediction" in *Proceedings of The First International Conference on Artificial Intelligence Applications on Wall Street*, IEEE Computer Society Press 1991.

[19] Weigend, Andreas S., David E. Rumelhart, and Bernardo A. Huberman, "Back-Propagation, Weight Elimination, and Time Series Prediction" in David S. Touretzky (ed), *Connectionist Models: Proceedings of the 1990 Summer School.*

[20] Weigend, Andreas S., David E. Rumelhart, and Bernardo A. Huberman, "Generalization by weight elimination with application to forecasting", In R. Lippmann, J. Moody, and D. Touretzky (eds.) *Advances in Neural Information Processing Systems* 3, pp. 875-882, Morgan Kaufmann, 1991.

[21] Weigend, Andreas S., Bernardo A. Huberman, and David E. Rumelhart, "Predicting Sunspots and Exchange Rates with Connectionist Networks", in Martin Casdagli and Stephen Eubank (eds), *Nonlinear Modeling and Forecasting*, Addison-Wesley, 1992.

[22] Williams, Ronald J. and Jing Peng, "An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories", *Neural Computation* 2 p. 490 (1990).

[23] Williams, Ronald J. and David Zipser, "Experimental Analysis of the Real-time Recurrent Learning Algorithm", *Connection Science*, vol. 1, no. 1 (1989).

[24] Williams, Ronald J. and David Zipser, "A Learning Algorithm for Continually Running Fully Recurrent Networks", *Neural Computation* 1, p 270 (1989).