Knowledge Discovery for Time Series

Matthew John Saffell
B.S., LeTourneau University (1992)
M.S., University of Tennessee (1994)

A dissertation presented to the faculty of the
OGI School of Science and Engineering
at Oregon Health & Science University
in partial fulfillment of the
requirements for the degree
Doctor of Philosophy
in
Computer Science and Engineering

October 2005

The dissertation "Knowledge Discovery for Time Series" by Matthew John Saffell has been examined and approved by the following Examination Committee:

Dr. John Moody
Professor
Thesis Research Adviser

Dr. Todd Leen
Professor

Dr. Dan Hammerstrom
Professor
Portland State University

Dr. Melanie Mitchell
Professor
Portland State University

Dedication

To my parents, whose unconditional love has supported me through my entire life.

Acknowledgments

The work in this thesis was made possible through support from DARPA under Contract DAAH01-96-C-R026, AASERT Grant DAAH04-95-1-0485, the National Science Foundation under ITR grant NSF-0342634, and the generous support of the OGI Computer Science Department. Many thanks go to my advisor John Moody for the contribution of his time and guidance through the course of this work. Many thanks also go to the committee members for their insightful questions, comments, and feedback. Special thanks go to Yuansong Liao, Steve Rehfuss, Lizhong Wu, and Kyoungju Youn for assistance and collaboration on the research included here, and thanks to Aron Rempel for proofreading assistance and helpful comments.

Contents

List of Tables

List of Figures

Abstract

Knowledge Discovery for Time Series

Matthew John Saffell

Ph.D., OGI School of Science and Engineering
at Oregon Health & Science University
October 2005

Thesis Advisor: Dr. John Moody

My thesis investigates the use of machine learning methods for analysis of economic and financial time series. Since structural models in economics and finance are known to have limited predictive power, I study a data driven, time series approach to knowledge discovery in these domains. The ultimate goal of building predictive models of such time series is to support decision making in areas such as business, investing, and government policy.

Machine learning offers powerful tools for forecasting and decision making. Supervised learning methods can be used to develop forecasting models of economic series that can aid in decision support. Reinforcement learning methods can produce systems capable of making investment decisions. Hence, my thesis consists of two main investigations: a study of methods for predicting macroeconomic and financial time series, and a study of extensions to a reinforcement learning algorithm for constructing financial decision systems.

In the forecasting project, I develop a supervised training methodology for models that predict challenging macroeconomic and financial time series. I compare the performance of linear and nonlinear networks with a diverse set of standard linear benchmark models. While some advantage is obtained from the use of nonlinear networks for certain of these time series, a key result is that linear network models trained with stochastic, nonlinear neural network learning algorithms can achieve greatly improved performance over the benchmark methods on most of the data sets.

The second topic investigated is enhancements to the Recurrent Reinforcement Learning (RRL) algorithm. The RRL approach to trading system design has been shown to be effective at learning strategies that directly maximize financial objective functions, and also has been shown to outperform approaches based on supervised learning on artificial data sets. In my work, I investigate several significant extensions of RRL: to incorporate downside risk measures, to compare the RRL policy approach to an alternate RL value function approach, to extend the approach to portfolio management, and to conduct simulation studies on a number of artificial and real data sets, including an S&P-500 asset allocation system and a high frequency foreign exchange trader.

Chapter 1

Introduction

In this thesis, I investigate the use of machine learning methods, including supervised and reinforcement learning algorithms, to analyze and forecast economic and financial time series. My goal is to utilize and extend existing techniques and methods to produce algorithms and automated methodologies for building nonlinear decision making models in economics and finance. There are many theories regarding the behavior of the world's economies and financial markets, however most of these have limited predictive power [24]. The algorithms I present are data driven, nonlinear time series based approaches that produce quantitative forecasts and actions that result in measurable performance gains relative to standardly used models. As such, these algorithms have the potential to support real decision making in business, financial, and governmental institutions.

There is a dichotomy in the arena of macroeconomic and financial decision making: A) on the macroeconomic scale, possible actions to be taken are complex and multifaceted, and the outcomes of these actions have broad impacts, the desirability of which can be difficult to measure accurately; and B) on the scale of investment decisions, actions are easier to specify, and the desirability of outcomes is more easily measured. So to support macroeconomic decision making, I present a *Supervised Learning* (SL) methodology for forecasting with neural network models, where the forecasts can be used as an input to the decision making process. On the other hand, *Reinforcement Learning* (RL) techniques are more appropriate in the investment decision arena as the models make decisions directly, and can be trained to maximize the investor's utility.

Supervised learning is used to develop forecasting models that can aid in decision support. SL is used to extract a robust representation of the relationship between the past

and future values of macroeconomic and financial variables based on historical datasets. These monthly datasets are considered very challenging by standard time series measures, and are characterized by a very low signal-to-noise ratio and a high degree of nonstationarity. The series often exhibit non-normal distributions and heteroskedacticity (time-varying variance).

Reinforcement learning is used to produce systems that can make investment decisions. In RL, a goal-directed agent actively explores unknown environments, and attempts to maximize rewards received from the actions taken. RL is useful when the structure of the desired information (eg. optimal trading decisions) is unknown, but there is a performance measure which can be used to evaluate the agent's action. Based on performance feedback, the agent adjusts its actions and explores its environment in order to maximize its rewards.

In some cases, the feedback received by an RL agent can depend on a sequence of interdependent actions or can be substantially delayed from the associated actions that produced them. The need to assign credit among various actions over time is known as the *temporal credit assignment problem* [102]. This ability to interact with (and potentially influence) the environment, and having to deal with delayed feedback makes RL a fundamentally different type of learning problem from supervised learning.

1.1 Overview

In Chapter 2, I develop a supervised learning methodology for predicting challenging macroeconomic and financial time series. I present a methodology that addresses the challenges inherit in these series, and that automatically selects learning hyperparameters and chooses the model complexity. The methodology includes nonlinear, nonparametric models, sliding window retraining, various model regularization techniques, model complexity reduction, and bagging committee combinations to produce forecasts of the 1980–1989 test period. The application of this methodology is novel for neural networks in this problem domain. Previous studies [107] of this scope for these types of series have used relatively rudimentary neural network models that do not use regularization, and that use a stepwise model construction method that does not fully take advantage of the nonlinear

capabilities of the neural networks.

I present results for 3 and 12-month forecasts of both level and direction of the series using both linear and nonlinear neural networks trained using the presented methodology. These results are compared with a number of more standard linear and trivial predictors. I find non-trivial predictability in many of the series, and evidence in some of nonlinear structure. Also, both the linear and nonlinear network models outperform the more standard linear models including a Bayesian Vector Autoregressive model, which is also a regularized model. Another key result is that the linear network models perform almost as well as the nonlinear models for these series. This argues strongly for the use of appropriate linear models for comparison when fitting nonlinear models to these types of series. It could be that positive results in the literature from using nonlinear network models are due more to the estimation methodology than to actual nonlinear structure.

In Chapter 3, I investigate enhancements for the investment decision problem. I extend the Recurrent Reinforcement Learning (RRL) trading model of Moody & Wu [74] in order to make it more practically useful for investors. Some characteristics of investors are that they are more risk averse than standard risk measures would indicate, they typically hold diversified portfolio of assets and thus need to make optimal decisions in that context, and they are more comfortable making investment decisions when they can have an intuitive sense of the rules being used to make the decision.

To address these issues, I first implement downside performance measures for the model and examine the effect this has on trading performance. I find the RRL model to be capable of producing very desirable changes in behavior when using this modified risk measure. The systems trained using downside risk measures learn to cut their losses much more quickly than those using more traditional risk measures. I also present results for the single asset trader on a number of real financial datasets including an S&P-500 stock index asset allocation system and an intra-daily foreign exchange trading system. The models are very successful in finding tradeable structure in these series. These models are then examined using sensitivity analysis to gain insight into their operation. I also provide a discussion of the difference between representing policy functions and value functions, and present a comparison of the RRL model with a value function model on the S&P-500

stock index asset allocation problem. Finally, I extend the single-asset model to manage a portfolio of assets. I find the systems perform well in simulations using artificial portfolios. The systems shows the ability to produce allocations that take into account the differing predictabilities of the underlying assets.

In Chapter 4, I present a summary of the work and discuss some possible extensions. Additionally, Appendix A contains tables of forecasting results for the series forecasted in Chapter 2.

Chapter 2

Knowledge Discovery Through Supervised Learning

2.1 Introduction

In this chapter, I investigate the application of nonlinear estimation methods for linear and nonlinear predictive models for a spectrum of U.S. financial and macroeconomic time series. These series are very important measures of economic growth and the health of the economy. Forecasts of such macroeconomic variables play a key role in making decisions and setting policies in many arenas, from the level of international organizations down to the level of individual corporations. They are also very challenging series to forecast, and are characterized by very low signal-to-noise ratios and a high degree of nonstationarity. The data series are short, being comprised of monthly data for a 45 year period, and the potential feature set is very large. These characteristics pose special challenges for predictive modeling.

The typical econometric approaches to dealing with these challenges are to use a linear statistical model or a nonlinear, structural model based on some economic hypothesis. The approach I take here is based on nonlinear, nonparametric statistical models. These models attempt to learn the predictive function from the data. Due to the inherent flexibility of these models, general constraints and restrictions are used during the training process to help prevent overfitting. The degree of constraint in the final model structure is determined by the data itself rather than being specified in an ad hoc manner.

The goals of this research are: (1) to develop a methodology capable of dealing with

the challenges mentioned above, (2) to examine the predictability of both point and directional forecasts of monthly time series, (3) to test for the presence of nonlinearity in the series, and (4) to compare standard linear estimation methods to state-of-the-art methods developed in the adaptive systems and machine learning literatures. The techniques I consider include data-based selection of hyperparameters, early stopping of training, weight decay regularization, year-by-year retraining, Principal Component Pruning for model complexity reduction, and bagging committees.

I find nontrivial predictability in making point forecasts for many of the series studied. For several, the results show some advantage from the use of nonlinear models over linear models when using the same sophisticated estimation methods. The Index of Industrial Production and the Housing Starts Index show the most advantage from the use of nonlinear versus linear models. I also find that the application of nonlinear neural network learning techniques to a linear network model considerably improves the performance relative to standard linear models such as least squares regression, univariate AR, and Bayesian vector AR models. When making directional forecasts, I find the output representation of the data during training to have a substantial impact on the learned model's accuracy. Also, for the series in this study, training more complex models to predict quintiles of directional movement does not improve performance relative to simpler models trained to make point forecasts.

I start in Section 2.2 by introducing some notation and nomenclature related to regression and time series analysis that will be used later in the chapter. In Section 2.3 I introduce the macroeconomic time series domain and discuss the issues that make it challenging for forecasting, and review the related literature. Section 2.4 presents the methodological approach used to create the linear and nonlinear network forecasting models. Section 2.5 describes the benchmark models that will be compared against. In Section 2.6 the implementation details of the models and datasets are described. In Sections 2.7 and 2.8 I present the results from the forecasting experiments, and in Section 2.9 I review the findings presented in the chapter.

2.2 Regression and Time Series Analysis

In this chapter, data sets are represented as time series. That is, the data are a series of measurements of some variable measured sequentially through time. The process that creates these measurements is the *data-generating process.* A data generating process (DGP) is a process that takes some number of sources (including purely random signals) and produces a value that can be measured. For example, a generic DGP could have the form

$$y_i = g(\mathbf{x_i}, \epsilon) \tag{2.1}$$

where the output, y_i, is a function of the input vector, $\mathbf{x_i}$, and some random noise variable, ϵ. There could be multiple sources of noise, as is common in state space models. A *dataset* consists of some number of pairs of inputs and outputs $\{\mathbf{x_i}, y_i\}$, $i = 1, 2, \ldots, K$. The DGP typically embodies some relationship between the inputs and the outputs. That is, even though there may be a stochastic element, a given set of values for the sources will produce some predictable change in the measurement due to the deterministic element of the data generating process.

In a typical regression problem, there is not necessarily any significance to the ordering of the data in the dataset. That is, the data point with index i is not necessarily related to the data point with index $i - 1$. However, in a time series there is a clear ordering of data points based on the times at which the measurements were recorded. The value of an observation measured at time t, y_t, may well be related to or explicitly dependent on previously measured values such as y_{t-1}. These types of causal relationships will have implications on the way such problems are modeled.

2.2.1 Time Series Modeling

A time series whose future values can be exactly predicted from its historical values is said to be *deterministic.* Most series of interest to forecasters however cannot be fully predicted in this way, and are called *stochastic* processes as they have some random or unknown component that influences their future behavior.

Autoregressive Moving Average Model

The principal stochastic time series model is the *autoregressive moving average* (ARMA) process. A time series is an autoregressive process of order p, denoted $AR(p)$, if the DGP is represented by

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_p y_{t-p} + \epsilon_t \tag{2.2}$$

where the α_i are constant-valued parameters representing the relationship among the lagged values of y_t, and ϵ_t is a random variable. When a model is *univariate*, that is only containing lagged values of itself as inputs, it is usual to use y instead of x to represent the inputs. A time series is a moving average process of order q, denoted $MA(q)$, if the DGP is represented by

$$y_t = \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + \ldots + \beta_q \epsilon_{t-q} + \epsilon_t \quad , \tag{2.3}$$

where the β_i are constant-valued parameters representing the relationship among the lagged values of the noise variable ϵ_t. By extension, the $ARMA(p, q)$ process is represented by

$$y_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_p y_{t-p} + \beta_1 \epsilon_{t-1} + \beta_2 \epsilon_{t-2} + \ldots + \beta_q \epsilon_{t-q} + \epsilon_t \tag{2.4}$$

For a time series to be *stationary*, all transition probabilities from one state to another must be independent of time. A weak form of stationarity commonly cited in linear time series analysis requires that the first two moments of the series' returns distribution exist and that they be constant over time. For example, an $AR(1)$ process

$$y_t = \alpha_1 y_{t-1} + \epsilon_t \tag{2.5}$$

is nonstationary if $|\alpha_1| >= 1$ because the variance of the series is not constant over time.

A time series is said to have a *unit root* if it is nonstationary, but has a differenced series that is stationary. For example, the AR(1) process with $\alpha_1 = 1$ is a random walk; y_t has mean y_0, but nonstationary variance $t\sigma_\epsilon^2$. A generalization of this process may include a trend:

$$y_t = y_{t-1} + \mu + \epsilon_t \tag{2.6}$$

This series is a random walk with drift and has a nonstationary mean of $y_0 + t\mu$. However, the differenced series

$$\Delta y_t = y_t - y_{t-1} = \mu + \epsilon_t \tag{2.7}$$

is stationary in mean and variance. Many of the economic time series discussed in this chapter contain trends or unit roots. This is one of the main motivations for forecasting the changes in the series (returns) rather then the levels themselves.

Linear Vector Autoregression

The standard linear vector autoregression (VAR) model extends the AR model to a multivariate setting:

$$\mathbf{Y}_t = \sum_{k=1}^{p} \mathbf{A}_k \mathbf{Y}_{t-k} + \epsilon_t \tag{2.8}$$

where $\mathbf{Y}_t$ is a vector of n time series y_i measured at time t, A_k is a matrix of parameters α_{ijk}, p is the number of time lags to include, and ϵ_t is a vector of *iid* noise with variances σ_i^2. To be explicit, Equation 2.8 represents a set of n equations:

$$y_{it} = \sum_{k=1}^{p} \left[\alpha_{iik} y_{i(t-k)} + \sum_{j \neq i} \alpha_{ijk} y_{j(t-k)} \right] + \epsilon_{it} \quad . \tag{2.9}$$

The parameters of the VAR can be estimated using the standard ordinary least squares estimation method. Some limitations of this framework occur when the number of variables and the number of time lags included in the model begin to grow. The amount of data needed to estimate the parameters becomes very large very quickly.

2.2.2 Nonparametric Regression and Generalization

Stated simply, the goal of nonparametric regression is to learn a relationship between two sets of variables. That is, given a set of data pairs, $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_K, y_K)$, where $\mathbf{x}$ is an n-dimensional vector of "inputs" and where y is (without loss of generality) a scalar "output", a function of x,

$$y_t = g(\mathbf{x}_t) + \eta_t \quad , \tag{2.10}$$

and η_t is a zero-mean random variable. We wish to learn a representation, $f(\mathbf{x})$, of the relationship, $g(\mathbf{x})$, that takes values of $\mathbf{x}$ to values of y. The learning of the model, $f(\mathbf{x})$, involves setting the values of a set of parameters, θ, in response to the observed training data set, $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_K, y_K)\}$. Following the notation in [35] we emphasize the dependence of the model on the training data by writing $f(\mathbf{x}; \mathcal{D})$ instead of $f(\mathbf{x})$.

A typical measure of how well a model represents a relationship is the *mean-squared error* (MSE) measure:

$$E_{\mathcal{D}}\left[(y - f(\mathbf{x}; \mathcal{D}))^2 | \mathbf{x}, \mathcal{D}\right] \quad , \tag{2.11}$$

where $E_{\mathcal{D}}[\cdot]$ denotes the expected value over possible datasets, $\mathcal{D}$. A straightforward learning algorithm would involve adjusting the model parameters, θ, to minimize the MSE measured on the available training data.

Because the individual values of y in the dataset typically contain noise, it is of interest to look at the value, $E[y|\mathbf{x}]$, which is the average value of y conditioned on x. It is easy to show that $E[y|\mathbf{x}]$ minimizes MSE for any given value of $\mathbf{x}$, and also has the property of being independent of any realization of $\mathcal{D}$. In the following sections we will be interested in looking at the MSE defined using this "best" estimate

$$E_{\mathcal{D}}[(f(\mathbf{x}; \mathcal{D}) - E[y|\mathbf{x}])^2] \quad . \tag{2.12}$$

The term *generalization* refers to how well the model $f(\mathbf{x}; \mathcal{D})$ is able to respond to values of $\mathbf{x}$ that were not contained in the training data set $\mathcal{D}$. The main factors that affect generalization ability are the amount of noise present in the training data set, the number of available points in the training data set, the coverage of the training data set relative to the range of the DGP, and the flexibility of the model being fit to the data. When the flexibility of a model is such that it can be trained to a low error on $\mathcal{D}$, but has a high error on similar but previously unseen data produced by the DGP, we say that the model has been *overfit*. Overfitting occurs when the model's flexibility allows it to model features produced by the noise in the training dataset. In this chapter, I will describe techniques such as regularization and Principal Component Pruning that can be used to reduce the effective number of parameters in a model, thus improving the generalization ability of a model.

2.2.3 The Bias / Variance Tradeoff

The *bias / variance tradeoff* is an old idea in statistics, and it describes the ways in which model flexibility contributes to the prediction error produced by a fitted model. The essence of the analysis is to decompose the estimation error into two components, a bias component and a variance component.

$$
\begin{aligned}
& E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E[y|\mathbf{x}])^2] \\
&= (E_{\mathcal{D}}[f(\mathbf{x};\mathcal{D})] - E[y|\mathbf{x}])^2 \quad \text{“bias”} \quad + && (2.13) \\
& \quad E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D}) - E_{\mathcal{D}}[(f(\mathbf{x};\mathcal{D})])^2] \quad \text{“variance”} && (2.14)
\end{aligned}
$$

$E_{\mathcal{D}}[\cdot]$ denotes the expectation over all possible datasets, $\mathcal{D}$, of length K. So a model, $f(\mathbf{x};\mathcal{D})$, is said to be *biased* as an estimator of $E[y|\mathbf{x}]$ if on average it is different from $E[y|\mathbf{x}]$. Even if $E[y|\mathbf{x}]$ is unbiased, the MSE can still be large due to the *variance*, that is the model is sufficiently flexible that any single model can be very different from the average model. Relevant references include [33, 116, 35].

2.2.4 Noise & Nonstationarity

The *noise / nonstationarity tradeoff* was first noted by Moody [62] (see also [66, 67]). Similarly to the *bias / variance tradeoff*, the *noise / nonstationarity tradeoff* for nonstationary time series can be described by decomposing MSE into two components. The “noise” component represents the contribution to the error coming from the variance caused by having too short of a history on which to train. The “nonstationarity” component represents the portion of the error caused by having too long of a history, and thus including information that is more misleading than helpful to the model fitting process.

2.2.5 Measuring Nonstationarity

One method for measuring the effects of nonstationarity is to train models using a variety of training window lengths and then measure the test set errors for each model. Figure 2.1 shows test set error vs. training window size for one of the macroeconomic series investigated in the remainder of this chapter. In this case the error is high due to noise effect

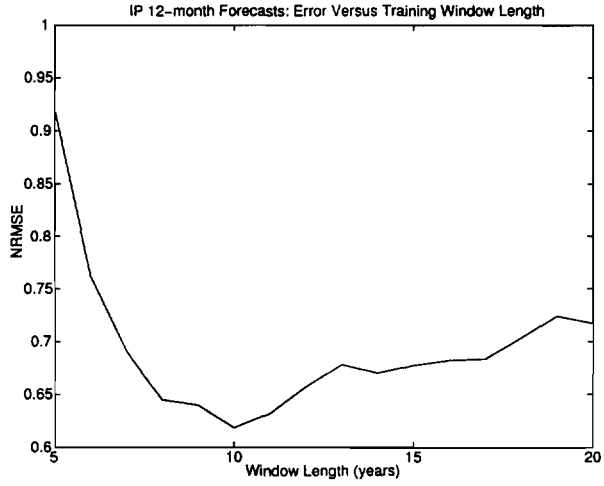

Figure 2.1: A graphical depiction of the *noise / nonstationarity tradeoff* [62, 66, 67]. The figure plots normalized mean-square error versus training window length for forecasts of the 12-month future log returns of the Index of Industrial Production from 1980-1989. Large error values occur for short window sizes due to large amounts of noise in the dataset. Error initially decreases as the training window size is increased, but then errors start increasing again as the effects of nonstationarity become evident.

from the short training window sizes. As the training window lengthens, error decreases due to the increasing number of data points available for model fitting. However, as the window length increases even more the error begins to rise again. This increase in error is due to the fact that the additional data points being used for model fitting are becoming less and less relevant to the data points that are in the test set. That is, the data generating process has evolved sufficiently over time so as to render data beyond a certain age useless for the task of more accurately learning the functional form of the current data generating process.

In this thesis I also use sensitivity analysis to demonstrate the effect of nonstationarity on the learned models. The sensitivity analysis is described in Section 2.7.5, and allows us to open the neural network "black-box" to a certain degree and to examine the changing relationships between the inputs and outputs. I find that the relationships learned by the models trained to forecast the datasets considered can change significantly over the course of the test period.

2.3 Problem Domain: Macroeconomic Forecasting

Macroeconomics is the study of economic behavior in the aggregate. A primary concern of macroeconomic analysis is sustainable economic growth. Related to economic growth are the allocation of resources, employment of the workforce, stability of prices, and relationships between national economies. A large number of time series exist which are updated periodically in an attempt to quantify the variables used to describe the macroeconomy.

Examples of macroeconomic variables include the gross domestic product, the inflation rate, and the unemployment rate. Often the values of these variables are estimated using statistical sampling or survey techniques. Figure 2.2 shows another macroeconomic variable, the Index of Industrial Production (IP), for the time period 1967-1993. IP is a monthly time series that measures the physical volume of output of the manufacturing sector. The shaded bars in the figure show the periods when the U.S. economy was officially in a recession. The correspondence between downturns in IP and the recessions is clear. Accurate forecasts of IP would be very useful for predicting recessions, and could allow for the adjustment of monetary and political policy to ameliorate or even prevent the recession.

Forecasts of macroeconomic variables play a key role in setting policies in many arenas, from the level of international organizations down to the level of some individual corporations. I find evidence of predictability in several of the macroeconomic series considered in this chapter including the Index of Industrial Production.

2.3.1 The Time Series

In this chapter, I consider a number of macroeconomic and financial data series for prediction and classification. The ten monthly series for which I produce results are listed in Table 2.1 and include the industrial production index, the housing starts index, the unemployment rate, the consumer price index (a measure of inflation), money supply, short and long-term interest rates, and the S&P 500 stock index. I include the 3-month treasury bill rate as an input, though do not produce predictions for it due to the amount of similarity between it and the Aaa bond yield.

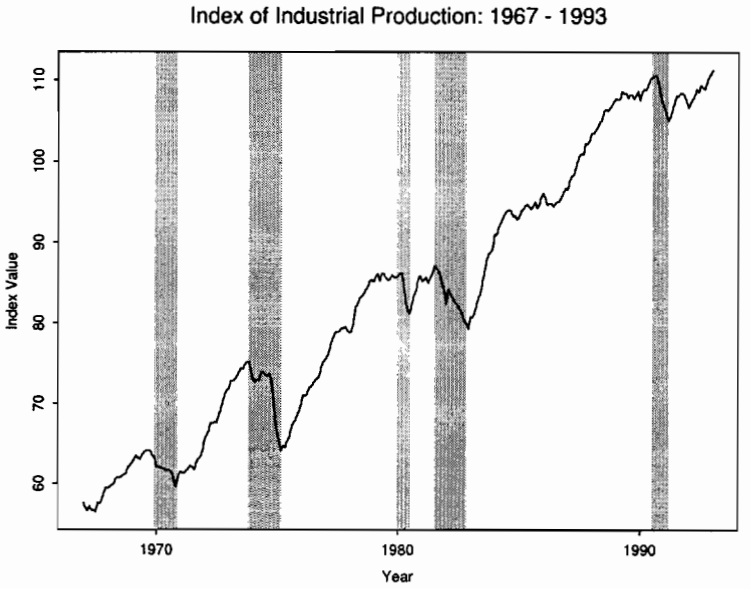

Figure 2.2: The Index of Industrial Production is a monthly series that measures the physical volume of output of the manufacturing sector. There is a strong correspondence between downturns in the Index of Industrial Production and recessions in the US economy (shaded periods).

Table 2.1: Data series are taken from the Citibase database for the period 1950 - 1995. All names except "DRM" and "YCS" are those given in the Citibase database. The notation 1987 = 100 indicates that the series is normalized to that base year.

Designation	Description
DLEAD	Index of Leading Indicators. Seasonally adjusted. 1987 = 100.
DRM	Default Risk Measure: (Baa Bond Yield) - (Aaa Bond Yield).
FM2DQ	M2 Money Supply
FSPCOM	Standard & Poor's 500. 1941 to 1943 = 10.
FYAAAC	Moody's Aaa Bond Yield.
FYGM3	3-month Treasury Bill Yield.
HSBP	Housing Starts. Seasonally adjusted. 1967 = 100.
IP	Index of Industrial Production. Seasonally adjusted. 1987 = 100.
LHUR	Unemployment Rate. Seasonally adjusted.
PUNEW	Consumer Price Index. Seasonally adjusted. 1982 to 1984 = 100.
YCS	Yield Curve Slope: (10-Year Bond Composite) - (3-Month Treasury Bill).

I construct both linear and nonlinear multivariate models to produce forecasts of these variables on 3 and 12 month horizons. The available dataset covers 1950–1995 and is of monthly resolution. Figure 2.3 shows examples of several of these series along with other indicative macroeconomic series.

The set of inputs to the predictive models is based on the same set of macroeconomic and financial time series listed in the table. Various transformations were applied to each of the input series, creating a large number of inputs from which to choose. One transformation involves taking the logs of the series to make the variances of the price changes more stationary. The other involves taking the differences of the series on telescoping time scales. This improves the performance of the model fitting process by removing nonstationarity due to unit-roots, reducing the noise in the input signals, and representing pertinent time scales returns as individual inputs. See Section 2.6.2 for the details of the transformations.

2.3.2 Challenges

Macroeconomic modeling and forecasting is challenging for many reasons [62, 25]. The lack of convincing *a priori* economic models [24] creates a severe model selection problem. Also, the series themselves display various characteristics which pose difficulties for standard modeling techniques, including high levels of noise, nonstationarity, and possible nonlinear structure. I will briefly discuss these issues in more detail.

No *a priori* models

Convincing and accurate scientific models of business cycle dynamics do not yet and may never exist. The U.S. economy is too large and too complex to ever measure exactly. There are many factors that influence the economy that cannot be measured or quantified including mass psychology and sociological effects. It is estimated that anywhere from 3-44% of the U.S. economic activity is conducted in a deliberately concealed manner [101]. In the face of these problems, two main approaches that economists use to model the macroeconomy are structural models and linear time series models.

Structural models are based on explicit economic theory, and are often nonlinear in

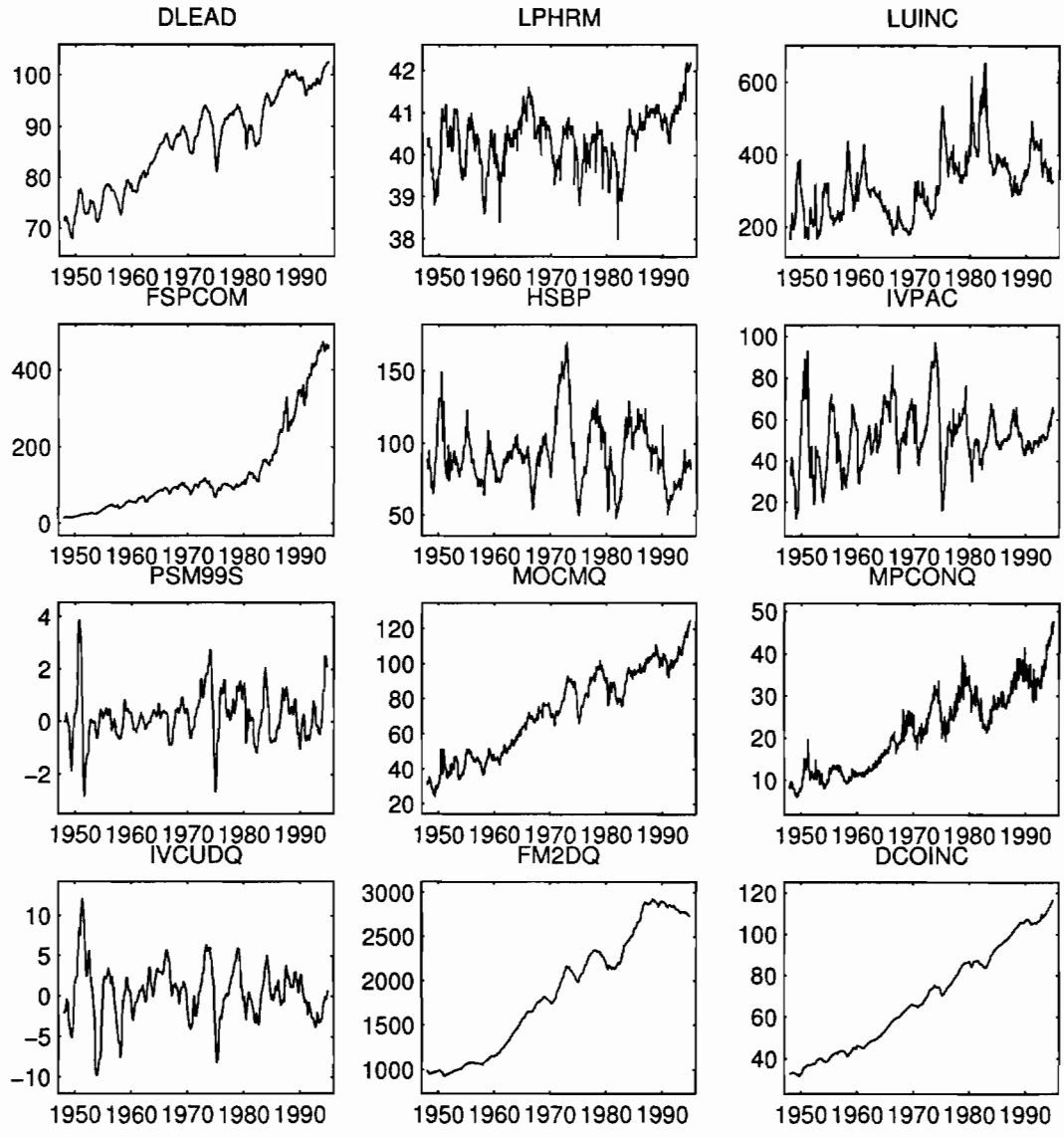

Figure 2.3: The U.S. Index of Leading Indicators (DLEAD) and its 10 component series as contained in the Citibase dataset. The Leading Index is a key tool for forecasting business cycles. The difficulty of macroeconomic forecasting is evident, due to the high noise levels and erratic behaviors of DLEAD and its components. Of the series shown here, the DLEAD, HSBP, FM2DQ, and FSPCOM series are used for prediction and classification. (Note that the component series included in DLEAD have been changed several times during the past 50 years.) Also shown is the Index of Coincident Indicators (*COINC*), another composite series which is meant to track the business cycle rather than lead it.

nature. These models attempt to model the macroeconomy at a relatively fine scale and can contain hundreds or thousands of equations and variables. The model structures are chosen by hand, as is the specification of endogenous and external variables. The model parameters can be estimated from the data, though their values are usually constrained by the underlying economic theory [56]. While structural models can be useful for understanding the economy qualitatively, they are notoriously bad at making quantitative predictions. The popularity of these model tend to rise and fall with the currently prevailing macroeconomic theory [24].

Given the poor forecasting performance of these structural models, many economists have taken to forecasting economic activity using the statistical techniques of standard linear time series analysis such as the Vector Autoregression (VAR). These time series models typically have only half a dozen to a dozen input series if they do not include some prior assumptions to guide the model fitting process. The most reliable of this type of statistical model during the past two decades or so have been Bayesian Vector Autoregressive (BVAR) models [51]. The BVAR model is a variation on the standard linear VAR model which allows the imposition of prior information or belief about model structure on the model building process. See Section 2.5.4 for an overview of the BVAR model.

The research presented here will show that neural networks and neural network techniques can outperform standard linear time series models. The lack of an *a priori* model of the economy makes input variable selection and network model complexity critical issues. See Section 2.4 for a description of the methods employed in this chapter.

Noise

Figure 2.3 shows examples of several macroeconomic time series. As can be seen, these series are very noisy and generally have poor signal to noise ratios. The noise is also apparent in Figure 2.4, which displays the logged Industrial Production series along with returns of the series on multiple time horizons. Among other sources, the noise is due to the many unobserved variables in the economy and to the survey techniques used to collect data for those variables that are measured [101]. Due to the complexity of

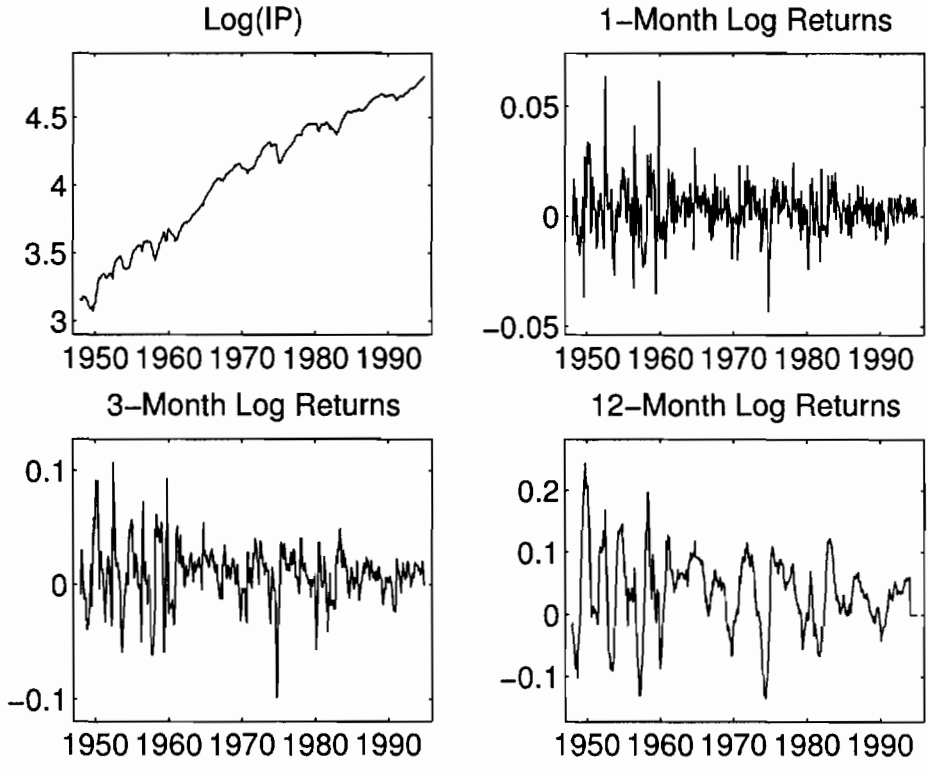

Figure 2.4: The U.S. Index of Industrial Production and several return series (rates of change measured as log returns) for the 1, 3, and 12-month time scales. The difficulty of the prediction task is evidenced by the poor signal to noise ratios and erratic behavior of the target series. For all returns series, significant nonstationarities and deviations from normality of the noise distributions are present.

gathering the large amounts of data needed to even estimate some of these economic variables, the values are often still being revised for years after the initial estimates are published. A short discussion on the use of revised data in macroeconomic forecasting is given in Section 2.3.3. Some features of the noise distributions that complicate the model fitting process are the presence of heavy tails and outliers. The significant noise levels in combination with limited data histories makes controlling model variance, model complexity, and the *bias / variance tradeoff* (discussed in Section 2.2.3) important issues.

Nonstationarity

Macroeconomic series are intrinsically nonstationary due to changes in governmental policy and innovations in business techniques. For example, the advent of computing technologies over the past 20 years has radically changed the nature of business worldwide. Other sources of nonstationarity include changing noise distributions, periodic redefinition of many series, and changing measurement techniques. In general, the presence of nonstationarities will shorten the usable length of the data series, since the older data will be less useful for predicting the future. The combination of noise and nonstationarity gives rise to a *noise / nonstationarity tradeoff* [62] (see Section 2.2.4). If training windows are made too short in an attempt to use only the most relevant data, the model variance will be high due to the noise in the remaining limited training data. Increasing the length of the training window will help mitigate the effects of noise, but if the window becomes too long, error will begin to rise due to the effects of nonstationarity.

Nonlinearity

Nonlinear theories and models of the economy have existed for quite a long time [24]. However, until relatively recent history, the lack of computational power has limited statistical analysis to the linear domain [108]. Hence, traditional macroeconomic time series models are linear [37, 40]. Recent work as well as the current work suggests that nonlinearities can improve macroeconomic forecasting models in some cases [38, 68, 79, 106]. Often the degree of nonlinearity captured by neural network models of macroeconomic series tends to be mild [68, 50, 89, 114, 127]. The current research suggests that simpler models can be favored due to the high noise levels and limited data. Also, weight-decay techniques for models using sigmoidal nonlinearities bias the fits toward linear structure, making reliable estimation of nonlinearities more difficult.

2.3.3 Literature Review

There are numerous texts on economic and econometric time series forecasting [61, 32, 34], with one of the more seminal works being Granger and Newbold [37]. There are many decent texts with introductions to macroeconomic issues and economic indicators [83, 88,

101]. The sections on the leading indicators are of particular relevance to the forecasting approach taken in this chapter.

Historical Highlights

Early attempts at modeling the economy involved using "structural models". Structural models interpret economic data using a particular economic theory [24]. These models can be quite large, involving hundreds of variables. The use of specific structural models tends to rise and fall based on the fortunes of the underlying economic theory. Nonstructural models, being less tied to a specific theory, have been gaining in popularity over the years, and continue to progress at a rapid pace. Diebold [24] states that the popularity of nonstructural models for macroeconomic forecasting began to grow in the 1970s as the structural models based on Keynesian theory began to fall out of favor, though certainly nonstructural models existed prior to this period. Some of the earlier work on nonstructural models of economic forecasting was done in the 1920s by Slutsky and Yule who suggested that autoregressions were appropriate. Box and Jenkins ARMA-fitting methodology [13] was a very influential work in the 1970s on nonstructural time series analysis and forecasting. Vector autoregressions [128] are important advances around this time as well.

In 1986, McNees [56] compared the performance of the BVAR forecasts produced in real time by Litterman with several prominent forecasters of the time. McNees finds that the BVAR-generated forecasts present a strong challenge to conventional practices and serve as a powerful standard of comparison for other forecasts. McNees also notes that the forecasts produced by structural models are often adjusted by the model user to take into account the user's own judgment of the most likely outcome. BVAR forecasts are typically not adjusted by the forecaster, showing a difference in the philosophical positions of the forecasters regarding the role of empirical models in the social sciences.

Swanson and White [107, 106] perform an extensive comparison of econometric forecasting models including artificial neural networks on a set of nine macroeconomic series. Since they used quarterly data, they were able to compare the performance of their models with the forecasts produced by the Society of Professional Forecasters, a number of

professional forecasters from various professions, whose forecasts are recorded every quarter in real time. One of the questions Swanson & White address is whether adaptability in linear and nonlinear models is useful. They also take a model selection approach to the prediction problem as opposed to a hypothesis testing approach. One of their main findings is that their multivariate adaptive linear models outperform nonadaptive models, as well as the adaptive nonlinear models they considered and the forecasts of the Society of Professional Forecasters. They also find that various models are preferred depending on the performance measure, emphasizing the importance of model selection criteria.

Data Vintages

The problem of *data vintages* is usually ignored when macroeconomic series are considered. Macroeconomic series can never be truly, accurately measured, so, as discussed previously, the values reported are at best estimates. As time passes since the initial estimate of a series is made, more accurate estimates of the data point are often able to be made, and are used to replace the initial estimates. Swanson and White [107] use unrevised, or "first reported" quarterly data which allows them to produce true *ex-ante* forecasts. However, this dataset was constructed by hand by searching through over 30 years of monthly issues of the "Survey of Current Business". Of course, even this type of dataset is incomplete by itself. To make "optimal" *ex-ante* predictions one would like to use maximally revised datasets that do not violate causality constraints [90]. This would correspond to having a full, unique dataset at each time step containing the most recently available revision of every historical data point. The "first reported" dataset used in [107] will have an exceptionally high level of noise due to the use of the initial estimates only at all points in time. Also, presumably the errors are being calculated on these same noisy "first-reported" data points. As the amount of revision can often be large, this leads to questions as to the "true" accuracy of any error measure being calculated using this type of dataset. Diebold and Rudebusch [26] use a partially revised, causal dataset to build linear models for predicting the Composite Leading Index. They find a significant degradation in their linear models when switching from a fully revised to a partially revised dataset.

Due to costs and availability I use fully revised monthly time series. The reported data

is not actually publicly available, and substantial effort would be needed to attempt to collect it. Also, since the emphasis of this work is on model selection and the evaluation of the presence of nonlinearity in the series, I feel it is appropriate to use full revised data. This should allow a clearer picture of the actual underlying relationships present in the datasets.

2.4 Modeling Approach

In this section I discuss the various techniques that are used when training the networks to help deal with the problems of model selection, noise, nonstationarity, and nonlinearity described in Sections 2.1 and 2.3. These techniques include year-by-year retraining, model regularization, variable selection, and committees of models. Often there are a number of related techniques that could be used to achieve similar effects. This work does not try to compare techniques, but rather seeks to demonstrate the need for techniques such as those presented here. It is likely that different techniques will be preferred depending on the specific datasets under consideration.

2.4.1 Neural Network Model Architectures

In order to model the possible nonlinear structure in the macroeconomic time series, I use nonlinear models with sigmoid-type nonlinearities. These nonlinear models have been studied extensively, and further discussion beyond the scope of this thesis can be found in a number of references [42, 41, 36]. This section describes the nonlinear models used in this project, and provides a brief overview of the training process.

The model architecture used is a two-layer feed-forward neural network with *tanh* activation functions in the hidden layer, and a single linear output. The output of an individual network is given by the network equation:

$$\hat{f}_{net} = v_0 + \sum_{j=1}^{n} v_j \tanh\left(u_{j0} + \sum_{i=1}^{m} u_{ji} x_i\right) \quad , \tag{2.15}$$

where x_i is the value of the ith input, there are m inputs to the model, and n nodes in the hidden layer. The first-layer weight u_{ji} is the weight from the ith input to the jth hidden

node and u_{j0} is the bias of the jth hidden node. The second-layer weight v_j is the weight connecting the jth hidden-layer node to the output, and v_0 is the output bias.

For comparison, I also train linear networks:

$$\hat{f}_{lin} = u_0 + \sum_{i=1}^{m} u_i x_i \quad . \tag{2.16}$$

The linear networks differ from standard linear models in that they are trained using the methodology described in this section.

The networks are trained using the well-known stochastic backpropagation algorithm. The backpropagation algorithm provides a method for adjusting the weights of a network to minimize some measure of error for a given dataset. This requires that the gradient of the error function with respect to the weights of the network be calculated.

2.4.2 Learning

Since there is not typically a well-defined mapping between forecast errors and the cost of a forecast error in macroeconomics, forecast accuracy is often based on the Mean Square Error (MSE) measure. Mean Square Error is defined as:

$$\mathcal{MSE} = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{f}(\mathbf{x}_i; \mathbf{w}))^2 \quad , \tag{2.17}$$

where the $\mathbf{w}$ represent the parameters of the model, y_i is the target value being predicted, and $\hat{f}(\mathbf{x}_i; \mathbf{w})$ is the model forecast based on the input vector $\mathbf{x}_i$ and weight vector $\mathbf{w}$.

I train the networks to minimize the sum of the MSE and the weighted regularization term

$$\mathcal{E} = \mathcal{MSE} + \lambda \mathcal{REG} \quad , \tag{2.18}$$

subject to the minimum error on the validation set. The regularization term is used to help avoid learning the noise in the training data set. Regularization will be discussed further in Section 2.4.4.

The stochastic backpropagation algorithm proceeds by updating the network weights after each presentation of a data point from the training data set. The weights are initialized to small random values at the beginning of the learning process. The desired change

in the weights is proportional to the gradient of the error function with respect to the model parameters:

$$\Delta w_i = \frac{-\eta \partial \mathcal{E}(\hat{f}(\mathbf{x}; \mathbf{w}))}{\partial w_i} \quad , \tag{2.19}$$

where the w_i represent the individual weights of the network. The weights are adjusted by a small amount in the direction of the negative gradient:

$$w_i(t) = w_i(t-1) + \Delta w_i(t-1) \quad , \tag{2.20}$$

where η is the called the *learning rate* as it controls the amount by which the weights are changed in response to a training exemplar. A single pass through the entire training data set is commonly referred to as an *epoch*. Each network was trained for a maximum of 200 epochs using a learning rate of $\eta = 0.16$. These parameters were set by examination of the training and validation curves during training. The models showed good convergence on the training set, and the validation error reached its minimum before the maximum number of epochs. As will be described in Section 2.6.2, the inputs and targets are normalized prior to training; this helps the gradient descent by normalizing the scales of the inputs with respect to one another.

2.4.3 Moving Window Retraining

To help deal with the problem of nonstationarity I use a sliding, fixed-size training window, and perform model retraining on a year-by-year basis. The training window size is 25 years, or 300 months of data. A larger window size is chosen as the noise problem seems to have a more significant effect than nonstationarity for these datasets. For the datasets used here, the data prior to 1980 is used for parameter selection and training, and predictions are made for the 1980–1989 test period.

To produce forecasts for a 10-year test period, the models are trained using data prior to the beginning of the 10-year period. Forecasts are then produced using these models for the first 12 months of the test period. The training window is shifted forward by 12 months, dropping off the oldest 12 months of data that were previously in the training set, and the models are retrained from a new random initial set of weights (See Figure 2.5).

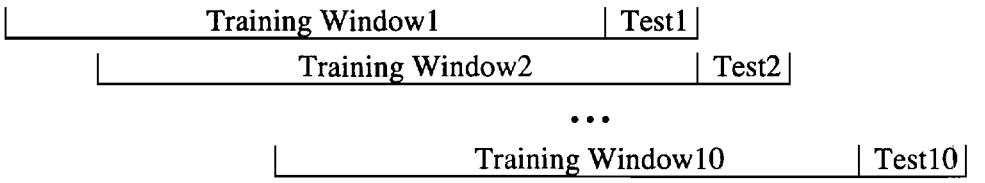

Figure 2.5: A depiction of the year-by-year retraining procedure. Model parameters are reset to random values and retrained each time the training window is shifted forward.

This process proceeds until the final retraining which uses a training window that contains all the data up to the end of the 9th year of the test period. Forecasts are produced for the 10th year of the test period, and are concatenated with the 9 previous sets of predictions.

It is very important to manage the datasets correctly lest future information be incorporated into the training dataset. For example, consider building a model to make h-month forecasts starting Jan. 1980. After constructing backward looking inputs and forward looking target pairs for the training process, the last date actually available for training on is Dec. 1979 $- h$. This is the date of the last full target return that can be calculated without overlapping the out-of-sample test period. In order for the prediction made at the beginning of 1980 to be a true *ex-ante* prediction, this total separation of training set and test set must be enforced. Also, the monthly macroeconomic series considered here must be delayed by one month before being used, as the reported value for the Index of Industrial Production for the month of June is not actually available until the middle of July. So this information can not be used to produce a forecast until the end of July. However, the financial time series are real-time series, and the values are readily available at the end of the current month.

One of the issues investigated for this project was to treat the training window length as a hyperparameter, and to use the training data to select the training window length to be used for the test set. I found this type of selection to be too unstable for the datasets studied. There did not seem to be much correlation between the best window length for adjacent periods of time. For such a method to work well, the optimal training window length would have to change very slowly over time. It would require a fair amount of

historical data to have any confidence that such a procedure would produce stable or reliable results. This type of procedure might work for a higher frequency dataset, but for this study the training window is held constant at 25-years. As we can see from the shape of the curve in Figure 2.1, for these series it is better to have a too long training window than a too short training window.

2.4.4 Regularization

Due to the large amounts of noise in the macroeconomic data, I apply two different regularization techniques to the networks in an attempt to keep the networks from overfitting or learning the noise. The two techniques are often used in neural network training: early stopping and quadratic weight-decay. Both techniques introduce model bias in order to reduce model variance and thereby reduce prediction error. These methods control model complexity as measured by the *effective number of parameters* [63, 64, 65].

Early stopping

The goal of early-stopping is to minimize the prediction error on a test set by stopping training before over-fitting occurs. Figure 2.6 shows a stylized representation of the errors on the training, validation, and testing data sets as a function of training time. Due to the nature of the gradient descent process, the error on the training set will continue to decreases on average. However, once over-fitting begins to occur, the error on the test set begins to increase.

Early stopping involves holding out a random 20% of the training window for validation. The networks are trained on the remaining 80% of the data (the training set), and the hold-out data (the validation set) is used to estimate the prediction error. After each training epoch, the error is measured on the validation data. Training is stopped after the error on the validation data begins to increase, and the set of weights corresponding to the minimum error on the validation data is used to make forecasts of the test data. As will be discussed in Section 2.4.6, committees of 10 models each are used to produce forecasts. One major source of variation in the committee members is that each uses separate, randomly chosen training and validation data subsets of the training window.

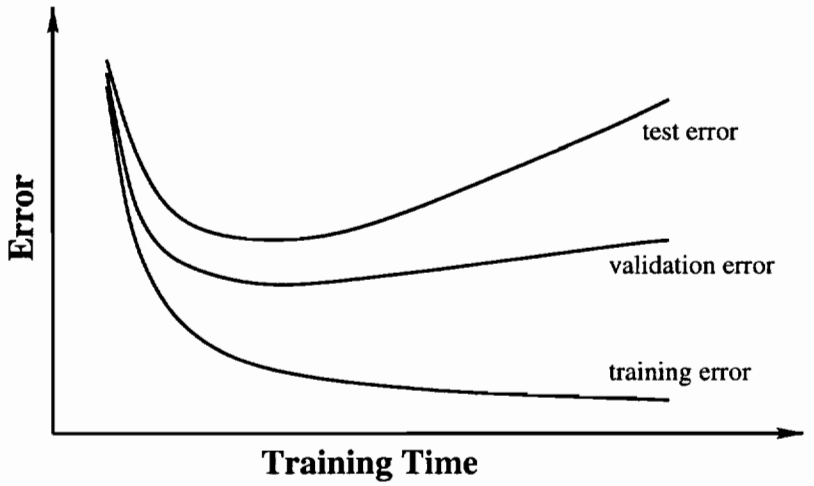

Figure 2.6: This is a stylized representation of the errors on the training, validation, and testing data sets as a function of training time. Due to the nature of the gradient descent process, the error on the training set will continue to decreases on average. However, once overfitting begins to occur, the error on the test set begins to increase. The error on the hold-out validation set is used to approximate the error on the test set, and the training is stopped at the time corresponding to the minimum error on the validation set.

Figure 2.7 depicts how the training and validation datasets vary between different committee members. Early stopping occurs for committee member i when the error measured on the data points marked with black arrows (which are not used to train the parameters) is at a minimum. Early stopping for committee member j is based on the data points marked with white arrows.

Quadratic weight decay

Quadratic weight decay (similar to *ridge regression* in the context of linear regression in the statistical literature [44, 43]) is a commonly used method to help avoid fitting the noise in a data set. Weight decay involves adding a term to the cost function that penalizes large weight values in the network

$$\mathcal{E} = \mathcal{MSE} + \lambda \sum_i w_i^2 \tag{2.21}$$

where the w_i are all of the weights in the model including the biases, and N is the number of data points in the training set. The weight decay parameter, λ, determines the amount

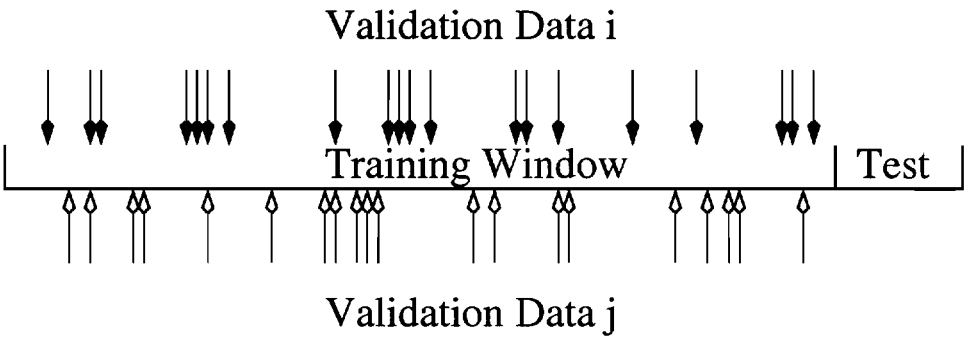

Figure 2.7: A graphical depiction of how the validation datasets vary between committee members i and j. Early stopping occurs for committee member i when the error measured on the data points marked with black arrows (which are not used to train the parameters) is at a minimum. Early stopping for committee member j is based on the data points marked with white arrows.

of contribution the penalty term makes to the total error, and has the effect of controlling the *effective number of parameters.* The best choice of λ is problem dependent so a separate λ is chosen for each series and forecast horizon, though the individual committee members all use the same value of λ as described below. It is worth noting that for networks with sigmoidal nonlinearities, weight decay regularization in its usual form biases the model to a linear structure.

For a given series and a given forecast horizon, I choose the weight decay parameter by performing a search over λ based on the training set data. The search consists of evaluating each choice of λ by testing the methodology using the sliding window retraining on the 1970-1979 training data as described in Section 2.4.3 and depicted in Figure 2.5. The value of λ which gives the best aggregate performance on this 10 year *ex-ante* simulation test period is then used to train the models for predicting the desired test period of the 1980's. The reason this full 10-year period is used to evaluate the choice of λ instead of a much smaller validation set as used for early stopping is so that models are not fooled into choosing too small or too large values based on a possibly unrepresentative sample of data.

The importance of choosing an appropriate value of λ is shown in Figure 2.8. This

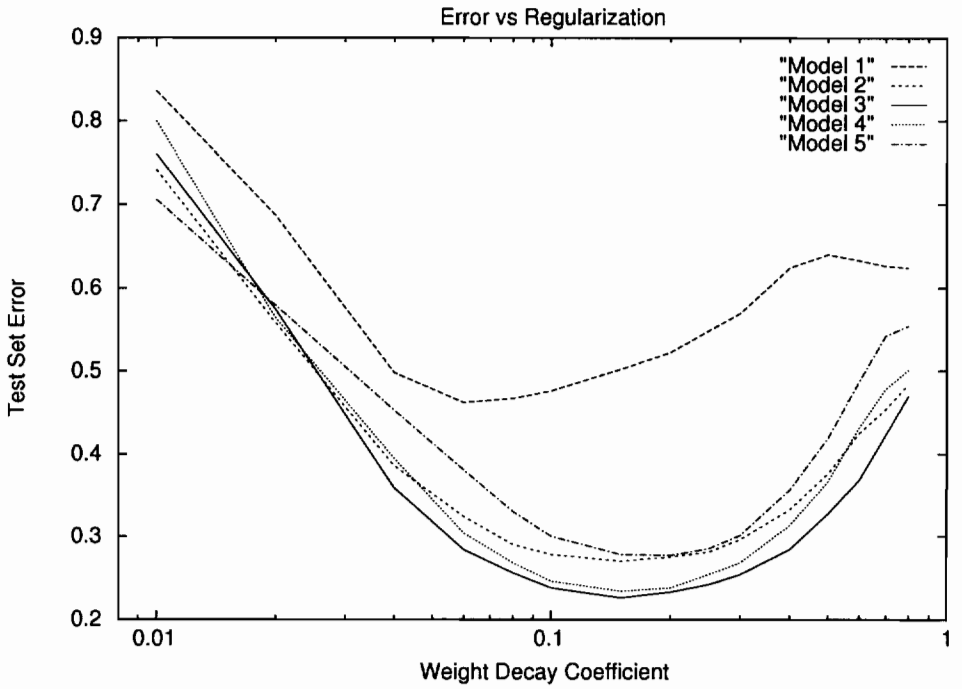

Figure 2.8: Mean Square Error values of a test set as a function of the regularization parameter λ. The separate curves represent various models trained to predict the test set. When the weight decay coefficient is small, errors are high due to the large model variance. When the weight decay coefficient is too large, errors increase because the models become too biased, and can not adequately represent the underlying structure in the data. Most of the models shown here produce the lowest errors when the regularization parameter is approximately 0.1 to 0.3 in magnitude.

figure shows the characteristic effect on test-set error of using different amounts of regularization during training. If the value of λ is too small then the network overtrains on the training data, fitting noise features which have no relevance to the test data. As the value of λ increases, the model variance decreases and generalization increases. If λ becomes too large, error begins to increase again because now the network is too biased, and is no longer able to fit the true structure of the data. Most of the models shown here produce the lowest errors when the regularization parameter is approximately 0.1 to 0.3 in magnitude. This relatively large value for λ indicates that these series really have a very large amount of noise.

Bayesian Interpretation of Weight Decay

If we exponentiate the regularized error, Equation 2.21, we have

$$\exp\left(-\mathcal{MSE}\right) \cdot \exp\left(-\lambda \sum_i w_i^2\right) \tag{2.22}$$

which is the penalized likelihood function. The first part of this represents the likelihood of the data given the model, $p(\mathcal{D}|w)$. The second part, derived from the regularization term, can be viewed as a prior on the model parameters, $p(w)$. According to Bayes theorem, the posterior model probability $p(w|\mathcal{D})$ is proportional to 2.22, so minimizing 2.21 corresponds to maximizing the posterior model probability.

Discussion

As will be shown in Section 2.7.4, I have found that both early stopping and quadratic weight decay are useful for improving the performance of the models. Though often early stopping offers only an incremental enhancement when used in the presence of weight decay. It is worth noting again the different ways in which the two methods are used to determine the amount of regularization each applies to the model. The weight decay parameter is chosen with respect to ex-ante performance from simulations on historical data, while the early stopping uses a validation dataset sampled from the current training window to estimate out-of-sample performance. Also, all committee members use the same weight decay coefficient while the early stopping point is determined on an individual basis using individual randomly selected validation datasets.

Figure 2.9 is a depiction of the effects early stopping and weight decay have on the learned value of the weights. Note that W_{opt} is the optimal weight vector for the training data set, not the true underlying function. Indeed, through the use of validation data sets, the presumption is that regularization will result in weight vectors closer to the true optimal value than could be learned from the training data alone. Early stopping halts training somewhere along the path from the initial weight vector to the training set optimal vector. Weight decay as implemented here biases the weight vector toward zero. If the initial weights are close to zero then the effect of early stopping may be hard to distinguish from the effect of weight decay.

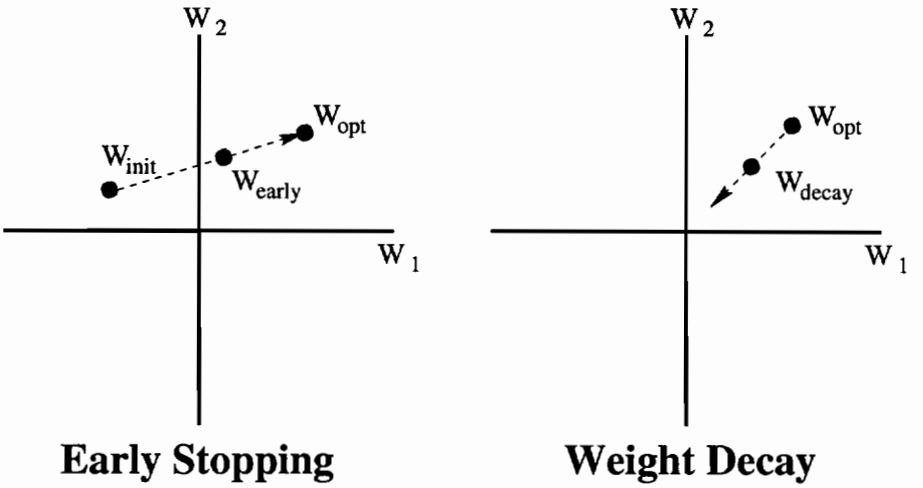

Figure 2.9: A graphical representation of the effect of regularization on the weights of a model. W_{opt} is the optimal weight vector for the training data set. Early stopping halts training somewhere along the path from the initial weight vector to the training set optimal vector. Weight decay as implemented here biases the weight vector toward zero.

For the network models used here, the use of weight decay to shrink toward zero corresponds to a prior of a random walk plus drift model. This is due to the preprocessing performed on the inputs and targets (See Section 2.6.2). To improve the learning dynamics of the network models, the target returns are demeaned and scaled to have unit variance based on the training set data. The out-of-sample forecasts are then transformed back using the previously calculated training set mean and variance. As the model weights shrink to zero, the model forecasts shrink to zero as well, but the the postprocessing adds the training set mean back in, resulting in the drift component.

2.4.5 Complexity Reduction & Model Selection

Model selection is a critical component of the forecasting problem. Model selection includes choosing the input variables as well as defining the model topology. If there are too many adjustable parameters in a model relative to the amount of available data, then overfitting can occur during the model optimization procedure, resulting in poor generalization on the test set. This problem is particularly acute when data sets are small and very noisy as is the case with the macroeconomic data sets considered in this chapter.

Some authors, such as Swanson and White [107], use methods that grow the model structure starting from a linear model and add nonlinear components according to saliency measures that measure a tradeoff between model complexity and goodness of fit. This thesis uses a methodology that starts with a large number of inputs and a relatively complex model structure, and then proceeds to reduce the complexity of the model using data-based saliency measures. This section briefly discusses some input selection and model pruning methods including Principal Component Pruning which is used for the results presented later in this chapter.

Input Selection

There have been a number of model-independent and model-dependent variable selection procedures proposed [48, 1]. The Delta Test, a model-independent procedure, is a nonparametric statistical algorithm that selects meaningful predictor variables by direct examination of the data set [87]. The reader is referred to the preceding reference for a description of the algorithm. A preliminary study used the Delta Test to find a very successful set of inputs for predicting 12-month ahead values of the Index of Industrial Production. However, further experimentation with the technique revealed that it is very sensitive to the effects of noise in the data, and was not useful in general for the macroeconomic data sets.

Sensitivity-Based Pruning (SBP) techniques are model-dependent algorithms that prune unnecessary or harmful input variables from a trained network [77, 73, 65, 114]. Sensitivity results presented in Section 2.7.5 indicate that explicit input selection may be too difficult given the apparent amount of nonstationarity present in the data.

Pruning & Rank Reduction

There are a number of techniques available for pruning the structure of a network including the Optimal Brain Damage (OBD) [97] and Optimal Brain Surgeon (OBS) [100] algorithms which prune weights from the network. These algorithms train the network to a local minimum in error, and then calculate the second derivative of the error function. OBD uses a diagonal approximation of the Hessian while OBS calculates the entire matrix.

Then, weights are removed from the network model based on a saliency calculation using the Hessian or its approximation.

In the work presented here I use supervised Principal Component Pruning (PCP) [50] to reduce the complexity of the network structures. PCP is a method for pruning the eigen-nodes in each layer of a neural network. PCP is intermediate between OBD and OBS in that it uses a block-diagonal approximation of the Hessian to select candidate weights for pruning. Unlike OBD and OBS, PCP does not require training to a local minimum, allowing it to be used in conjunction with early stopping and weight decay. Principal Component Pruning uses more information than OBD and the approximation to the Hessian is faster to compute than the OBS method. The PCP algorithm provides a tradeoff between accuracy and computational complexity.

The PCP algorithm starts with a network structure where each layer calculates some function $\Gamma(\cdot)$ of its weighted input

$$y^i = \Gamma(x^i) \equiv \Gamma(W^i u^i) \quad , \tag{2.23}$$

where u^i is the input for layer i and W^i is the weight for layer i. The algorithm then calculates the correlation matrix for each layer, i, in the network

$$\Sigma^i = \frac{1}{N} \sum_{k=1}^{N} u^i(k) u^{iT}(k) \tag{2.24}$$

where there are N data points and $u(k)$ is the vector of layer inputs given network input k. Σ is then diagonalized

$$\Lambda^i \equiv C^{iT} \Sigma^i C^i \tag{2.25}$$

and the eigenvectors are used to transform the inputs and weights of layer i

$$\tilde{u}^i = C^{iT} u^i \tag{2.26}$$

$$\tilde{W}^i = W^i C^i \quad . \tag{2.27}$$

The weighted input to the layer is then

$$x^i = \tilde{W}^i \tilde{u}^i \quad . \tag{2.28}$$

The PCP algorithm uses a diagonal pruning matrix P^i to prune the eigen-nodes for layer i

$$x^i = \tilde{W}^i P^i \tilde{u}^i \quad . \tag{2.29}$$

Supervised PCP

The supervised PCP algorithm as implemented here proceeds by ranking the eigen-nodes of Σ^i in order of increasing effect on the training error. Then, starting with a diagonal pruning matrix $P^i = I$, eigen-node j is deselected by setting P^i_{jj} to 0, and the prediction error is estimated using a validation data set. If the prediction error is reduced then the eigen-node remains deselected and the process continues to the next eigen-node. If the prediction error increases, then the eigen-node is reselected and the pruning process is halted. The layer weights are then set to

$$W^i_{new} = \tilde{W}^i C^{iT} P^i \quad , \tag{2.30}$$

and the remaining, unprocessed layers of the network are retrained on the training set. This entire process is then repeated on the next layer of the network, and so on until each layer's eigen-nodes have been pruned.

2.4.6 Committees & Variance Reduction

Due to the extremely noisy nature of economic time series and the influence of initial conditions on the model optimization process, the control of forecast variance is a critical issue. One approach for reducing forecast variance is to average the forecasts of a committee of models.

In the work presented in this chapter, I use the simple average of ten individual forecasts as the committee forecast

$$F_{committee} = \frac{1}{10} \sum_{i=1}^{10} f_i \tag{2.31}$$

where the individual f_i are the predictions of the individual networks. Each network is trained using a separate, randomly selected validation data set, and starts with a different random set of initial weights. The efficacy of the data-driven portions of the methodology

is directly related to the characteristics of the validation data compared to the unseen test data. Averaging over multiple realizations of the validation sets is crucial to not being misled by a specific feature or event that may occur in any specific validation set. Also, the stochastic backpropagation learning algorithm is sensitive to the model's starting point in weight space. Similarly, the averaging over initial starting points ameliorates the problems with any individual optimization becoming stuck in a local minimum of weight space. Breiman's bagging [15] algorithm relies on the use of bootstrapped data sets to introduce variation in the created models. He notes that the more variation that can be induced, the more advantage there is to the combining process.

Figure 2.10 shows the variance reduction effect from using committees of models to produce forecasts. The comparison is between 1000 individual networks, and the same networks randomly divided into committees of 10. The committee approach sharply reduces the variances in the resulting errors of the forecasts. With a convex error function, committee errors are necessarily less than the average of the individual errors; in this case they are typically very close, and committees serve mainly as a variance reduction technique.

Researchers in economics have studied and used combined estimators for many years, and generally find that they outperform their component estimators and that unweighted averages tend to outperform weighted averages, for a variety of weighting methods [37, 126, 21]. These weighting methods usually try to take advantage of the correlations between the committee members.

Another reason to combine forecasts would be to aggregate alternative sources of information. These sources of information could be different inputs sets or even different model structures. These types of combinations are not considered here.

2.5 Benchmark Models

I compare the results of the network models with several benchmark models. There are two trivial predictor models: a random walk with drift model and a random trend model. There are two basic linear models: the multivariate ordinary least squares regression

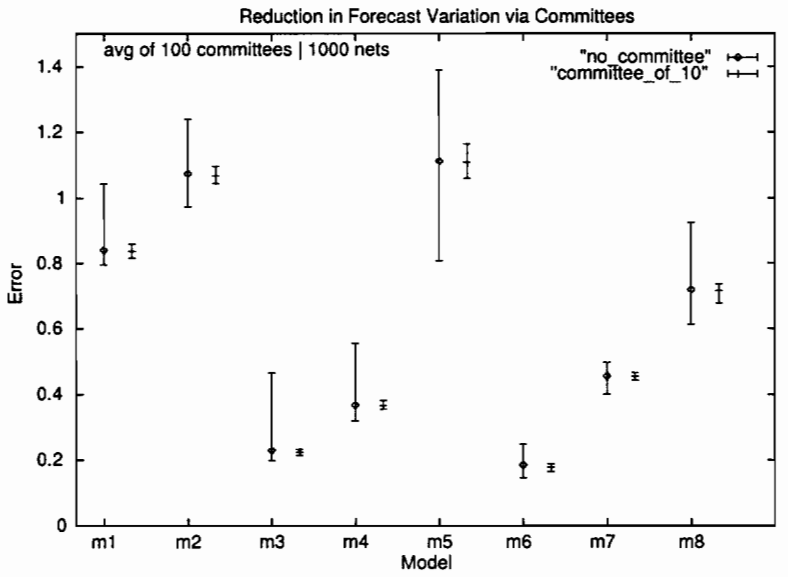

Figure 2.10: A demonstration of the use of committees to reduce the variance of forecasts. In each pair of bars, the left bar shows the spread in average errors for the total 1000 individual forecasting models. The right bar shows the spread in average errors when the forecasts are first combined using committees of ten models each.

model, and the univariate linear autoregressive model. The final comparison model is a Bayesian Vector Autoregression (BVAR) model. The BVAR model incorporates Bayesian regularization, and has performed well compared to standard models in previous macroeconomic forecasting studies [51, 56].

The following sections will assume that the models are forecasting returns. The h-day ahead return at time t will be denoted $y_t(h)$.

2.5.1 Trivial Predictors

Comparing model performance to relevant trivial predictors is important for establishing a baseline, especially when there is no way to know what the best practically achievable performance is for a dataset. Otherwise, comparisons between more complicated models may be irrelevant if they cannot outperform very simple models. There are two trivial predictor models for which I include results.

The *median-return* trivial predictor corresponds to a random walk with drift model.

Its h-day ahead point forecast, $\hat{y}_t(h)$, is the median h-month return for the series measured over the preceding 25-year training period. While the *no-change-in-level* predictor, $\hat{y}_t(h) = 0$, would correspond to a random walk model, the random walk with drift model is a more appropriate baseline since the training set mean return is removed during training in a preprocessing step, and is later added back to the test set forecasts (see Section 2.6.2).

The *no-change-in-return* trivial predictor corresponds to a random walk plus random trend model. For a forecast horizon h, the forecast at time t is

$$\hat{y}_t(h) = y_{t-h}(h) \quad . \tag{2.32}$$

This predicts that the next value, over horizon h, is equal to the current level plus the current trend as given by the just-completed return.

2.5.2 Linear Regression

The linear regression model is a basic linear regression based on all 76 inputs,

$$\hat{f}_t = u_0 + \sum_{i=1}^{m} u_i x_{it} \quad , \tag{2.33}$$

and produces direct predictions of the target variable. The method of Ordinary Least Squares is used to estimate the parameters of these models. New models are fit each year using the sliding window retraining methodology described in Section 2.4.3.

2.5.3 Linear Autoregression

The linear AR model is univariate and has the structure as described in Section 2.2.1

$$\hat{y}_t = \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_p y_{t-p} \quad . \tag{2.34}$$

The AR order, p, for each series is determined by the statistically significant lags of the partial-autocorrelation of the series as listed in Table 2.2. Since the models are univariate, they only include lagged values of themselves, up to a maximum of 24 for each series. h-month forecasts are produced by iterating predictions of 1-month returns. That is, the 1-month iterated forecasts

$$\hat{y}_t \quad = \quad \alpha_1 y_{t-1} + \alpha_2 y_{t-2} + \ldots + \alpha_p y_{t-p} \tag{2.35}$$

Table 2.2: The number of lags included in each univariate AR model. The number was chosen based on the significant lags of the partial autocorrelation function of each series up to a maximum of 24.

Series	# of AR Lags
DLEAD	24
DRM	24
FM2DQ	6
FSPCOM	16
FYAAAC	24
HSBP	14
IP	24
LHUR	24
PUNEW	24
YCS	24

$$\hat{y}_{t+1,t} = \alpha_1 \hat{y}_t + \alpha_2 y_{t-1} + \ldots + \alpha_p y_{t-p+1}$$

$$\hat{y}_{t+2,t} = \alpha_1 \hat{y}_{t+1,t} + \alpha_2 \hat{y}_t + \ldots + \alpha_p y_{t-p+2}$$

are used to generate the 3-month ahead forecast

$$\hat{y}_t(3) = \hat{y}_t + \hat{y}_{t+1,t} + \hat{y}_{t+2,t} \quad . \tag{2.36}$$

2.5.4 Bayesian Vector Autoregression

The Bayesian Vector Autoregression (BVAR) is a method for imposing prior information on vector autoregressions in order to mitigate the effect of overfitting the observed data. The BVAR method was first proposed by Litterman [51]. BVAR was proposed as an alternative to the standard economic approach of using an economic theory to suggest a small number of places to look for useful information. Litterman's BVAR approach is based on the belief that useful information about the future is likely to be spread across a wide spectrum of economic data and time lags. The priors in the BVAR approach allow all of this data to be incorporated into a model when the relatively limited amount of available data would be inadequate to support an unconstrained fit.

The BVAR model is based on the standard VAR model described in Section 2.2.1, and extends it by using priors to bias the parameter values. The induced parameter bias

improves the model fitting process by helping make up for the lack of the large amounts of data needed to estimate the VAR parameters.

Recall the component equation for the VAR model (Equation 2.9):

$$y_{it} = \sum_{k=1}^{p} \left[\alpha_{iik} y_{i(t-k)} + \sum_{j \neq i} \alpha_{ijk} y_{j(t-k)} \right] + \epsilon_{it} \quad . \tag{2.37}$$

A commonly used prior definition [27] sets the prior distributions of the VAR parameters to have the form:

$$\alpha_{ijk} \sim N(0, \sigma_{ijk}^2) \tag{2.38}$$

where α_{ijk} is the coefficient at lag k, weighting the contribution of input j in the forecast of variable i. The magnitude of the variances σ_{ijk}^2 decrease with increasing k and are described below. This prior emphasizes a random walk with drift prior model. Litterman [51] motivates the use of a 0 prior for many of the variables by noting a correspondence to ridge regression.

The prior variances specify uncertainty about the prior means. Due to the large numbers of parameters in a typical VAR model, Doan et al. [27] suggested a formula to generate the standard deviations from a small number of hyperparameters: θ, ϕ, and a weighting matrix $\mathbf{W}$ with elements $w(i,j)$, allowing the specification of prior variances for a large number of coefficients using only a few hyperparameters. The standard deviation of the prior imposed on variable j in equation i at lag k is:

$$\sigma_{ijk} = \theta w(i,j) k^{-\phi} \left(\frac{\hat{\sigma}_{uj}}{\hat{\sigma}_{ui}} \right) \tag{2.39}$$

where $\hat{\sigma}_{ui}$ is the estimated standard error from a univariate autoregression involving variable i, so that $\frac{\hat{\sigma}_{uj}}{\hat{\sigma}_{ui}}$ is a scaling factor that adjusts for varying magnitudes of the variables across equations i and j. Doan et al. [27] label θ the "overall tightness", reflecting the standard deviation of the prior on the first lag of the dependent variable. The term $k^{-\phi}$ is a lag decay function with $0 \leq \phi \leq 1$ which shrinks the standard deviation with increasing lag length. This represents the belief that more distant lags contain less useful information. The function $w(i,j)$ specifies the tightness of the prior for the parameters

for variable j in Equation 2.8 relative to the tightness of the priors for the parameters for the dependent lags of variable i in Equation 2.8.

The hyperparameters used in the standard prior described above have values $\theta = 0.1$ and $\phi = 1$. The weighting matrix used is:

$$\mathbf{W} = \begin{bmatrix} 1 & 0.5 & \cdots & 0.5 \\ 0.5 & 1 & & 0.5 \\ \vdots & & \ddots & \vdots \\ 0.5 & 0.5 & \cdots & 1 \end{bmatrix} . \tag{2.40}$$

Implementation Details

The standard prior described is used in the simulation results reported in Section 2.7. The sliding window methodology described in Section 2.4.3 was used when creating the BVAR models. The inputs for each model are lags of the the one-month log returns of the 11 series listed in Table 2.1. The maximum input lag is $k = 24$, producing a total of 264 inputs per model. The h-month forecasts were made by iterating the one-month forecasts forward in time as described for the AR models in the previous section. The BVAR models presented in this chapter were created using a Matlab toolbox freely distributed by James P. LeSage[1].

2.6 Empirical Methodology & Setup

In this section I describe the details of the models and training procedures used in forecasting the macroeconomic and financial time series. In the following section I will present the results of the studies done using the methodology described previously in this chapter.

2.6.1 Model Structure

The nonlinear models used for point forecasts are feedforward neural networks with three *tanh* units in the hidden layer (see Section 2.4.1). Because of the relatively small amount of available training data, I limited the size of the point forecast networks to only three units

[1] http://www.spatial-econometrics.com/

to help avoid overfitting problems. A total of ten committees of ten models each are used to predict the 10-year period, 1980–1989, with one committee per year. First, a committee is trained on a window of the training data prior to the first month of $1980 - h$, where h is the forecast horizon in months, and the window size is 300 months. The committee is then used to make monthly predictions for the year of 1980, the training window is shifted forward by 12 months, the training procedure is repeated, and predictions are made for the next year. Each time the training window is shifted, the weights of each committee member are reinitialized to random starting values. This is done also to help avoid overfitting. The weight decay regularization parameter is chosen once for the ten-year period based on the training data prior to $1980 - h$ as described in Section 2.4.4.

The nonlinear models used for directional forecasts are feedforward neural networks with 3 or 10 *tanh* units in the hidden layer (see Section 2.4.1). I compare with networks containing 10 hidden units due to the increased number of output units (4 or 5 outputs per network depending on the output representation, see Section 2.8). Following the methodology of the point forecast studies, a total of ten committees of ten models each are used to predict the 10-year period, 1980–1989, with one committee per year. The directional forecast models are used to predict quintiles of directional movement. That is, the training returns are divided up into sets representing "large negative", "small negative", "flat", "small positive", and "large positive" log price changes. Each class is divided to contain 20% of the data in the initial training set. Note that the classes are relative to the median return instead of the zero return value. So the "flat" class is centered around the median return over the training period as opposed to representing no return.

2.6.2 Data

The dataset used consists of monthly time series taken from the Citibase database. The macroeconomic series are fully revised (see Section 2.3.3 for a discussion of revised vs real-time macroeconomic datasets), and cover the time period, 1950-1995. The data prior to 1980 is used for hyperparameter and model selection purposes as described in Section 2.4, and the results reported here are for the 1980-1990 test period. The macroeconomic data

are lagged by one month before being used in the models to account for reporting delays. The financial time series are real-time series, and the values are readily available.

In addition to the results presented in this section, tables of numerical results for all the series are listed in Appendix A.

Inputs

The macroeconomic series are preprocessed before being used by the networks. The representation used for most input series is log returns, ie. the first difference on varying time scales of the logged series.

$$r_t(-h) = \Delta_h log\,(x_t) = log\left(\frac{x_t}{x_{t-h}}\right) \quad , \tag{2.41}$$

where h takes on the values $\{1, 3, 6, 9, 12, 18, 24\}$. Taking the logs of series is a commonly used transformation in econometrics, and is an example of a Box-Cox transformation [12]. These types of transformations are used to alleviate problems with heteroskedacticity in the returns series.

The telescoping returns representation is used rather than a standard AR representation in order to reduce the amount of noise in the input signals. This can help with the overfitting problems as there are fewer significant noise features in the inputs due to the averaging process implicit in the multi-month returns calculation. This may also enable faster learning as the future trend of a series is more likely to be related to the previous trend as opposed to any single month's return. On the other hand, the resulting correlation between the input signals contributes to a condition called "multicollinearity" where the effective dimension of the inputs can be less than the number of input signals, and can lead to increased variance in the optimized parameters. However, this type of problem can dealt with through the use of regularization techniques as described in this chapter.

The short-hand notation used in this thesis when referring to a specific input series consists of the designation (Table 2.1) of the given series, for example "DLEAD", followed by an "L" if logs were taken of the series, and ended with an indication of the time scale on which the first difference was taken, for example "D6". So the notation for the 6 month

log return of the Index of Leading Indicators would be

$$\text{DLEAD.L.D6} \equiv \ln(\text{DLEAD}_t) - \ln(\text{DLEAD}_{t-6}) \quad . \tag{2.42}$$

Using the returns of the series is necessary to avoid complications due to unit roots[2] in many of the series. The Housing Starts and the Unemployment Rate series did not appear to exhibit heteroskedacticity, and were not logged.

In order to facilitate the network learning, an additional preprocessing step removes the mean of each input series, and scales the signal to have a unit variance. The mean and variance are measured over the 25-year training set. This same mean and variance is used to preprocess the inputs when the forecasts are being made out of sample.

Targets

The target series are similarly preprocessed, except that forward returns are used

$$r_t(h) = B^{-h} r_t(-h) \tag{2.43}$$

where B is the backwards shift operator:

$$Bx_{t+1} = x_t \quad . \tag{2.44}$$

So, if the target is the 12 month forward return of the Index of Industrial Production, then this is denoted

$$\text{IP.L.FD12} \equiv \ln(\text{IP}_{t+12}) - \ln(\text{IP}_t) \quad . \tag{2.45}$$

Predicting the 12 month forward return of a series at time t, contains the same information as predicting the value of the series 12 months into the future, but the properties of the returns series are more amenable to prediction methods.

Again, an additional preprocessing step removes the mean of the target series, and scales the series to have a unit variance. The mean and variance are measured over the 25-year training set. This same mean and variance is used to transform the out-of-sample forecasts.

[2]See Section 2.2.1.

In the results sections, I present results for making 3 and 12 month forecasts of value and of direction. As inputs to the models I include the 1, 3, 6, 9, 12, 18, and 24 month returns of the series listed in Table 2.1. Also, I include the levels of the interest rate series (including the Default Risk Measure series), the Unemployment Rate, and the Housing Starts Index. This makes a total of a 76 inputs to the forecasting models. The one exception is the M2 money supply series (FM2DQ). This series was added to the study at a later date, so while the other series do not contain it among their inputs, it of course includes itself among its inputs, and thus has a total of 83 inputs.

The point forecast network models have 0 or 3 internal units and are structured as described in Section 2.4.1. Thus they contain 77 and 235 adjustable parameters respectively (84 and 256 for FM2DQ). The directional forecast network models have 0, 3, or 10 internal units, and 4 or 5 output nodes depending on the target representation. This corresponds to 308, 247, and 814 parameters (336, 268, and 884 for FM2DQ) for the models with 4 outputs, and 385, 251, and 825 parameters (420, 272, and 895 for FM2DQ) for models with 5 outputs.

2.6.3 Statistical Significance

Due in part to the large amount of regularization used in producing the forecasting models, the forecast errors, $e_{it} = (y_{it} - \hat{f}_i(\mathbf{x}_t; \mathbf{w}))$, have properties that invalidate the use of standard hypothesis testing. These properties include a non-gaussian distribution, non-zero mean forecast error, serial correlation, and contemporaneous correlation among different model structures.

Recently in macroeconomic forecasting, a widely-used test for forecast accuracy has been the Diebold-Mariano test [25]. Diebold and Mariano propose a hypothesis test on the difference between two models errors that is generally valid under a wide variety of conditions including non-normal distributions and serial correlation. The test is also valid for loss functions besides the mean square error measure.

First, compute the loss-differential series for two models i and j, $d_t = [g(e_{it}) - g(e_{jt})]$,

where $g(\cdot)$ is a generalized loss function. If the loss-differential series is covariance stationary and short memory then

$$\sqrt{T}(\bar{d} - \mu) \xrightarrow{d} N(0, 2\pi f_d(0)) \quad , \tag{2.46}$$

where

$$\bar{d} = \frac{1}{T}\sum_{t=1}^{T}[g(e_{it}) - g(e_{jt})] \tag{2.47}$$

is the sample mean loss differential,

$$f_d(0) = \frac{1}{2\pi}\sum_{\tau=-\inf}^{\inf}\gamma_d(\tau) \tag{2.48}$$

is the spectral density of the loss differential at frequency 0, $\gamma_d(\tau) = E[(d_t - \mu)(d_{t-\tau} - \mu)]$ is the autocovariance of the loss differential at displacement τ, and μ is the population mean loss differential.

In large samples then, the statistic for testing the null hypothesis is

$$S_1 = \frac{\bar{d}}{\sqrt{\frac{2\pi\hat{f}_d(0)}{T}}} \quad , \tag{2.49}$$

where $\hat{f}_d(0)$ is a consistent estimate of $f_d(0)$. A consistent estimate of $2\pi f_d(0)$ may be obtained by taking a weighted sum of the available sample autocovariances,

$$2\pi\hat{f}_d(0) = \sum_{\tau=-(T-1)}^{(T-1)} 1\left(\frac{\tau}{S(T)}\right)\hat{\gamma}_d(\tau) \quad , \tag{2.50}$$

where

$$\hat{\gamma}_d(\tau) = \frac{1}{T}\sum_{t=|\tau|+1}^{T}(d_t - \bar{d})(d_{t-|\tau|} - \bar{d}) \quad , \tag{2.51}$$

$1\frac{\tau}{S(T)}$ is the lag window, and $S(T)$ is the truncation lag. Diebold and Mariano suggest that using a rectangular window with length $k-1$ for a k-step-ahead forecast is adequate in many cases. In the presentation of results for this chapter I use squared error as the generalized loss function $g(\cdot)$, and use the $k-1$ rectangular window.

A number of more general non parametric tests such as the Wilcoxon Rank Sum test [60] were investigated for measuring the statistical significance of the results. Generally, these tests were not able to distinguish between the different models performance

very well. Even the Deibold-Mariano test sometimes has difficulty distinguishing between models which appear to exhibit marked differences in accuracy. There is the potential for more research to be done in this area, possibly with the use of the stationary bootstrap [30] to set confidence intervals.

2.7 Empirical Studies: Point Forecasts

In this and the next section I present results for point and directional forecasts of 3 and 12 month changes in the macroeconomic and financial time series described previously. I show results for a number of linear and nonlinear models and compare their performance in terms of mean-square error and directional accuracy. I present the point forecast results in this section, and directional forecast results in the next section.

The regression network models are trained with a single linear output node, and with either no hidden nodes or with 3 nonlinear hidden nodes. The target is the 3 or 12 month future returns for the respective series, either logged or unlogged as described in Section 2.6.2. I find that the network models perform favorably compared to the other benchmarks considered. Figures 2.11 through 2.14 compare the errors produced by these models, and will be discussed in the next section.

The results presented here are stated in terms of *Normalized Mean Square Error* (NMSE).

$$\text{NMSE} = \frac{\frac{1}{N}\sum_{i=1}^{N}(t_i - F(x_i))^2}{\frac{1}{N-1}\sum_{i=1}^{N}(t_i - \bar{t})^2} \tag{2.52}$$

That is, NMSE is the Mean Square Error performance of the model on the test set normalized by the variance of the test set. An NMSE of 1 would be achieved by predicting the average value of the test set.

2.7.1 Point Forecast Results Summary

Figures 2.11 and 2.12 show barcharts of the normalized mean-square error results for point forecasts of the 3-month ahead changes in the target series. Figures 2.11 and 2.14 show barcharts of the results for 12-month ahead point forecasts. Numerical tables of these and

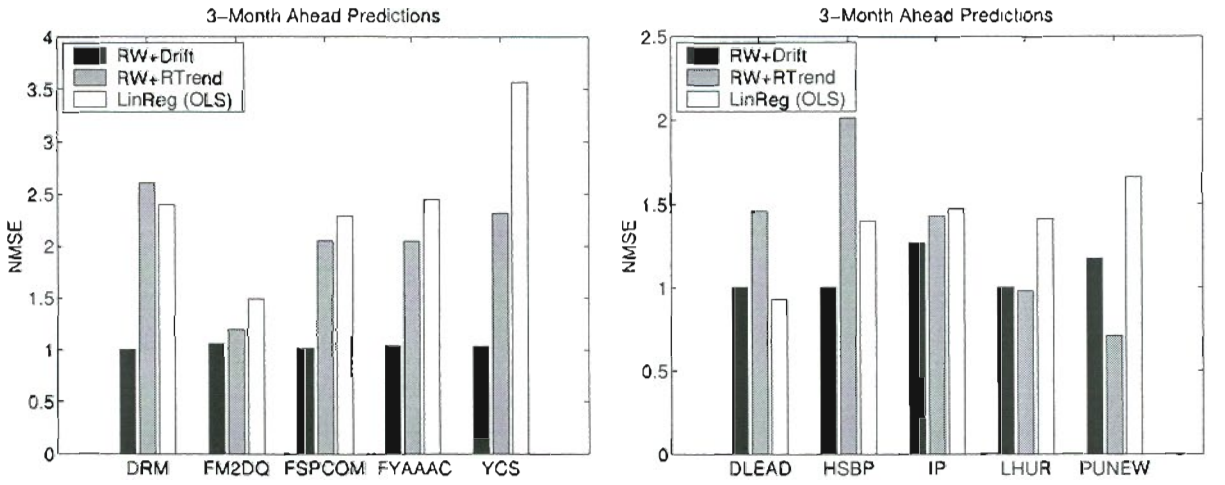

Figure 2.11: Normalized mean square errors for making 3-month point forecasts of the test set. The "RW+Drift" model uses the median return measured on the training set as a forecast. The "RW+RTrend" model uses the most recent 3-month return as a forecast. The "LinReg (OLS)" forecast is an ordinary least squares linear regression based on the 76 inputs as described in the text. An NMSE of 1 would be achieved by predicting the average value of the test set.

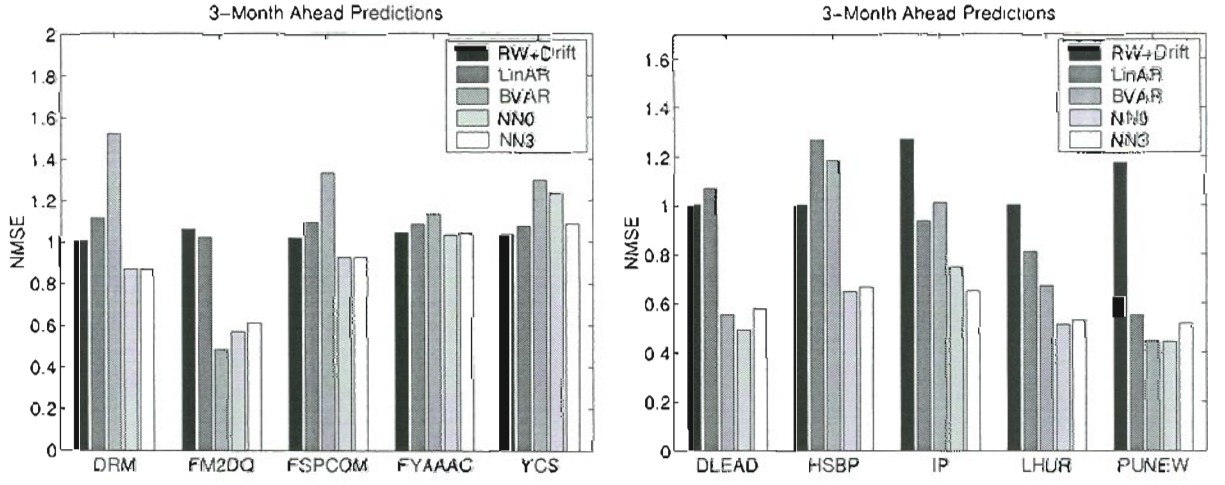

Figure 2.12: Normalized mean square errors for making 3-month point forecasts of the test set. The "RW+Drift" model uses the median return measured on the training set as a forecast. "LinAR" is a univariate AR time series model that uses iterated predictions. "BVAR" is a Bayesian Vector Autoregression model as described in Section 2.5.4. "NN0" is the linear network model, and "NN3" is the 3-unit nonlinear network model. An NMSE of 1 would be achieved by predicting the average value of the test set.

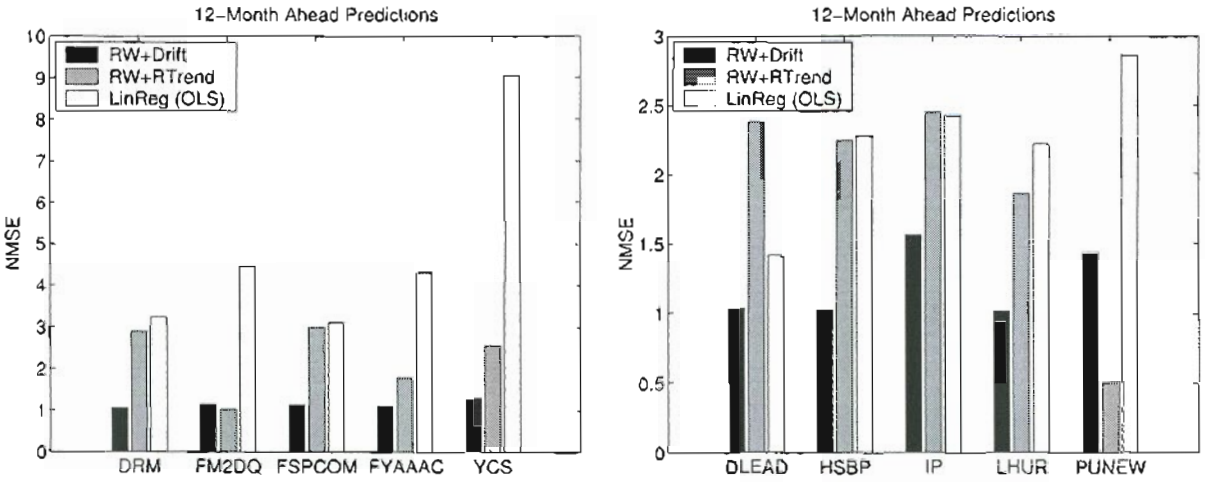

Figure 2.13: Normalized mean square errors for making 12-month point forecasts of the test set. The "RW+Drift" model uses the median return measured on the training set as a forecast. The "RW+RTrend" model uses the most recent 12-month return as a forecast. The "LinReg (OLS)" forecast is a linear regression based on the 76 inputs as described in the text. An NMSE of 1 would be achieved by predicting the average value of the test set.

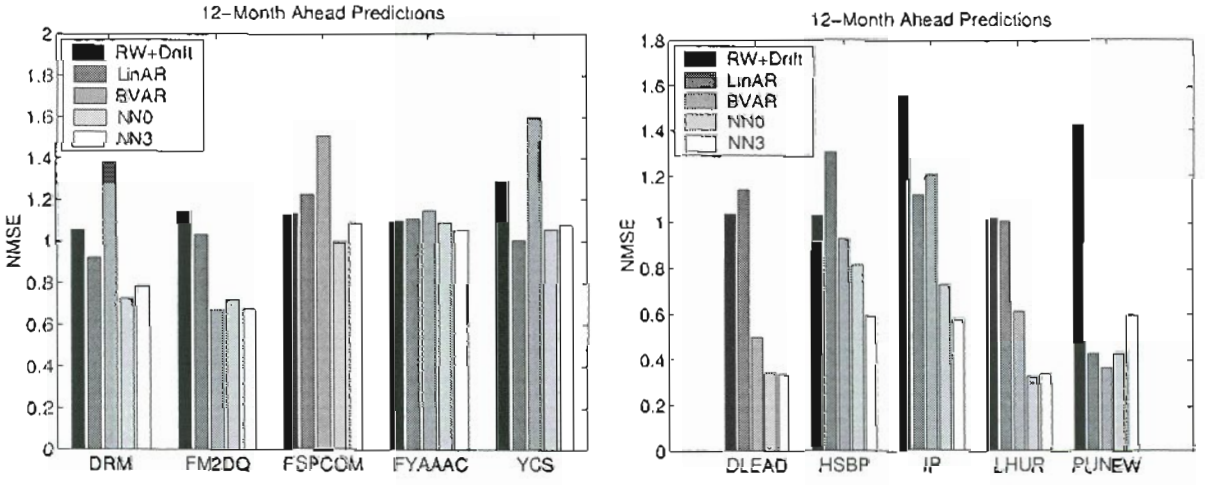

Figure 2.14: Normalized mean square errors for making 3-month point forecasts of the test set. The "RW+Drift" model uses the median return measured on the training set as a forecast. "LinAR" is a univariate AR time series model that uses iterated predictions. "BVAR" is a Bayesian Vector Autoregression model as described in Section 2.5.4. "NN0" is the linear network model, and "NN3" is the 3-unit nonlinear network model. An NMSE of 1 would be achieved by predicting the average value of the test set.

Table 2.3: Summary of the number of times each point forecast model was not significantly worse than the best model at an estimated 5% significance level as determined by the Diebold-Mariano test. The 3-unit nonlinear network performs the best followed closely by the linear network. The more standard AR models and the random walk with drift model performed well on roughly half the series, while the linear regression and the random walk with random trend model were almost always significantly worse than the best model.

	# Times Indistinguishable From Best		
Model	3-Month MSE Win	12-Month MSE Win	Total
RW+Drift	4 / 10	7 / 10	11 / 20
RW+RTrend	0 / 10	2 / 10	2 / 20
Linear AR	6 / 10	7 / 10	13 / 20
BVAR	6 / 10	6 / 10	12 / 20
Linear Regression	0 / 10	0 / 10	0 / 20
Linear Network	8 / 10	8 / 10	16 / 20
Neural Network	8 / 10	10 / 10	18 / 20

following results are included in Appendix A. The linear regression models estimated with ordinary least squares (OLS) stand out as the worst performers for all series. The linear network and nonlinear network models are usually among the best models, due to the use of neural network-inspired estimation methods.

Table 2.3 displays the number of times each model was the best, or indistinguishable from the best, in terms of normalized mean-square error at an estimated 5% significance level using the Diebold-Mariano test (Section 2.6.3 describes how the significance test is performed). Table 2.3 indicates that while the level of noise present makes if difficult to distinguish between some of the forecasting models, the network models are consistently among the best.

Tables 2.4 and 2.5 list the best models for each of the ten series along with the models that are not significantly worse, followed by the models that are significantly worse. Among the network models, the 3-unit network is the winner for 3 and 12-month forecasts of IP, 3-month forecasts of YCS, and 12-month forecasts of FYAAAC. Note that for these last two interest rate based series, the random walk with random trend model does just as well. The linear network does better than the nonlinear network on 3-month forecasts

Table 2.4: Summary of the best 3-month point forecasts model, models that are not significantly worse, and models that are significantly worse than the best at an estimated 5% significance level.

Series	Best Model	Alternatives	Significantly Worse
DLEAD	LinNet	BVAR	RW+Drift, RW+RTrend, AR, LinReg, 3Net
DRM	LinNet	RW+Drift, AR, 3Net	RW+RTrend, BVAR, LinReg
FM2DQ	BVAR	LinNet, 3Net	RW+Drift, RW+RTrend, AR, LinReg
FSPCOM	3Net	RW+Drift, AR, LinNet	RW+RTrend, BVAR, LinReg
FYAAAC	LinNet	RW+Drift, AR, BVAR, 3Net	RW+RTrend, LinReg
HSBP	LinNet	3Net	RW+Drift, RW+RTrend, AR, BVAR, LinReg
IP	3Net	AR	RW+Drift, RW+RTrend, BVAR, LinReg, LinNet
LHUR	LinNet	BVAR, 3Net	RW+Drift, RW+RTrend, AR, LinReg
PUNEW	LinNet	AR, BVAR	RW+Drift, RW+RTrend, LinReg, 3Net
YCS	RW+Drift	AR, BVAR, 3Net	RW+RTrend, LinReg, LinNet

of PUNEW and DLEAD. On both of these the BVAR model, also a regularized linear model, does just as well. Both AR models and the linear network outperform the nonlinear network for 12-month predictions of PUNEW.

The random walk with random trend model only performs well on the 12 month forecasts of PUNEW and FM2DQ. The random walk with drift model does as well as the other models for most of the financial series, but has the lowest error for only 3-month forecasts of the yield-curve slope. Other than these financial series, the results indicate that there is nontrivial predictability in the macroeconomic series. Also, for all the series, at least one of the network models was the best model, or performed at least as well as the best model.

Table 2.5: Summary of the best 12-month point forecasts model, models that are not significantly worse, and models that are significantly worse than the best at an estimated 5% significance level.

Series	Best Model	Alternatives	Significantly Worse
DLEAD	3Net	BVAR, LinNet	RW+Drift, RW+RTrend, AR, LinReg
DRM	LinNet	RW+Drift, AR, 3Net	RW+RTrend, BVAR, LinReg
FM2DQ	BVAR	RW+Drift, RW+RTrend, AR, LinNet, 3Net	LinReg
FSPCOM	LinNet	RW+Drift, AR, BVAR, 3Net	RW+RTrend, LinReg
FYAAAC	3Net	RW+Drift, AR, BVAR	RW+RTrend, LinReg, LinNet
HSBP	3Net	RW+Drift, LinNet	RW+RTrend, AR, BVAR, LinReg
IP	3Net	AR	RW+Drift, RW+RTrend, BVAR, LinReg, LinNet
LHUR	LinNet	3Net	RW+Drift, RW+RTrend, AR, BVAR, LinReg
PUNEW	BVAR	RW+Drift, RW+RTrend, AR, LinNet, 3Net	LinReg
YCS	AR	RW+Drift, BVAR, LinNet, 3Net	RW+RTrend, LinReg

2.7.2 Nonlinear structure

Overall, evidence of nonlinear structure appears to be small. The Diebold-Mariano test identifies the decrease in error for the 3 and 12-month forecasts of the Index of Industrial Production as a significant change. There appears to be some advantage due to nonlinear structure found on the 12-month horizon for the Housing Starts Index, though the DM test does not identify this as significant.

Swanson and White [107] find little support for nonlinear structure in their study using quarterly data, however they use a stepwise model building procedure, fitting a linear component first, and then adding sigmoidal units incrementally. This type of method can significantly bias models to linear structure even when significant nonlinearity is present. Their reason for using a stepwise procedure to build nonlinear models was to avoid overfitting the training data. In this work I use regularization methods to deal with the overfitting issue, though it is worth pointing out that weight decay regularization will tend to bias the *tanh* units toward a more linear structure as well.

A possible reason for the lack of more evidence of nonlinear structure is that the high levels of noise in the data and the associated large amounts of regularization used during the model building process tend to bias the models to a linear structure. Another possibility is that while economic theories support nonlinear models, the theory of Leading Indicators, which is the basis of the model inputs used here, is largely based on observed, linear correlations between economic series. Thus, there is not necessarily a reason why the relationships between these indicators *should not* be largely linear.

2.7.3 Complexity Reduction & Model Selection

Figure 2.15 shows examples of the eigenvalue spectrum for the input and hidden layers of a model trained to make 12-month forecasts of the Index of Industrial Production. The dashed line represents the cutoff point chosen by the Principal Component Pruning algorithm. In this example, 63 of the input layer eigen-nodes and 2 of the hidden layer eigen-nodes are pruned. The number of eigen-nodes pruned is determined using the validation data set as described in Section 2.4.5.

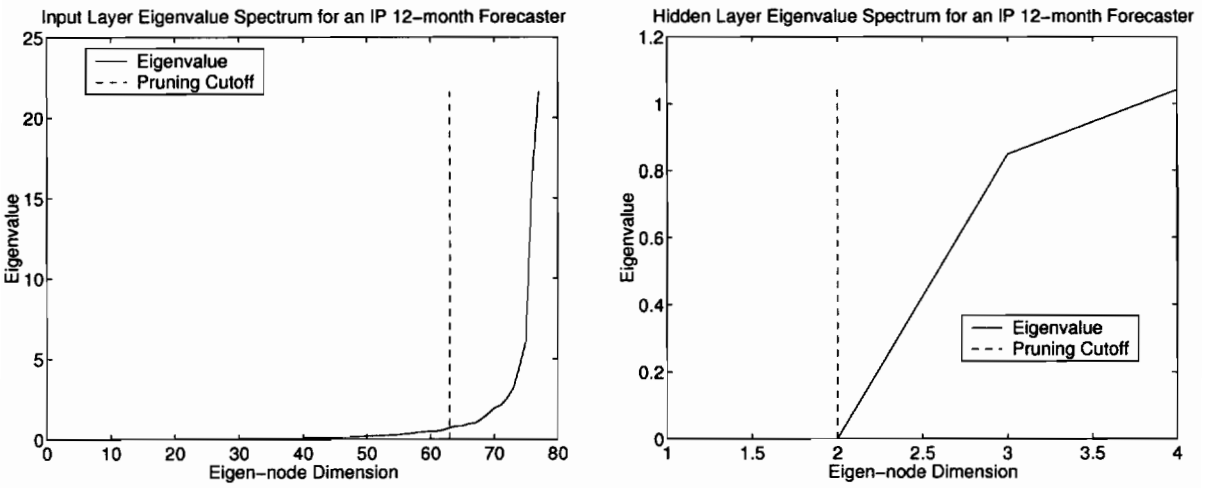

Figure 2.15: An example of the eigenvalue spectrum for the input and hidden layers of a model trained to make 12-month forecasts of the Index of Industrial Production. The dashed line represents the cutoff point chosen by the Principal Component Pruning algorithm. In this example, 63 of the input layer eigen-nodes and 2 of the hidden layer eigen-nodes are pruned.

Table 2.6 shows the average number of eigen-nodes pruned using the PCP algorithm during the testing period. The average is taken over the 10 committee members, 10 test set retrainings, and 10 time series. The average number of eigen-nodes pruned when forecasting the economic time series is less than the number pruned when forecasting the financial time series. This suggests that the economic series are more predictable given these input series, and that there is more usable nonlinear structure in the economic series. This is consistent with the other results presented here.

Table 2.6: The average number of eigen-nodes pruned using the PCP algorithm during the testing period. The average number of eigen-nodes pruned when forecasting the economic time series is less than the number pruned when forecasting the financial time series.

	3-Month Forecasters		12-Month Forecasters	
	Input Layer	Hidden Layer	Input Layer	Hidden Layer
All Series	73%	43%	80%	47%
Economic Series	65%	36%	74%	40%
Financial Series	86%	53%	90%	56%

Table 2.7: The change in normalized mean square test set errors for training the 3-unit nonlinear network models with and without the use of Principal Component Pruning. The test error was reduced in 15 out of 20 cases. The 3 changes marked with a '*' were determined to be significant by the Diebold-Mariano test. These results demonstrate that using PCP to reduce model complexity may improve forecasts and is unlikely to hurt.

Series	Forecast Horizon	All Inputs No PCP	All Inputs With PCP	% Error Reduction With PCP
DLEAD	3 month	0.534	0.580	-8.6
	12 month	0.309	0.334	-8.1
DRM	3 month	0.881	0.870	1.3
	12 month	0.785	0.787	-0.3
FM2DQ	3 month	0.601	0.612	-1.8
	12 month	0.679	0.676	0.4
FSPCOM	3 month	0.947	0.924	2.4
	12 month	1.090	1.088	0.2
FYAAAC	3 month	1.057	1.043	1.3
	12 month	1.054	1.053	0.1*
HSBP	3 month	0.719	0.664	7.7
	12 month	0.785	0.591	24.7
IP	3 month	0.655	0.649	0.9
	12 month	0.633	0.578	8.7*
LHUR	3 month	0.608	0.531	12.7*
	12 month	0.349	0.342	2.0
PUNEW	3 month	0.539	0.518	3.9
	12 month	0.686	0.598	12.8
YCS	3 month	1.051	1.086	-3.3
	12 month	1.128	1.078	4.4

Table 2.7 shows a comparison on a sample of the series between using and not using Principal Component Pruning during training of the 3-unit nonlinear network models. The use of Principal Component Pruning reduces the test set error in 15 out of 20 cases. Three of the reductions were considered significant by the Diebold-Mariano test, while none of the increases in error were determined to be significant.

It is interesting to note that the DM test identifies the small improvement in 12-month forecasts of FYAAAC to be a significant improvement. This is because the Principal Component Pruning algorithm was very effective in reducing the complexity of the model. Figure 2.16 shows the resulting forecasts as PCP essentially reduced the 12-month FYAAAC

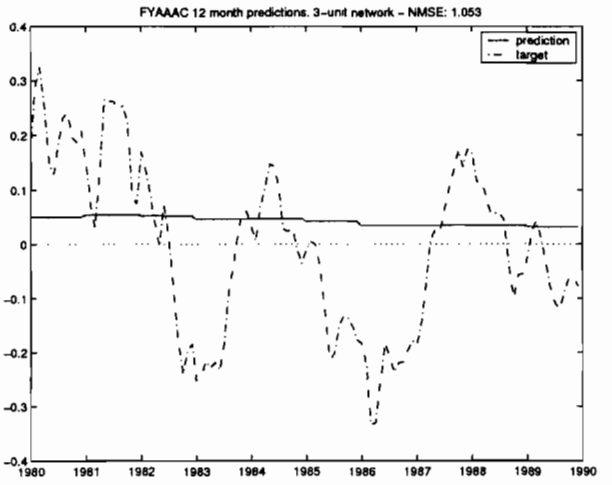

Figure 2.16: The training methodology reduced the 12-month FYAAAC NN model to a random walk with drift. This result is consistent with standard results in finance [45]. The nonzero forecast values are due to the fact that the in-sample drift is positive.

NN model to a random walk with drift. This result is consistent with standard results in finance [45]. The pruning procedure effectively reduced the second layer of the networks to constant models, and learned a bias value. The nonzero forecast values are due to the fact that the in-sample drift is positive.

2.7.4 Regularization Methods

Figures 2.17 and 2.18 show the results from varying which regularization techniques are used during the training of the 3-tanh unit nonlinear networks. The "NoReg" models are unregularized, the "E.S." models correspond to only using early stopping for regularization, the "W.D." models use only weight decay regularization, and the "Both" models use both early stopping and weight decay together. The figures show that just using early stopping alone can provide a large enhancement over not using any regularization at all. Also, using weight decay regularization alone is very effective, usually more so than early stopping. I find that using early stopping in addition to weight decay usually does not help much except for the 12-month forecasts of the Yield-Curve Slope. Early stopping alone seems to be more effective on the 3-month horizon than on the 12-month horizon

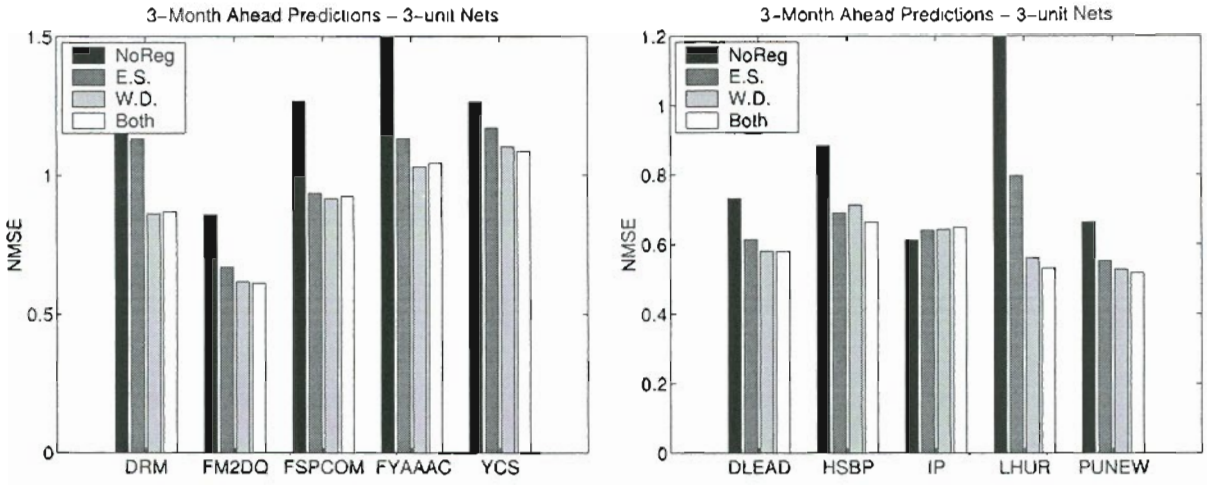

Figure 2.17: The results for leaving out the regularization techniques when training 3-hidden unit networks to make 3-month forecasts. The "NoReg" models are unregularized, the "E.S." models correspond to only using early stopping for regularization, the "W.D." models use only weight decay regularization, and the "Both" models use both early stopping and weight decay together.

relative to weight decay alone. This may be because there are effectively more independent data points in the validation data set for the 3-month forecasts compared to the 12-month forecast. Since the validation data set contains the same number of points in both cases, the targets for the 3-month forecasts will tend to overlap less frequently, and thus may provide a better estimate of out-of-sample performance.

2.7.5 Model Insight via Sensitivity Analysis

I use a sensitivity analysis to open the neural network black box and examine how the way the networks use the input variables changes over time. Sensitivity analysis can be used to gain insight into the relationships learned by the neural network models.

I define the relative sensitivity of input i as:

$$S_i = \frac{\sum_t \left| \frac{dF}{dx_i} \right|}{\max_j \left(\sum_t \left| \frac{dF}{dx_j} \right| \right)} \tag{2.53}$$

where F is the model output and x_i denotes input i.

The sensitivity is calculated for each input for each year and averaged over the 10 committee members. The sensitivity value for each input is then normalized by maximum

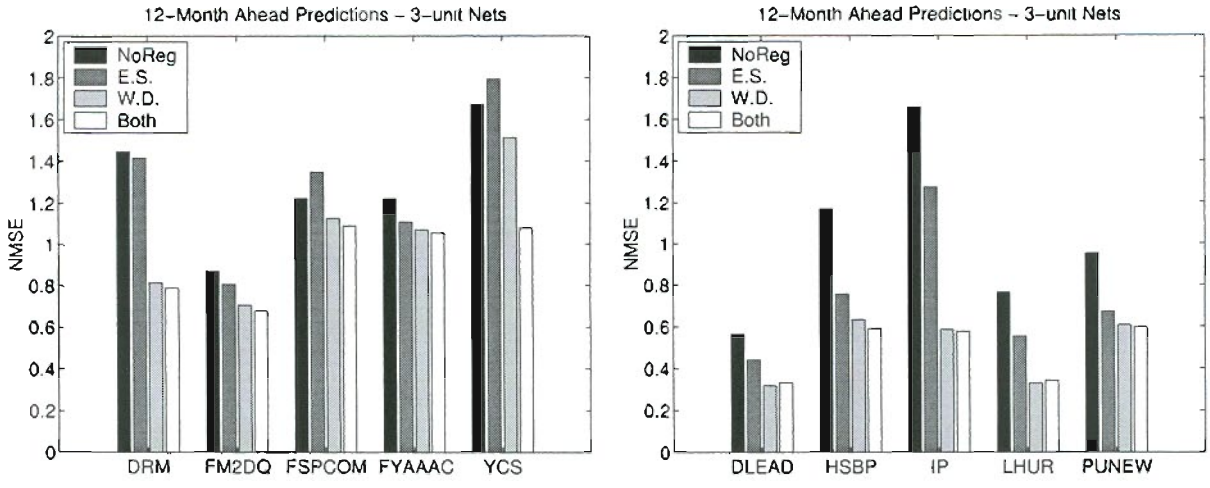

Figure 2.18: The results for leaving out the regularization techniques when training 3-hidden unit networks to make 12-month forecasts. The "NoReg" models are unregularized, the "E.S." models correspond to only using early stopping for regularization, the "W.D." models use only weight decay regularization, and the "Both" models use both early stopping and weight decay together.

sensitivity value calculated for that year. Thus a relative sensitivity value close to 1 means that the input was highly influential for the predictions made during that year while a value close to 0 means that changes in that input have relatively very little influence on the output of the model during that time period.

Figure 2.19 shows input sensitivity traces for several of the inputs to a 3-unit neural network trained to make 12-month forecasts of the Index of Industrial Production over the period 1970-1989. A high degree of nonstationarity in the underlying relationship between the input and target variables is evident in the figure. This example shows that 12-month changes in the S&P-500 stock index were highly weighted during the 1970's, but then were largely ignored during the 1980's. Also, the importance of the 9-month changes in the Consumer Price Index to the model increased steadily over this time period.

While Figure 2.19 seems to indicate the presence of significant nonstationarity in the datasets, Swanson and White [107] conclude that the amount of nonstationarity in the macroeconomic series they studied is small based on their finding that longer training windows are preferable to shorter windows. Another explanation for their result can be found by referring to the *noise / nonstationarity* tradeoff discussed in Section 2.2.4. Given

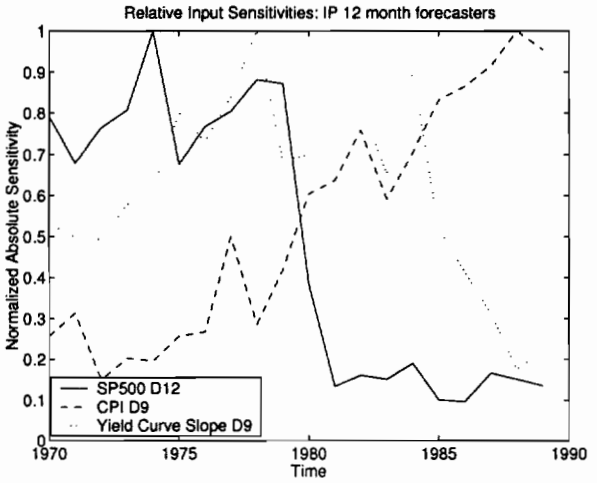

Figure 2.19: Input sensitivity traces for several of the inputs to a 3-unit neural network trained to make 12-month forecasts of the Index of Industrial Production over the period 1970-1989. The figure shows a large amount of variation in how the model uses these inputs. These nonstationary relationships highlight the challenges inherit in macroeconomic forecasting.

the extremely large amounts of noise present in their "first-reported" quarterly dataset, it is quite likely that the noise component of the error dominated the nonstationarity contribution.

2.8 Empirical Studies: Directional and Quintile Forecasts

Aside from making actual point forecasts, even being able to forecast only the future direction of a macroeconomic or financial series adds value to the decision making process. In this section I present results for models trained to forecast directional movements of the series.

2.8.1 Representations for Directional Forecasts

One reason to try to forecast directional movements is that quantization of the target variables may result in a more robust representation for prediction by removing a certain amount of noise from the targets. The targets can be quantized into varying degrees

of resolution. In this section, the targets are quantized into quintiles representing "large down", "small down", "no change", "small up", and "large up" as measured on the training dataset. The first quintile is defined to contain the lowest 20% of the data. The second quintile contains the data falling within the 20%–40% range, and so on. Thus the third quintile, "no change" corresponds to the median return and small deviations from it, and the fifth quintile corresponds to large positive returns relative to the training set. The quintiles were determined using the data prior to 1980, and not changed thereafter.

There were several different approaches considered to solving this classification problem. The first was using a unary output representation. The target classes would be represented as follows:

```
class 1 --- 1 0 0 0 0
class 2 --- 0 1 0 0 0
class 3 --- 0 0 1 0 0
class 4 --- 0 0 0 1 0
class 5 --- 0 0 0 0 1     .
```

The network models were trained with 5 output nodes, one for each class. The output of the forecasting model would be calculated using the softmax [17] representation

$$p_i = \frac{exp(f_i)}{\sum_{j=1}^{5} exp(f_j)} \quad , \tag{2.54}$$

where f_i is the i^{th} output of a network with a similar structure as described in Section 2.4.1. p_i then represents the posterior probability of quintile i given the current input vector. The class chosen by the network is then determined as $f^c = argmax_i(p_i)$. The output of the committee is the class voted for by the most committee members.

A drawback of using the unary representation is that the original relationship between classes is not exploited. In other words, it is likely that economic conditions that would cause a large relative increase (class 5) in IP are similar to conditions that would cause a moderate relative increase (class 4), but are largely different from conditions that would cause a large relative decrease (class 1) in IP. However, the unary target representation shown above does not contain any of this relative information. In other words, the target representation could be reassigned to arbitrary classes prior to training, and this would

have no effect on the final solution produced. It would be desirable to be able to incorporate all available information in the training procedure including the relationships between classes.

A simple method for doing this is to use a "thermometer code". The thermometer code represents the target classes as:

```
class 1 --- 0 0 0 0
class 2 --- 1 0 0 0
class 3 --- 1 1 0 0
class 4 --- 1 1 1 0
class 5 --- 1 1 1 1     .
```

With this representation, the Hamming distance between adjacent classes is equal to 1 and increments monotonically with the distance between classes. The first representation has a Hamming distance of exactly 2 between all classes. This feature of the target representation should result in an effect on the internal representation that is learned by the predictors. Training with these targets to minimize mean-square error should result in improved classification accuracy, and indeed the results show a marked improvement using the thermometer code as opposed to the softmax outputs. The class for each committee member is determined by $f^c = argmax_i(f_i)$, and the output of the committee is the class voted for by the most committee members.

2.8.2 Network Models for Directional Forecasting

The nonlinear network models considered for the directional forecast problem have the same general form as described in Section 2.4.1, but have multiple outputs. Output i is calculated as

$$f_i = v_{i0} + \sum_{j=1}^{n} v_{ij} \tanh\left(u_{j0} + \sum_{k=1}^{m} u_{jk} x_k\right) \quad , \tag{2.55}$$

where the x_k are the input signals, u_{jk} are the weights connecting the inputs to the internal tanh units, and v_{ij} are the weights connecting the internal units to the ith output. Due to the increased complexity in the output representation, I consider nonlinear models with both $n = 3$ and 10 internal tanh units.

2.8.3 Comparison Models

I compare these directional forecast models to results from quantizing the point forecasts made by the network and benchmark models. The quantization of the point forecasts uses the same quintile definitions measure on the training set as described in Section 2.8.1. The point forecasts are given class labels depending on which quintile contains the forecasted value.

I include quantized forecasts for all of the point forecast models described in Section 2.7. The random-walk-with-drift and random-walk-with-random-trend models give the same result whether they are defined on the actual returns and quantized, or just calculated on the quantized returns.

2.8.4 Directional Forecast Results Summary

The directional forecasting results presented are stated in terms of the Mean Square Error based on the quintile classes. That is, if $class(\cdot)$ is the function that takes the quintile representation used by the models and transforms it into an ordinal class, then

$$\text{Quintile MSE} = \frac{1}{N}\sum_{i=0}^{N}(class(targ_i) - class(F(\mathbf{x}_i)))^2 \quad , \tag{2.56}$$

where F is the committee prediction. Note that the class errors are not normalized as the point forecast errors are. The class errors are to be interpreted in terms of the class differences, so a model that could forecast the median return on the test set would have an MSE of 2 if the median return corresponded to the median return measured prior to 1980. The significance results were calculated at the 5% level using the Diebold-Mariano test as described previously in Section 2.6.3.

Figures 2.20 and 2.21 show the performance comparison between the unary and thermometer code representations graphically. Tables of numeric values are given in Appendix A along with the point forecast results. Table 2.8 also summarizes the classification results by model type including the quantized point forecasts. Both of the quantized network models and the directional forecast networks that use the thermometer code representation perform well on average, being competitive in a total of 15 of the 20 series.

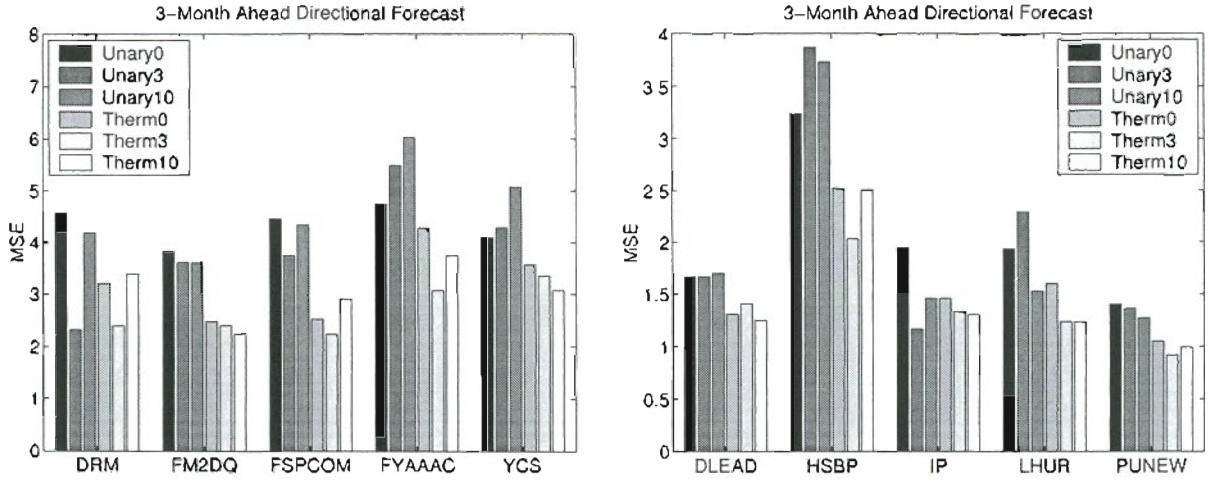

Figure 2.20: Comparison of the normalized errors for making 3-month directional forecasts for the two class representations, the unary and thermometer code. In most cases, the thermometer code representation produces superior models.

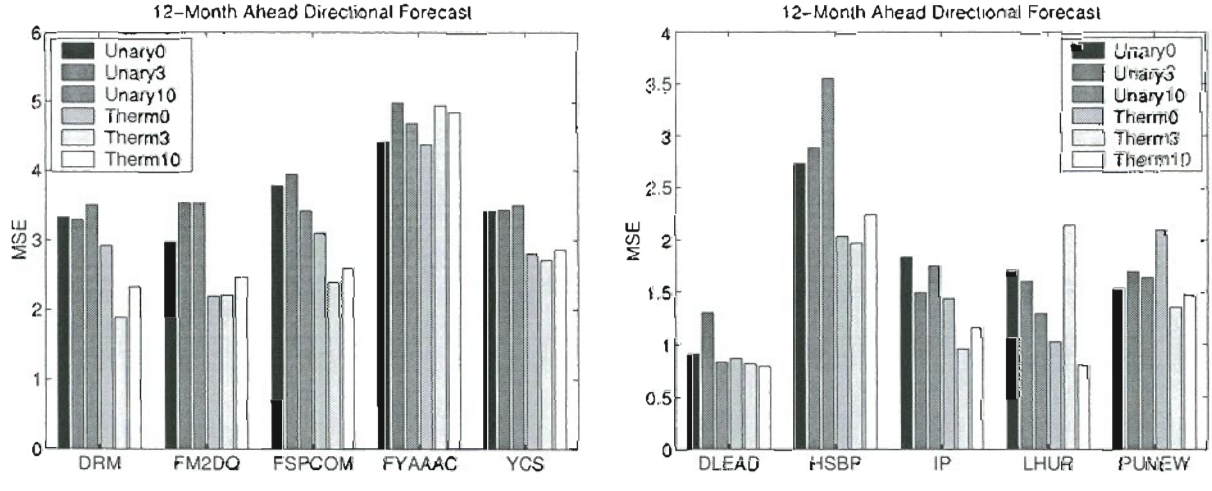

Figure 2.21: Comparison of the normalized errors for making 12-month directional forecasts or the two class representations, the unary and thermometer code. In most cases, the thermometer code representation produces superior models.

Table 2.8: Summary of the number of times each directional forecast model was not significantly worse than the best model at an estimated 5% significance level. Both of the quantized network models and the directional forecast networks that use the thermometer code representation perform well on average, and outperform the random walk with drift model on average.

Model	# Times Indistinguishable From Best 3-Month MSE Win	12-Month MSE Win	Total
RW+Drift	4 / 10	7 / 10	11 / 20
RW+RTrend	1 / 10	4 / 10	5 / 20
Quant. Linear AR	4 / 10	5 / 10	9 / 20
Quant. BVAR	3 / 10	7 / 10	10 / 20
Quant. Linear Regression	3 / 10	3 / 10	6 / 20
Quant. Linear Network	6 / 10	8 / 10	14 / 20
Quant. 3-Unit Neural Network	6 / 10	9 / 10	15 / 20
Unary Linear Network	2 / 10	3 / 10	5 / 20
Unary 3-Unit Neural Network	4 / 10	2 / 10	6 / 20
Unary 10-Unit Neural Network	4 / 10	6 / 10	10 / 20
Therm. Linear Network	6 / 10	8 / 10	14 / 20
Therm. 3-Unit Neural Network	8 / 10	7 / 10	15 / 20
Therm. 10-Unit Neural Network	5 / 10	6 / 10	11 / 20

While the random walk with drift model performs well in 11 of the 20 series, there does seem to be some additional predictable structure found by the network models.

2.8.5 Target Representation Comparison

Table 2.11 compares the performance of the directional forecast models using the unary and thermometer code representations. The additional information being encoded in the thermometer code gives these models a clear advantage over those that use the naive unary representation. The use of the thermometer code also reduced the number of model parameters because it only required 4 output units as compared to 5 for the unary representation. The thermometer code models are competitive in over 15 of the series while the unary representation models are competitive in only 8.

2.8.6 Quantized Regression vs Directional Forecasters

Table 2.8 compares the performance of the quantized regression forecast models and the directional forecast models. The linear and 3-unit neural network quantized regression models are competitive on balance with the corresponding thermometer code directional forecast models. This suggests that the additional flexibility of the thermometer code networks is not usually needed to achieve good forecasts of direction.

2.9 Discussion

I find non-trivial predictability in the forecasts for all series considered except the Aaa-bond yield, the S&P-500, and the yield-curve slope. There is also evidence of nonlinear structure in the Index of Industrial Production and possibly Housing Starts. While most theories of the economy are nonlinear, it seems difficult in practice to exploit these nonlinearities. There are a variety of factors that could make nonlinearity difficult to detect in the remaining series. Among them are the large amounts of noise in the series, the resulting need for heavy regularization, and the types of inputs being used. Many of the series included in this study are higher level, more broad based series than the ones that structural economic theories use as explanatory variables. However the focus of this study

Table 2.9: Summary of the best 3-month directional forecast model, models that are not significantly worse, and models that are significantly worse than the best at an estimated 5% significance level.The prefix "Q" denotes a quantized regression model, "T" denotes the use of the thermometer code class representation during training, and "U" denotes the use of the unary class representation.

Series	Best Model	Alternatives	Significantly Worse
DLEAD	Q3Net	RW+Drift, QBVAR, QLinReg, QLinNet, ULinNet, U3Net, U10Net, TLinNet, T3Net, T10Net	RW+RTrend, QLinAR
DRM	U3Net	QLinNet, Q3Net, TLinNet, T3Net	RW+Drift, RW+RTrend, QLinAR, QBVAR, QLinReg, ULinNet, U10Net, T10Net
FM2DQ	QLinReg	RW+RTrend, QLinAR, ULinNet, U3Net, U10Net, TLinNet, T3Net	RW+Drift, QBVAR, QLinNet, Q3Net, T10Net
FSPCOM	T3Net	RW+Drift, QLinAR, QLinNet, Q3Net, TLinNet, T10Net	RW+RTrend, QBVAR, QLinReg, ULinNet, U3Net, U10Net
FYAAAC	RW+Drift	QLinNet, Q3Net, T3Net	RW+RTrend, QLinAR, QBVAR, QLinReg, ULinNet, U3Net, U10Net, TLinNet, T10Net
HSBP	QLinNet	Q3Net	RW+Drift, RW+RTrend, QLinAR, QBVAR, QLinReg, ULinNet, U3Net, U10Net, TLinNet, T3Net, T10Net
IP	QLinNet	QLinAR, QLinReg, Q3Net, U3Net, U10Net, T3Net, T10Net	RW+Drift, RW+RTrend, QBVAR, ULinNet, TLinNet
LHUR	T3Net	QBVAR, U10Net, TLinNet, T10Net	RW+Drift, RW+RTrend, QLinAR, QLinReg, QLinNet, Q3Net, ULinNet, U3Net
PUNEW	T3Net	QLinAR, QBVAR, TLinNet, T10Net	RW+Drift, RW+RTrend, QLinReg, QLinNet, Q3Net, ULinNet, U3Net, U10Net
YCS	RW+Drift		RW+RTrend, QLinAR, QBVAR, QLinReg, QLinNet, Q3Net, ULinNet, U3Net, U10Net, TLinNet, T3Net, T10Net

Table 2.10: Summary of the best 12-month directional forecast model, models that are not significantly worse, and models that are significantly worse than the best at an estimated 5% significance level. The prefix "Q" denotes a quantized regression model, "T" denotes the use of the thermometer code class representation during training, and "U" denotes the use of the unary class representation.

Series	Best Model	Alternatives	Significantly Worse
DLEAD	QLinNet	QBVAR, Q3Net, U10Net, TLinNet, T3Net, T10Net	RW+Drift, RW+RTrend, QLinAR, QLinReg, ULinNet, U3Net
DRM	QBVAR	RW+RTrend, QLinReg, ULinNet, U3Net, U10Net	RW+Drift, QLinAR, QLinNet, Q3Net, TLinNet, T3Net, T10Net
FM2DQ	Q3Net	RW+Drift, RW+RTrend, QBVAR, QLinReg, TLinNet	QLinAR, QLinNet, ULinNet, U3Net, U10Net, T3Net, T10Net
FSPCOM	T3Net	RW+Drift, QLinAR, QBVAR, QLinNet, Q3Net, U10Net, TLinNet, T10Net	RW+RTrend, QLinReg, ULinNet, U3Net
FYAAAC	RW+Drift	RW+RTrend, QLinAR, QBVAR, QLinReg, QLinNet, Q3Net, ULinNet, U10Net, TLinNet, T3Net, T10Net	U3Net
HSBP	Q3Net	RW+Drift, QLinNet, TLinNet, T3Net, T10Net	RW+RTrend, QLinAR, QBVAR, QLinReg, ULinNet, U3Net, U10Net
IP	T3Net	RW+Drift, QLinAR, QLinNet, Q3Net, U3Net, TLinNet	RW+RTrend, QBVAR, QLinReg, ULinNet, U10Net, T10Net
LHUR	T10Net	QLinNet, Q3Net, U10Net, TLinNet	RW+Drift, RW+RTrend, QLinAR, QBVAR, QLinReg, ULinNet, U3Net, T3Net
PUNEW	QBVAR	RW+Drift, RW+RTrend, QLinAR, QLinNet, Q3Net, ULinNet, U10Net, T3Net	QLinReg, U3Net, TLinNet, T10Net
YCS	RW+Drift	QLinAR, QBVAR, QLinNet, Q3Net, TLinNet, T3Net, T10Net	RW+RTrend, QLinReg, ULinNet, U3Net, U10Net

Table 2.11: Comparison of the directional forecast target representation types. The table counts the number of times each directional forecast model was not significantly worse than the best model at an estimated 5% significance level. The additional class ordering information included in the thermometer code representation gives these models an advantage over those using a straight unary class representation.

Model	# Times Indistinguishable From Best 3-Month MSE Win	12-Month MSE Win	Total
RW+Drift	7 / 10	8 / 10	15 / 20
RW+RTrend	1 / 10	3 / 10	4 / 20
Unary Linear Network	0 / 10	5 / 10	5 / 20
Unary 3-Unit Neural Network	2 / 10	4 / 10	6 / 20
Unary 10-Unit Neural Network	2 / 10	6 / 10	8 / 20
Therm. Linear Network	7 / 10	8 / 10	15 / 20
Therm. 3-Unit Neural Network	9 / 10	9 / 10	18 / 20
Therm. 10-Unit Neural Network	7 / 10	8 / 10	15 / 20

was on the broader indicators of economic activity.

One feature the forecast models exhibit is a large amount of nonstationarity in the learned relationship between the target and input variables. Swanson and White [107] conclude that nonstationarity is not a factor in the models they build because the largest available training window they used was preferable to smaller windows. However, through the lens of sensitivity analysis, the relationships between variables learned by the models is seen to change significantly over the course of the test period.

The results show that applying nonlinear estimation and model selection techniques to linear models results in large improvements over more standard linear model fitting techniques. This includes regularized models such as the Bayesian Vector Autoregression. This result argues strongly for the use of appropriate linear models for comparison when fitting nonlinear models to these types of series. It could be that positive results in the literature from using nonlinear network models are due more to the estimation methodology than to actual nonlinear structure.

When making directional forecasts, the models that incorporated class-ordering information in the target representation via use of a thermometer code representation yielded

significantly better results than those using a naive unary representation. Also, the regression network models trained on point forecasts perform as well in terms of directional forecasts as do the more complex classification models trained to directly forecast class labels.

Chapter 3

Knowledge Discovery Through Reinforcement Learning

3.1 Introduction

In this chapter I present enhancements to the Recurrent Reinforcement Learning (RRL) algorithm proposed by Moody and Wu [74] with the goal of making the trading systems trained via RRL more useful and desirable to investment managers. These enhancements include methods for optimizing portfolios, asset allocations, and trading systems using risk-adjusted performance measures that more accurately reflect investor preferences. Also, I show that the direct policy representation of RRL produces more stable, interpretable trading decisions than that of a value function reinforcement learning approach called Q-Learning.

In the RRL approach, investment decision making is viewed as a stochastic control problem, and strategies are discovered directly. The need to build forecasting models is eliminated, and better trading performance is obtained by directly optimizing the relevant performance measure. The direct reinforcement approach differs from dynamic programming and reinforcement algorithms such as TD-learning and Q-learning, which attempt to estimate a value function for the control problem. The RRL direct reinforcement framework enables a simpler problem representation, avoids Bellman's curse of dimensionality and offers compelling advantages in efficiency.

My contributions presented in this chapter include RRL systems trained using downside risk measures (Sections 3.6 and 3.9.2). I also present a portfolio management system

that allocates among several artificial assets (Section 3.9.6). In other simulation work using real financial data, the RRL approach is compared to an alternative RL value function approach utilizing Q-Learning on a monthly asset allocation system for the S&P 500 Stock Index and T-Bills (Section 3.9.4). A simple example, the Oracle problem, is used to illustrate the difference between the compared approaches (Section 3.8.3). Another real world application I present in this chapter includes an intra-daily currency trader (Section 3.9.3). Much of this work has been previously published in journal articles [72, 71].

In the remainder of this section I will briefly overview the main types of reinforcement learning, and contrast reinforcement learning and supervised learning. Section 3.2 introduces the financial problem domain in which the algorithms will be tested, and Section 3.3 reviews the pertinent literature. Section 3.4 will review the recurrent reinforcement learning model of Moody and Wu [74] and describe the single asset trader setup. In Section 3.5 I present the setup for managing a portfolio of assets. In Section 3.6 I introduce downside risk measures and formulate the downside deviation in a manner compatible with the recurrent reinforcement learning setup. In Section 3.7 I discuss the problem of learning how to trade and the means by which the different types of reinforcement learning approach the problem, and in Section 3.8 compare the policy and value approaches for reinforcement learning. In Section 3.9 I present results from a number of simulations demonstrating the performance of the downside deviation ratio and portfolios, and comparing the recurrent reinforcement learning algorithm to a Q-Learning algorithm. In Section 3.10 I conclude by reviewing the main results of the chapter.

3.1.1 Reinforcement Learning

Reinforcement Learning (RL) is a learning paradigm that involves a goal-directed agent actively exploring an unknown environment, attempting to maximize rewards received from the actions taken. RL falls between supervised and unsupervised learning in the sense that it does not require the feedback of a knowledgeable teacher as in supervised learning, nor does it mechanically process data in the way that unsupervised learning does. Instead, RL uses indirect or limited feedback regarding the correctness of actions taken. In some cases, the rewards received can be substantially delayed from the associated actions

that produced them, and can depend on a sequence of interdependent actions. The need to assign credit among various actions over time is known as the *temporal credit assignment problem.* This ability to interact with (and potentially influence) the environment, and having to account for delayed rewards makes RL a fundamentally different type of learning problem.

The most general framework for an RL problem involves an environment consisting of a set of (possibly continuous) states through which an actor transits by following some policy for taking actions. The actor receives feedback based on the states it has visited and the actions which it has taken. The goal of RL typically is to maximize a function of the feedback (or reward) signal. The feedback signals in response to an action may be presented immediately following the action, or they may be significantly delayed or spread out across time. Typically the actor must utilize trial-and-error search to explore the environment as it seeks to maximize its feedback.

There are several major categories of RL methods: value methods, policy methods, and actor-critic methods. Most value and actor-critic methods involve some form of approximate dynamic programming.

Value Methods

Value methods involve learning a representation of the expected value of the reward or discounted reward. This representation is known as the "value function". The value function enumerates the expected value of future reward signals for each possible system state given a policy for deciding how to move between states. The value function is usually encoded in a lookup table or a function approximator. Policies are usually represented implicitly in value functions. That is, the best action for the current state is the one that takes you to the state with the highest value associated with it. As described in Section 3.7 the value function can be used to improve the policy. One of the advantages to a value function approach is that there are a number of proofs for the convergence of certain value function algorithms. A popular value function method is Q-Learning [118]. Q-Learning encodes the policies explicitly in the value function, and will be discussed further in Section 3.7.

Policy Methods

Policy methods (also called Direct Reinforcement) involve learning the representation of the policy function directly without having to learn a value function. Policy methods are most useful when there is immediate feedback on the results of actions taken, or when feedback delays are not large. Policy methods update the parameters of a *policy function* directly based on feedback from the environment. This often results in the use of gradient ascent to maximize the feedback signal. Often, a direct representation of the policy function is much more natural and less complex than implicitly representing the policy using a value function. The Recurrent Reinforcement Learning algorithm [74] is a policy method that incorporates recurrence in order to accurately account for the effect of previous actions. I compare value and policy methods in Section 3.8.

Actor-Critic Methods

Actor-critic methods involve explicitly representing both the policy function and the value function. The policy function is known as the "actor" and the "critic" learns a value function to provide feedback to the actor. Feedback from the critic is used to update the parameters of the actor. Actor-critic methods use a *temporal difference* signal, which is the difference between the estimate of the value for the current state and the estimate of the value for the previous state. Thus the policy function is being updated while the value function is still being learned.

3.1.2 Reinforcement vs Supervised Learning

[1]A block diagram for a generic trading system based on forecasts is shown in Figure 3.1. In such a system, a forecast module is optimized to produce price forecasts from a set of input variables. Supervised learning techniques are used to minimize forecast error (typically mean-squared-error) on a training sample. The forecasts are then used as input to a trading module that makes buy and sell decisions or adjusts portfolio weights. Parameters

[1] "Performance Functions and Reinforcement Learning for Trading Systems and Portfolios", J. Moody, L. Wu, Y. Liao, and M. Saffell, *Journal of Forecasting*, vol. 17, no. 5/6,

of the trading module may be optimized, but are often set by hand.

Trading based on forecasts involves two steps, and minimizing forecast error is an *intermediate step* that is not the ultimate objective of the system. Moreover, the common practice of using only the forecasts as input to the trading module results in a loss of information relative to that available to the forecast module, in effect producing a *forecast bottleneck*. Both of these effects may lead to suboptimal performance.

It is possible to train a system to make trading decisions directly from the input variables, while avoiding the intermediate step of making forecasts. This is more direct, and avoids the forecast bottleneck. One technique for optimizing such a system is to use a supervised learning algorithm to train the system to make desired trades, as shown in Figure 3.2. A sequence of desired target trades (or portfolio weights) used for training the system is first determined via a *labeling procedure*. The data can be labeled by a human "expert" or by an automatic labeling algorithm. The labeled trades are then used to train the trading system.

Training on labeled data is a two-step process. The procedure for labeling the data attempts to solve the *temporal credit assignment* problem, while subsequently training the system on the labeled data attempts to solve the *structural credit assignment* problem.[2] Certain difficulties arise when trying to solve the structural and temporal credit assignment problems separately in this way, particularly when transaction costs are included.

The performance achievable in practice by the trading module will usually be substantially worse than that suggested by the labeled trades. This is because most labeling procedures are based on only the target series (possibly taking into account transaction costs), ignore the input variables and do not consider the conditional distributions of price changes in the target series given the input variables. Moreover, since transactions costs depend upon the actual sequence of trades made, the simulated costs associated with the labeled trades will differ from those incurred in practice. Hence, a labeling procedure is not likely to give rise to a sequence of trades that is realizable in practice or to a realistic assessment of the actual transaction costs likely to occur. Finally, since $U(\theta, \theta')$ is not

[2]This terminology was proposed by Sutton [103].

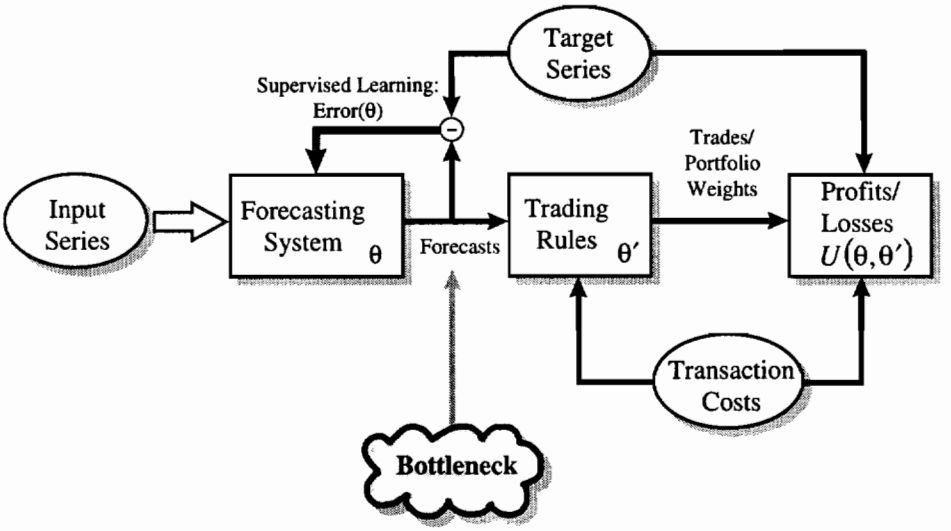

Figure 3.1: A trading system based on forecasts. The system includes a forecast module with adjustable parameters θ followed by a trading module with parameters θ'. Price forecasts for the target series are based on a set of input variables. The forecast module is trained by varying θ to minimize forecast error (typically mean squared error), which is an *intermediate quantity*. A more direct approach would be to simultaneously vary θ and θ' to maximize a measure of ultimate performance $U(\theta, \theta')$, such as trading profits, utility or risk-adjusted return. Note that the trading module typically does not make use of the inputs used by the forecast module, resulting in a loss of information or a *forecast bottleneck*. Performance of such a system is thus likely to be suboptimal.

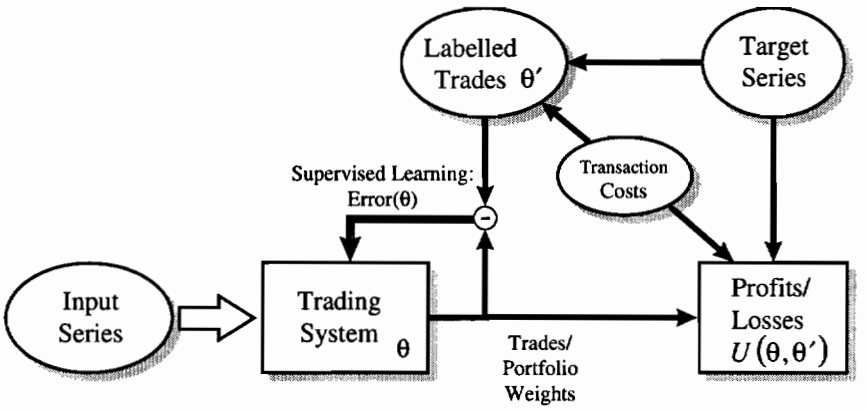

Figure 3.2: A trading system trained with labeled data. The system includes a trading module with parameters θ and a labeling procedure with parameters θ'. Trades are based on a set of input variables. Target trades are produced by the labeling procedure. The trading module is trained on the labeled trades using a supervised learning approach to vary θ. The ultimate performance of the system depends upon how good the labeling algorithm is (as determined by θ'), and how well the trading module can learn to trade (by varying θ) using the input variables and labeled trades. Since the ultimate measure of performance $U(\theta, \theta')$ is not used to optimize θ directly, performance of such a system is thus likely to be suboptimal.

optimized directly (see Figure 3.2), supervised learning based on labeled data will yield suboptimal performance.

A trading system can be optimized to solve both the temporal credit assignment and structural credit assignment problems mentioned above *simultaneously* using *reinforcement learning* (see for example [104] and references therein).

Reinforcement learning algorithms find approximate solutions to stochastic dynamic programming problems [9] and can do so in an on-line mode. In reinforcement learning, target outputs are not provided. Rather, the system takes actions (makes trades), receives feedback on its performance (an evaluation signal) and then adjusts its internal parameters to increase its future rewards. With this approach, an ultimate measure of trading performance $U(\theta)$, such as profit, utility or risk-adjusted return is optimized directly. See Figure 3.3.

A simultaneous solution of the structural and temporal credit assignment problems will generally require using a *recurrent* learning algorithm. Trading system profits depend upon sequences of interdependent decisions, and are thus path-dependent. Optimal trading decisions when the effects of transactions costs, market impact and taxes are included require knowledge of the current system state. Including information related to past decisions in the inputs to a trading system results in a *recurrent* decision system.[3] The proper optimization of a recurrent, path-dependent decision system is quite different from the simple supervised optimization techniques used for direct forecasts or for labeled trading data.

Reinforcement learning analogs of recurrent learning algorithms are required to train the systems. Such recurrent learning algorithms include both off-line (batch) training algorithms like backpropagation through time (BPTT) [91, 119] or on-line (adaptive) algorithms such as real-time recurrent learning (RTRL) [125] or dynamic backpropagation [78]. The recurrent reinforcement learning algorithms utilized here are variants of the

[3]Here, recurrence refers to the nature of the algorithms required to optimize the system. For example, optimizing a feed forward NAR(p) model for one-step-ahead prediction does not require a recurrent learning algorithm, while optimizing the same NAR(p) model to perform iterated predictions *does*. The fact that a forecast or decision is made by a feedforward, non-recurrent network does not mean that optimizing it correctly can be done with a standard, non-recurrent training procedure.

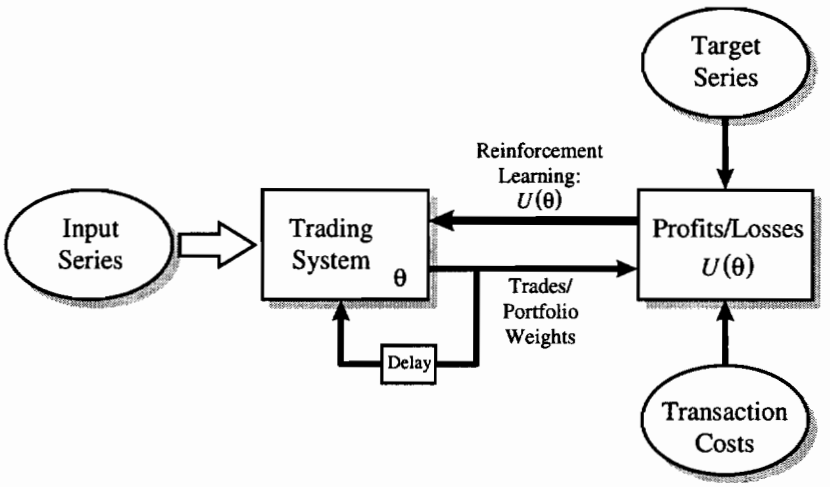

Figure 3.3: A trading system based on recurrent reinforcement learning, the approach taken in this chapter. The system makes trading decisions directly based upon a set of input variables and the system state. A trading performance function $U(\theta)$, such as profit, utility or risk-adjusted return, is used to *directly optimize* the trading system parameters θ using reinforcement learning. The system is recurrent; the feedback of system state (current positions or portfolio weights) enables the trading system to learn to correctly incorporate transactions costs into its trading decisions. In comparison to the systems in Figures 3.1 and 3.2, no intermediate steps such as making forecasts or labeling desired trades are required.

above mentioned algorithms, and maximize immediate rewards in an on-line fashion.

3.2 Problem Domain: Financial Decision Making

[4]The investor's or trader's ultimate goal is to optimize some relevant measure of trading system performance, such as profit, economic utility or risk-adjusted return. Investment performance depends upon sequences of interdependent decisions, and is thus path-dependent. Optimal trading or portfolio rebalancing decisions require taking into account the current system state, which includes both market conditions and the currently held positions. Market frictions, the real-world costs of trading,[5] make arbitrarily frequent trades or large changes in portfolio composition become prohibitively expensive. Thus, optimal decisions about establishing new positions must consider current positions held.

One class of performance criteria frequently used in the financial community are measures of risk-adjusted investment returns. RRL can be used to learn trading strategies that balance the accumulation of return with the avoidance of risk. The commonly used measures of risk in finance is the standard deviation of returns. Later in this chapter, I present the *Downside Deviation Ratio*, a risk-adjusted performance measure that is more in line with the typical investor's risk preferences in that it does not penalize large positive returns in the way that a symmetric risk measure does. The investor's utility also depends on the performance of all the assets held. To optimize this utility correctly requires a portfolio management system that makes decisions for individual assets in the context of the entire portfolio.

3.3 Literature Review

The term "Direct Reinforcement" is used to refer to algorithms that *do not* have to learn a value function in order to derive a policy. Direct Reinforcement methods date back to the pioneering work by Farley and Clark [19, 20], but have received little attention from

[4]©2001 IEEE. Reprinted, with permission, from "Learning to Trade via Direct Reinforcement", John Moody and Matthew Saffell, *IEEE Transactions on Neural Networks*, July 2001, vol. 12, no. 4.

[5]Market frictions include taxes and a variety of transaction costs, such as commissions, bid / ask spreads, price slippage and market impact.

the reinforcement learning community during the past two decades. Notable exceptions are Williams' REINFORCE algorithm [123, 124] and Baxter & Bartlett's recent work [6].[6]

Methods such as dynamic programming [7], TD-Learning [103] or Q-Learning [118, 117] have been the focus of most of the modern research. These methods attempt to learn a value function or the closely related Q-function. Such value function methods are natural for problems like checkers or backgammon where immediate feedback on performance is not readily available at each point in time. Actor-critic methods [5, 4] have also received substantial attention. These algorithms are intermediate between Direct Reinforcement and Value Function methods, in that the "critic" learns a value function which is then used to update the parameters of the "actor".[7]

Though much theoretical progress has been made in recent years in the area of value function learning, there have been relatively few widely-cited, successful applications of the techniques. Notable examples include TD-gammon [109, 110], an elevator scheduler [23] and a space-shuttle payload scheduler [129]. Due to the inherently delayed feedback, these applications all use the TD-Learning or Q-Learning value function RL methods.

For many financial decision making problems, however, results accrue gradually over time, and one can immediately measure short-term performance. This enables use of a Direct Reinforcement approach to provide immediate feedback to optimize the strategy.

While this work emphasizes Direct Reinforcement, most applications in finance to date have been based upon dynamic programming type methods. Elton & Gruber [31] provide an early survey of dynamic programming applications in finance. The problems of optimum consumption and portfolio choice in continuous time have been formulated by Merton [57, 58, 59] from the standpoints of dynamic programming and stochastic control. The extensive body of work on intertemporal (multi-period) portfolio management and asset pricing is reviewed by Breeden [14]. Duffie [28, 29] describes stochastic control and dynamic programming methods in finance in depth. Dynamic programming provides the

[6]Baxter & Bartlett have independently proposed the term "Direct Reinforcement" for *policy gradient* algorithms in a Markov Decision Process framework. Moody et al. use the term in the same spirit, but perhaps more generally, to refer to any reinforcement learning algorithm that *does not* require learning a value function.

[7]For reviews and in-depth presentations of value function and actor-critic methods, see [46, 10, 104].

basis of the Cox, Ross, Rubinstein [22] and other widely used binomial option pricing methods. See also the strategic asset allocation work of Brennan et al. [16]. Due to the curse of dimensionality, approximate dynamic programming is often required to solve practical problems, as in the work by Longstaff and Schwartz [52] on pricing American options. During the past six years, there have been several applications that make use of value function reinforcement learning methods. Van Roy [115] uses a TD(λ) approach for valuing options and performing portfolio optimization. Neuneier [84] uses a Q-Learning approach to make asset allocation decisions, and Neuneier & Mihatsch [85] incorporate a notion of risk sensitivity into the construction of the Q-Function. Derivatives pricing applications have been studied by Tsitsiklis and Van Roy [112, 113]. Moody and Saffell compare Direct Reinforcement to Q-Learning for asset allocation [69], and explore the minimization of downside risk using Direct Reinforcement [70].

Several papers that relate to this work have appeared. Samuelson [94] stimulated interest in power law utility functions and their relation to the differential Sharpe ratio. White [121] suggested using a performance ratio based on the second lower partial moment.

Samuelson [94] uses logarithmic and power law utility functions to evaluate simple asset allocation or market timing strategies (which he calls "across-time diversification"). Samuelson's analysis shows that under the assumption of a random walk price process, the optimum behavior for a trader with a power law utility function is to hold constant proportions of risky and risk-free securities. That is, in the absence of superior forecasting ability, across-asset-class diversification will on a risk-adjusted basis outperform across-time diversification. In contrast, the RRL approach assumes that superior forecasting and trading strategies are not impossible, and that dynamic asset allocation strategies may in some cases achieve higher utility than simple fixed allocation methods.

Timmermann and Pesaran [111] use wealth and the Sharpe ratio as selection criteria (rather than optimization criteria) for trading systems. The set of traders considered are based on linear forecasting models that differ in the subsets of input variables included. The forecasting models are linear regressions with parameters estimated to minimize mean squared forecast error (MSFE). The wealth and the Sharpe ratio performance functions are not used for direct optimization of system parameters. The selection among forecasting

models is updated periodically. The authors are able to use their simulation results to document predictability in monthly U.S. stock returns. In related work, Satchell and Timmerman [95] provide arguments that MSFE is a bad indicator for potential trading profits. They prove a theorem that there is not necessarily any monotonic relationship between the size of the MSFE and the probability of correctly forecasting the sign of a variable.

In independent work, Bengio [8] points out that global optimization of trading systems consisting of separate forecasting and trading modules (such as that shown in Figure 3.1) provides better results than separately minimizing MSFE of the forecast module and subsequently maximizing profit of the trading module. Bengio optimizes portfolios and employs back-propagation through time to maximize final wealth. Kang et al [47] compare optimization of the Sharpe ratio and profits using a non-recursive supervised learning method applied to a simple asset allocation strategy. The allocation that is varied is the amount of capital invested in a fixed trading strategy. However, the authors do not directly optimize the trading system parameters (those that determine when to buy or sell) or take into account transaction costs. Neuneier [84] uses Q-Learning [118, 117] to train an asset allocation system to maximize profit. Transaction costs consisting of both fixed and proportional parts are included in the analysis. Simulation results are presented for an artificial exchange rate trading system and a system that switches between cash and a portfolio of German stocks that tracks the DAX. The description of the methods used is sketchy, and most relevant details of the empirical work are not disclosed. Finally, White [121] has done simulations that optimize the Sharpe ratio and other measures of risk-adjusted return based on downside risk (see Section 3.6).

3.4 Review of the Recurrent Reinforcement Learning Model

3.4.1 Trading Systems and Performance Criteria

Trading Systems

This section describes agents that trade fixed position sizes in a single security. The methods described here can be generalized to more sophisticated agents that trade varying

quantities of a security, allocate assets continuously or manage multiple asset portfolios. See Moody et al [75] and Section 3.5 for a discussion of multiple asset portfolios.

Here, the traders are assumed to take only long, neutral or short positions, $F_t \in \{1, 0, -1\}$, of constant magnitude. A *long* position is initiated by purchasing some quantity of a security, while a *short* position is established by selling the security.[8]

The price series being traded is denoted z_t. The position F_t is established or maintained at the end of each time interval t, and is re-assessed at the end of period $t+1$. A trade is thus possible at the end of each time period, although nonzero trading costs will discourage excessive trading. A trading system return R_t is realized at the end of the time interval $(t-1, t]$ and includes the profit or loss resulting from the position F_{t-1} held during that interval and any transaction cost incurred at time t due to a difference in the positions F_{t-1} and F_t.

In order to properly incorporate the effects of transactions costs, market impact and taxes in a trader's decision making, the trader must have internal state information and must therefore be recurrent. A single asset trading system that takes into account transactions costs and market impact has the following recurrent decision function:

$$\begin{aligned} F_t &= F(\theta_t; F_{t-1}, I_t) \quad \text{with} \\ I_t &= \{z_t, z_{t-1}, z_{t-2}, \cdots; y_t, y_{t-1}, y_{t-2}, \ldots\} \ , \end{aligned} \tag{3.1}$$

where θ_t denotes the (learned) system parameters at time t and I_t denotes the information set at time t, which includes present and past values of the price series z_t and an arbitrary number of other external variables denoted y_t. A simple example is a {long, short} trader with $m+1$ autoregressive inputs:

$$F_t = \text{sign}(\, uF_{t-1} + v_0 r_t + v_1 r_{t-1} + \ldots + v_m r_{t-m} + w \,) \ , \tag{3.2}$$

where r_t are the *price returns* of z_t (defined below) and the system parameters θ are the weights $\{u, v_i, w\}$. A trader of this form is used in the simulations described in Section 3.9.1.

[8] For stocks, a *short sale* involves borrowing shares and then selling the borrowed shares to a third party. A profit is made when the *shorted* shares are repurchased at a later time at a lower price. Short sales of many securities, including stocks, bonds, futures, options and foreign exchange contracts are common place.

The above formulation describes a discrete-action, deterministic trader, but can be easily generalized. One simple generalization is to use continuously valued $F()$, for example by replacing *sign* with *tanh*. When discrete values $F_t = \{1, 0, -1\}$ are imposed, however, the decision function is not differentiable. None-the-less, gradient based optimization methods for θ may be developed by considering differentiable pre-thresholded outputs or, for example, by replacing *sign* with *tanh* during learning and discretizing the outputs when trading.

Moreover, the models can be extended to a stochastic framework by including a noise variable in $F()$:

$$F_t = F(\theta_t; F_{t-1}, I_t; \epsilon_t) \quad \text{with} \quad \epsilon_t \sim p_\epsilon(\epsilon) \quad . \tag{3.3}$$

The random variable ϵ_t induces a joint probability density for the discrete actions F_t, model parameters and model inputs:

$$p(F_t; \theta_t; F_{t-1}, I_t) \quad . \tag{3.4}$$

The noise level (measured by σ_ϵ or more generally the scale of p_ϵ) can be varied to control the "exploration vs. exploitation" behavior of the trader. Also, differentiability of the probability distribution of actions enables the straightforward application of gradient based learning methods.

Profit and Wealth for Trading Systems

Trading systems can be optimized by maximizing performance functions, $U()$, such as profit, wealth, utility functions of wealth or performance ratios like the Sharpe ratio. The simplest and most natural performance function for a risk-insensitive trader is profit.

Additive profits are appropriate to consider if each trade is for a fixed number of shares or contracts of security z_t. This is often the case, for example, when trading small stock or futures accounts or when trading standard US$ FX contracts in dollar-denominated foreign currencies. I define $r_t = z_t - z_{t-1}$ and $r_t^f = z_t^f - z_{t-1}^f$ as the *price returns* of a risky (traded) asset and a risk-free asset (like T-Bills) respectively, and denote the transactions cost rate as δ. The additive profit accumulated over T time periods with trading position

size $\mu > 0$ is then defined in term of the *trading returns*, R_t, as:

$$\begin{aligned} P_T &= \sum_{t=1}^{T} R_t \quad \text{where} \\ R_t &\equiv \mu \left\{ r_t^f + F_{t-1}(r_t - r_t^f) - \delta |F_t - F_{t-1}| \right\} \end{aligned} \tag{3.5}$$

with $P_0 = 0$ and typically $F_T = F_0 = 0$. When the risk-free rate of interest is ignored ($r_t^f = 0$), a simplified expression is obtained:

$$R_t = \mu \left\{ F_{t-1} r_t - \delta |F_t - F_{t-1}| \right\} \ . \tag{3.6}$$

The wealth of the trader is defined as $W_T = W_0 + P_T$.

Multiplicative profits are appropriate when a fixed fraction of accumulated wealth $\nu > 0$ is invested in each long or short trade. Here, $r_t = (z_t / z_{t-1} - 1)$ and $r_t^f = (z_t^f / z_{t-1}^f - 1)$. If no short sales are allowed and the leverage factor is set fixed at $\nu = 1$, the wealth at time T is:

$$\begin{aligned} W_T &= W_0 \prod_{t=1}^{T} \{1 + R_t\} \quad \text{where} \\ \{1 + R_t\} &\equiv \left\{ 1 + (1 - F_{t-1}) r_t^f + F_{t-1} r_t \right\} \times \\ &\qquad \{1 - \delta |F_t - F_{t-1}|\} \ . \end{aligned} \tag{3.7}$$

When the risk-free rate of interest is ignored ($r_t^f = 0$), a second simplified expression is obtained:

$$\{1 + R_t\} = \{1 + F_{t-1} r_t\} \{1 - \delta |F_t - F_{t-1}|\} \ . \tag{3.8}$$

Relaxing the constant magnitude assumption is more realistic for asset allocations and portfolios, and enables better risk control. Related expressions for portfolios are presented in Section 3.5.

Performance Criteria

In general, the performance criteria that are considered are functions of profit or wealth $U(W_T)$ after a sequence of T time steps, or more generally of the whole time sequence of trades

$$U(W_T, \ldots, W_t, \ldots, W_1, W_0) \ . \tag{3.9}$$

The simple form $U(W_T)$ includes standard economic utility functions. The second case is the general form for path-dependent performance functions, which include inter-temporal utility functions and performance ratios like the Sharpe ratio and Sterling ratio. In either case, the performance criterion at time T can be expressed as a function of the sequence of trading returns

$$U(R_T, \ldots, R_t, \ldots, R_2, R_1; W_0) \quad . \tag{3.10}$$

For brevity, this general form is denoted by U_T.

For optimizing the traders, the RRL algorithm is interested in the marginal increase in performance due to return R_t at each time step:

$$D_t \propto \Delta U_t = U_t - U_{t-1} \ . \tag{3.11}$$

Note that U_t depends upon the current trading return R_t, but that U_{t-1} does not. The strategy is to derive *differential* performance criteria $D_t \propto \Delta U_t$ that capture the marginal "utility" of the trading return R_t at each period.[9]

The Differential Sharpe Ratio

Rather than maximizing profits, most modern fund managers attempt to maximize risk-adjusted return, as suggested by modern portfolio theory. The Sharpe ratio is the most widely-used measure of risk-adjusted return [96]. Denoting as before the trading system returns for period t (including transactions costs) as R_t, the Sharpe ratio is defined to be:

$$S_T = \frac{\text{Average}(R_t)}{\text{Standard Deviation}(R_t)} \tag{3.12}$$

where the average and standard deviation are estimated for periods $t = \{1, \ldots, T\}$. Note that for ease of exposition and analysis, Moody and Wu have suppressed inclusion of portfolio returns R_t^f due to the risk free rate on capital r_t^f. Substituting *excess* returns $\tilde{R}_t = R_t - R_t^f$ for R_t in the equation above produces the standard definition. With this caveat in mind, I use Equation (3.12) for discussion purposes without loss of mathematical generality.[10]

[9] Strictly speaking, many of the performance criteria commonly used in the financial industry are not true utility functions, so the term "utility" is used in a more colloquial sense.

[10] For systems that trade futures and forwards, R_t should be used in place of $\tilde{R}_t$, because the risk free rate is already accounted for in the relation between forwards prices and spot prices.

Proper on-line learning requires the computation of the influence on the Sharpe ratio (marginal utility D_t) of the trading return R_t at time t. To accomplish this, Moody & Wu have derived a new objective function called the *differential Sharpe ratio* for on-line optimization of trading system performance [74, 75]. It is obtained by considering exponential moving averages of the returns and standard deviation of returns in (3.12), and expanding to first order in the adaptation rate η:

$$\begin{aligned} S_t|_{\eta>0} &\approx S_t|_{\eta=0} + \eta \frac{dS_t}{d\eta}|_{\eta=0} + O(\eta^2) \\ &= S_{t-1} + \eta \frac{dS_t}{d\eta}|_{\eta=0} + O(\eta^2) \ . \end{aligned} \tag{3.13}$$

Note that a zero adaptation rate corresponds to an infinite time average. Expanding about $\eta = 0$ amounts to "turning on" the adaptation.

Since only the first order term in this expansion depends upon the return R_t at time t, the *differential Sharpe ratio* is defined as:

$$D_t \equiv \frac{dS_t}{d\eta} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{3/2}} \ . \tag{3.14}$$

where the quantities A_t and B_t are exponential moving estimates of the first and second moments of R_t:

$$\begin{aligned} A_t &= A_{t-1} + \eta \Delta A_t = A_{t-1} + \eta(R_t - A_{t-1}) \\ B_t &= B_{t-1} + \eta \Delta B_t = B_{t-1} + \eta(R_t^2 - B_{t-1}) \ . \end{aligned} \tag{3.15}$$

Treating A_{t-1} and B_{t-1} as numerical constants, note that η in the update equations controls the magnitude of the influence of the return R_t on the Sharpe ratio S_t. Hence, the *differential Sharpe ratio* represents the influence of the trading return R_t realized at time t on S_t. It is the marginal utility for the Sharpe ratio criterion.

The influences of risk and return on the differential Sharpe ratio are readily apparent. The current return R_t enters expression (3.14) only in the numerator through $\Delta A_t = R_t - A_{t-1}$ and $\Delta B_t = R_t^2 - B_{t-1}$. The first term in the numerator is positive if R_t exceeds the moving average of past returns A_{t-1} (increased reward), while the second term is negative if R_t^2 exceeds the moving average of past squared returns B_{t-1} (increased risk).

The differential Sharpe ratio D_t is used in the RRL algorithm (see Equation (3.37) in Section 3.7) as the current contribution to the performance function U_t. Since S_{t-1} in Equation (3.13) does not depend on R_t, then

$$\frac{dU_t}{dR_t} = \frac{dS_t}{dR_t} \approx \eta \frac{dD_t}{dR_t}. \tag{3.16}$$

When optimizing the trading system using Equation (3.14), the relevant derivatives have the simple form:

$$\frac{dD_t}{dR_t} = \frac{B_{t-1} - A_{t-1}R_t}{(B_{t-1} - A_{t-1}^2)^{3/2}} \quad . \tag{3.17}$$

The differential Sharpe ratio has several attractive properties:

- Facilitates recursive updating: The incremental nature of the calculations of A_t and B_t make updating the exponential moving Sharpe ratio straightforward. It is not necessary to recompute the average and standard deviation of returns for the entire trading history in order to update the Sharpe ratio for the most recent time period.

- Enables efficient on-line optimization: D_t and dD_t/dR_t can be cheaply calculated using the previously computed moving averages A_{t-1} and B_{t-1} and the current return R_t. This enables efficient stochastic optimization.

- Weights recent returns more: Based on the exponential moving average Sharpe ratio, recent returns receive stronger weightings in D_t than do older returns.

- Provides interpretability: The differential Sharpe ratio isolates the contribution of the current return R_t to the exponential moving average Sharpe ratio. The simple form of D_t makes clear how risk and reward affect the Sharpe ratio.

One difficulty with the Sharpe ratio, however, is that the use of variance or R_t^2 as a risk measure does not distinguish between upside and downside "risk". Assuming that $A_{t-1} > 0$, the largest possible improvement in D_t occurs when

$$R_t^* = B_{t-1}/A_{t-1} \quad . \tag{3.18}$$

Thus, the Sharpe ratio actually penalizes gains larger than R_t^*, which is counter-intuitive relative to most investors' notions of risk and reward. A more relevant performance function is described in Section 3.6.

3.5 Portfolios

As was mentioned previously, investors typically hold diversified portfolios of assets. One of the main purposes of holding portfolios is to control the impact that unique risks associated with individual assets have on the investor's wealth. Making trading decisions relative to the portfolio as a whole, rather than on an asset by asset basis, is crucial for managing these risks properly. Simulation results using the following setup are presented in Section 3.9.6.

For trading multiple assets in general (typically including a risk-free instrument), a multiple output trading system is required. Denoting a set of m markets with price series $\{\{z_t^a\} : a = 1, \ldots, m\}$, the market return r_t^a for price series z_t^a for the period ending at time t is defined as $(z_t^a / z_{t-1}^a - 1)$. Defining portfolio weights of the a^{th} asset as $F^a()$, a trader that takes only long positions must have portfolio weights that satisfy:

$$F^a \geq 0 \quad \text{and} \quad \sum_{a=1}^{m} F^a = 1 \ . \tag{3.19}$$

With these constraints, standard Markowitz mean-variance portfolio optimization is a quadratic programming problem. However, when optimizing the parameters of a nonlinear trading system, portfolio optimization becomes a nonlinear programming problem.

One approach to imposing the constraints on the portfolio weights (3.19) without requiring that a constrained optimization be performed is to use a trading system that has softmax outputs:

$$F^a() = \frac{\exp[f^a()]}{\sum_{b=1}^{m} \exp[f^b()]} \quad \text{for} \quad a = 1, \ldots, m \ . \tag{3.20}$$

Here, the $f^a()$ could be linear or more complex functions of the inputs, such as a two-layer neural network with sigmoidal internal units and linear outputs. Such a trading system can be optimized using unconstrained optimization methods. Note however that the portfolio weights F^a obtained are invariant under shifts in the values of the f^a of the form $\{f^a \rightarrow f^a + c;\ a = 1, \ldots, m\}$, so multiple solutions for the f^a exist. Denoting the sets of raw and normalized outputs collectively as the vectors $\mathbf{f}()$ and $\mathbf{F}()$ respectively, a recurrent portfolio manager will have structure

$$\mathbf{F}_t = \text{softmax}\left\{\mathbf{f}_t(\theta_{t-1}; \mathbf{F}_{t-1}, I_t)\right\} \ . \tag{3.21}$$

Similarly to the single asset trader, the portfolio management model must be recurrent to accurately account for transaction costs.

3.5.1 Profit and Wealth for Portfolios

When multiple assets are considered, the effective portfolio weightings change with each time step due to price movements. Thus, maintaining constant or desired portfolio weights requires that adjustments in positions be made at each time step. The wealth after T periods for a portfolio trading system is

$$\begin{aligned} W_T &= W_0 \prod_{t=1}^{T} \{1 + R_t\} \\ &= W_0 \prod_{t=1}^{T} \left\{ \left(\sum_{a=1}^{m} F_{t-1}^a \frac{z_t^a}{z_{t-1}^a} \right) \left(1 - \delta \sum_{a=1}^{m} |F_t^a - \tilde{F}_t^a| \right) \right\} \quad , \end{aligned} \tag{3.22}$$

where $\tilde{F}_t^a$ is the effective portfolio weight of asset a before readjusting, defined as

$$\tilde{F}_t^a = \frac{F_{t-1}^a (z_t^a / z_{t-1}^a)}{\sum_{b=1}^{m} F_{t-1}^b (z_t^b / z_{t-1}^b)} \quad , \tag{3.23}$$

and the trading returns R_t are defined implicitly. In (3.22), the first factor in the curly brackets is the increase in wealth over the time interval t prior to rebalancing to achieve the newly specified weights F_t^a. The second factor is the reduction in wealth due to the rebalancing costs. The profit after T periods is $P_T = W_T - W_0$.

3.6 Downside Risk

Symmetric measures of risk such as variance are more and more being viewed as inadequate measures due to the asymmetric preferences of most investors to price changes. Few investors consider large positive returns to be "risky", though both large positive as well as negative returns are penalized using a symmetric measure of risk such as the variance. To most investors, the term "risk" refers intuitively to returns in a portfolio that decrease its profitability.

Markowitz, the father of modern portfolio theory, understood this. Even though most of his work focused on the mean-variance framework for portfolio optimization, he proposed the semi-variance as a means for dealing with downside returns [55]. After a long

hiatus lasting three decades, there is now a vigorous industry in the financial community in modeling and minimizing downside risk. Criteria of interest include the Downside Deviation (DD), the Second Lower Partial Moment (SLPM) and the N^{th} Lower Partial Moment [99, 80, 81, 98, 82].

One measure of risk-adjusted performance widely used in the professional fund management community (especially for hedge funds) is the Sterling ratio, commonly defined as:

$$\text{Sterling Ratio} = \frac{\text{Annualized Average Return}}{\text{Maximum Drawn-Down}} \ . \tag{3.24}$$

Here, the maximum draw-down (from peak to trough) in account equity or net asset value is defined relative to some standard reference period, for example one to three years. Minimizing drawdowns is somewhat cumbersome, so I focus on the Downside Deviation as a measure of downside risk in this chapter.[11]

The Downside Deviation is defined to be the square root of the average of the square of the negative returns:

$$\text{DD}_T = \left(\frac{1}{T} \sum_{t=1}^{T} \min\{R_t, 0\}^2 \right)^{\frac{1}{2}} \ . \tag{3.25}$$

Using the Downside Deviation as a measure of risk, I can now define a utility function similar to the Sharpe ratio, which will be called the *Downside Deviation Ratio* (DDR):

$$\text{DDR}_T = \frac{\text{Average}(R_t)}{\text{DD}_T} \ . \tag{3.26}$$

The Downside Deviation Ratio rewards the presence of large average positive returns and penalizes risky returns, where "risky" now refers to downside returns.

In order to facilitate the use of the recurrent reinforcement learning algorithm (Section 3.7), the influence of the return at time t on the DDR must be computed. In a similar manner to the development of the differential Sharpe ratio in Moody et al. [75], I define exponential moving averages of returns and of the squared Downside Deviation:

$$\begin{aligned} A_t &= A_{t-1} + \eta(R_t - A_{t-1}) \\ \text{DD}_t^2 &= \text{DD}_{t-1}^2 + \eta(\min\{R_t, 0\}^2 - \text{DD}_{t-1}^2) \ , \end{aligned} \tag{3.27}$$

[11] White has found that the Downside Deviation Ratio tracks the Sterling Ratio effectively [121].

and define the Downside Deviation Ratio in terms of these moving averages. I obtain the performance function by considering a first order expansion in the adaptation rate η of the DDR:

$$\mathrm{DDR}_t \approx \mathrm{DDR}_{t-1} + \eta \frac{d\mathrm{DDR}_t}{d\eta}|_{\eta=0} + O(\eta^2) \; . \tag{3.28}$$

I define the first order term $d\mathrm{DDR}_t/d\eta$ to be the *Differential Downside Deviation Ratio.* It has the form

$$\begin{aligned} D_t &\equiv \frac{d\mathrm{DDR}_t}{d\eta} \\ &= \frac{R_t - \frac{1}{2}A_{t-1}}{\mathrm{DD}_{t-1}} \qquad , R_t > 0 \qquad (3.29) \\ &= \frac{\mathrm{DD}_{t-1}^2 \cdot (R_t - \frac{1}{2}A_{t-1}) - \frac{1}{2}A_{t-1}R_t^2}{\mathrm{DD}_{t-1}^3} \; , R_t \leq 0 \; . \qquad (3.30) \end{aligned}$$

From Equation (3.30) it is obvious that when $R_t > 0$, the utility increases as R_t increases, with no penalty for large positive returns such as exists when using variance as the risk measure. See Section 3.9 for detailed experimental results on the use of the Downside Deviation Ratio to build RRL trading systems.

3.7 Learning to Trade

Reinforcement learning adjusts the parameters of a system to maximize the expected payoff or reward that is generated due to the actions of the system. This is accomplished through *trial-and-error* exploration of the environment and space of strategies. In contrast to supervised learning, the system is not presented with examples of desired actions. Rather, it receives a reinforcement signal from its environment (a *reward*) that provides information on whether its actions are good or bad.

Moody et al. [74, 75] compared supervised learning to this Direct Reinforcement approach. The supervised methods discussed included trading based upon forecasts of market prices and training a trader using labelled data. In both supervised frameworks, difficulties are encountered when transaction costs are included. While supervised learning methods can be effective for solving the structural credit assignment problem, they do not typically address the temporal credit assignment problem.

Structural credit assignment refers to the problem of assigning credit to the individual parameters of a system. If the reward produced also depends on a series of actions of the system, then the *temporal credit assignment* problem is encountered, ie. assigning credit to the individual actions taken over time [102]. Reinforcement learning algorithms offer advantages over supervised methods by attempting to solve both problems simultaneously.

Reinforcement learning algorithms can be classified as either *Direct Reinforcement* (sometimes called "policy gradient" or "policy search"), *Value Function* or *Actor-Critic* methods. The choice of the best method depends upon the nature of the problem domain. I will discuss this issue in greater detail in Section 3.8. In this section, I present the Recurrent Reinforcement Learning algorithm for Direct Reinforcement and review value function based methods, specifically Q-Learning [118] and a refinement of Q-Learning called Advantage Updating [3]. In Section 3.9.4, I compare the RRL and value function methods for systems that learn to allocate assets between the S&P 500 stock index and T-Bills.

3.7.1 Recurrent Reinforcement Learning

In this section, I describe the Recurrent Reinforcement Learning algorithm for Direct Reinforcement [74, 75].

Given a trading system model $F_t(\theta)$, the goal is to adjust the parameters θ in order to maximize U_T. For traders of form (3.1) and trading returns of form (3.6) or (3.8), the gradient of U_T with respect to the parameters θ of the system after a sequence of T periods is

$$\frac{dU_T(\theta)}{d\theta} = \sum_{t=1}^{T} \frac{dU_T}{dR_t} \left\{ \frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta} \right\} \tag{3.31}$$

The system can be optimized in batch mode by repeatedly computing the value of U_T on forward passes through the data and adjusting the trading system parameters by using gradient ascent (with learning rate ρ)

$$\Delta\theta = \rho \frac{dU_T(\theta)}{d\theta} \tag{3.32}$$

or some other optimization method. Note that due to the inherent recurrence, the quantities $dF_t/d\theta$ are *total derivatives* that depend upon the entire sequence of previous time

periods. To correctly compute and optimize these total derivatives in an efficient manner requires an approach similar to Back-Propagation Through Time (BPTT) [91, 119]. The temporal dependencies in a sequence of decisions are accounted for through a recursive update equation for the *total* policy gradient:

$$\frac{dF_t}{d\theta} = \frac{\partial F_t}{\partial \theta} + \frac{\partial F_t}{\partial F_{t-1}} \frac{dF_{t-1}}{d\theta} \quad . \tag{3.33}$$

The first term on the right hand side is the *partial* or *naive* policy gradient. The above expressions (3.31) and (3.33) assume differentiability of F_t. For the long/short traders with thresholds described in Section 3.4.1, the reinforcement signal can be backpropagated through the pre-thresholded outputs in a manner similar to the Adaline learning rule [122]. Equations (3.31), (3.32), and (3.33) constitute the batch RRL algorithm.

There are two ways in which the batch algorithm described above can be extended into a stochastic framework. First, exploration of the strategy space can be induced by incorporating a noise variable ϵ_t, as in the stochastic trader formulation of Equation (3.3). The trade-off between *exploration* of the strategy space and *exploitation* of a learned policy can be controlled by the magnitude of the noise variance σ_ϵ. The noise magnitude can be annealed over time during simulation, in order to arrive at a good strategy.

Secondly, a simple on-line stochastic optimization can be obtained by considering only the term in (3.31) that depends on the most recently realized return R_t during a forward pass through the data:

$$\frac{dU_t(\theta)}{d\theta_t} \approx \frac{dU_t}{dR_t} \left\{ \frac{dR_t}{dF_t} \frac{dF_t}{d\theta_t} + \frac{dR_t}{dF_{t-1}} \frac{dF_{t-1}}{d\theta_{t-1}} \right\} \quad . \tag{3.34}$$

The parameters are then updated on-line using:

$$\Delta\theta_t = \rho \frac{dU_t(\theta_t)}{d\theta_t} \quad . \tag{3.35}$$

Such an algorithm performs a stochastic optimization, since the system parameters θ_t are varied *during* each forward pass through the training data. The stochastic, on-line analog of Equation (3.33) is:

$$\frac{dF_t}{d\theta_t} \approx \frac{\partial F_t}{\partial \theta_t} + \frac{\partial F_t}{\partial F_{t-1}} \frac{dF_{t-1}}{d\theta_{t-1}} \quad . \tag{3.36}$$

Equations (3.34), (3.35) and (3.36) constitute the stochastic (or adaptive) RRL algorithm. It is a reinforcement algorithm closely related to recurrent supervised algorithms such as Real Time Recurrent Learning (RTRL) [125] and Dynamic Backpropagation [78]. See also the discussion of backpropagating utility in Werbos [120].

For differential performance criteria D_t described in Equation (3.11) of Section 3.4.1 (such as the differential Sharpe ratio (3.14) and differential Downside Deviation ratio (3.30)), the stochastic update equations (3.34) and (3.35) become:

$$\begin{aligned} \Delta\theta_t &= \rho \frac{dD_t(\theta_t)}{d\theta_t} \\ &\approx \rho \frac{dD_t}{dR_t} \left\{ \frac{dR_t}{dF_t}\frac{dF_t}{d\theta_t} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta_{t-1}} \right\} . \end{aligned} \tag{3.37}$$

I use on-line algorithms of this recurrent reinforcement learning type in the simulations presented in Section 3.9. Note that I find that use of a noise variable ϵ_t provides little advantage for the real financial applications that I consider, since the data series contain significant intrinsic noise. Hence, I find that a simple "greedy" update is adequate.[12]

The above description of the RRL algorithm is for traders that optimize immediate estimates of performance D_t for specific actions taken. This presentation can be thought of as a special case of a more general Markov Decision Process (MDP) and policy gradient formulation. One straightforward extension of the formulation can be obtained for traders that maximize discounted future rewards. I have experimented with this approach, but found little advantage for the problems I consider. A second extension to the formulation is to consider a stochastic trader (Equation (3.3)) and an *expected* reward framework, for which the probability distribution of actions is differentiable. This latter approach makes use of the joint density of Equation (3.4). While the expected reward framework is appealing from a theoretical perspective, Equations (3.34), (3.35) and (3.36) presented above provide the practical basis for simulations.

[12] Tesauro finds a similar result for TD-Gammon [109, 110]. A "greedy" update works well, because the dice rolls in the game provided enough uncertainty to induce extensive strategy exploration.

3.7.2 Value Functions and Q-Learning

Besides explicitly training a trader to take actions, it is possible to also implicitly learn correct actions through the technique of *value iteration.* Value iteration uses a *value function* to evaluate and improve policies (see Kaelbling et al. [46] for a tutorial introduction and Sutton and Barto [104] for a full overview of these algorithms). The value function, $V^{\pi}(x)$, is an estimate of discounted future rewards that will be received from starting in state x, and by following the policy π thereafter. The value function satisfies *Bellman's equation*

$$V^{\pi}(x) = \sum_{a} \pi(x,a) \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{\pi}(y) \right\} \ , \tag{3.38}$$

where $\pi(x,a)$ is the probability of taking action a in state x, $p_{xy}(a)$ is the probability of transitioning from state x to state y when taking action a, $D(x,y,a)$ is the immediate reward (differential utility, as in Equation (3.11)) from taking action a and transitioning from state x to state y and γ is the discount factor that weighs the importance of future rewards versus immediate rewards.

A policy is an *optimal* policy if its value function is greater than or equal to the value functions of all other policies for a given set of states. The optimal value function is defined as:

$$V^{*}(x) = \max_{\pi} V^{\pi}(x) \ , \tag{3.39}$$

and satisfies Bellman's optimality equation

$$V^{*}(x) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{*}(y) \right\} \ . \tag{3.40}$$

The value iteration update:

$$V_{t+1}(x) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V_{t}(y) \right\} \ , \tag{3.41}$$

is guaranteed to converge to the optimal value function under certain general conditions. The optimal policy can be determined from the optimal value function through:

$$a^{*} = \arg\max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{*}(y) \right\} \ . \tag{3.42}$$

Q-Learning

The technique named Q-Learning [118] uses a value function which estimates future rewards based on both the current state and the current action taken. The Q-function version of Bellman's optimality equation is

$$Q^*(x,a) = \sum_y p_{xy}(a) \left\{ D(x,y,a) + \gamma \max_b Q^*(y,b) \right\} \ . \tag{3.43}$$

Similarly to Equation (3.41), the Q-function can be learned using a value iteration approach:

$$Q_{t+1}(x,a) = \max_a \sum_y p_{xy}(a) \left\{ D(x,y,a) + \gamma Q_t(y) \right\} \ . \tag{3.44}$$

This iteration has been shown [118] to converge to the optimal Q-function, $Q^*(x,a)$, given certain constraints. The advantage of using the Q-function is that there is no need to know the system model $p_{xy}(a)$ as in Equation (3.42) in order to choose the best action. One calculates the best action as

$$a^* = \arg\max_a (Q^*(x,a)) \ . \tag{3.45}$$

The update rule for training a function approximator is then based on the gradient of the error:

$$\frac{1}{2}(D(x,y,a) + \gamma \max_b Q(y,b) - Q(x,a))^2 \ . \tag{3.46}$$

Advantage Updating

A refinement of the Q-Learning algorithm is provided by Advantage Updating [3]. Advantage Updating was developed specifically to deal with continuous-time reinforcement learning problems, though it is applicable to the discrete-time case as well. It is designed to deal with the situation where the relative advantages of individual actions within a state are small compared to the relative advantages of being in different states. Also, Advantage Updating has been shown to be able to learn at a much faster rate than Q-Learning in the presence of noise.

Advantage Updating learns two separate functions: the advantage function $A(x,a)$, and the value function $V(x)$. The advantage function measures the relative change in

value of choosing action a while in state x versus choosing the best possible action for that state. The value function measures the expected discounted future rewards as described previously. Advantage Updating has the following relationship with Q-Learning:

$$Q^*(x,a) = V^*(x) + A^*(x,a) \quad . \tag{3.47}$$

Similarly to Q-Learning, the optimal action to take in state x is found by $a^* = \arg\max_a(A^*(x,a))$. See Baird [3] for a description of the learning algorithms.

3.8 Policy vs Value Functions

As mentioned in Section 3.7, reinforcement learning algorithms can be classified as either *Direct Reinforcement* (sometimes called "policy search"), *Value Function* methods or *Actor-Critic* methods. The choice of the best method depends upon the nature of the problem domain.

3.8.1 Immediate vs. Future Rewards

Reinforcement signals received from the environment can be *immediate* or *delayed.* In some problems, such as checkers [92, 93], backgammon [109, 110], navigating a maze [86], or maneuvering around obstacles [76], reinforcement from the environment is sometimes provided only at the end of the game or task. The final rewards received are {success, failure} or {win, lose}. For such task formulations, the temporal credit assignment problem is extreme. There is usually no *a priori* assessment of performance available during the course of each game or trial. Hence, one is forced to learn a *value function* of the system state at each time. This is accomplished by doing many runs on a trial and error basis, and discounting the ultimate reward received back in time. This discounting approach is the basis of Dynamic Programming [7], TD-Learning [103] and Q-Learning [118, 117].

For these Value Function methods, the action taken at each time is that which offers the largest increase in expected value. Thus, the policy is not represented directly. An intermediate class of reinforcement algorithms are *actor-critic* methods [5]. While the *actor* module provides a direct representation of the policy for these methods, it relies on the *critic* module for feedback. The role of the critic is to learn the value function.

In contrast, Direct Reinforcement methods represent the policy directly, and make use of immediate feedback to adjust the policy. This approach is appealing when it is possible to specify an instantaneous measure of performance, because the need to learn a value function is bypassed. For most problems of real world interest, it is possible to obtain feedback periodically. For example, intermediate game scores could be used in backgammon.

In trading, asset allocation and portfolio management problems, for example, overall performance accrues gradually over time. For these financial decision making problems, an immediate measure of incremental performance is available at each time step. Although total performance usually involves integrating or averaging over time, it is none-the-less possible to adaptively update the strategy based upon the investment return received at each time step.

Other domains that offer the possibility of immediate feedback include a wide range of control applications. The standard formulation for optimal control problems involves time integrals of an instantaneous performance measure. Examples of common loss functions include average squared deviation from a desired trajectory or average squared jerk.[13]

A related approach that represents and improves policies explicitly is the *policy gradient* approach. Policy gradient methods use the gradient of the expected average or discounted reward with respect to the parameters of the policy function to improve the policy. The expected rewards are typically estimated by learning a value function, or by using single sample paths of the Markov reward process. There have been several recent, independent proofs for the convergence of policy gradient methods. Marbach & Tsitsiklis [53, 54] and Baxter & Bartlett [6][14] show convergence to locally optimal policies by using simulation based methodologies to approximate expected rewards. Sutton et al. [105] and Konda & Tsitsiklis [49] obtain similar results when estimating expected rewards from a value function implemented using a function approximator. An application to robot navigation

[13] "Jerk" is the rate of change of acceleration.

[14] Baxter & Bartlett have independently coined the term "Direct Reinforcement" to describe policy gradient methods in an MDP framework based on simulating sample paths and maximizing average rewards. The usage of the term here is in the same spirit, but perhaps more general, referring to all algorithms that do not need to learn a value function in order to derive a policy.

is provided by Grudic and Ungar [39]. Note that some of the so-called "policy gradient" methods are not Direct Reinforcement methods, because they require the estimation of a value function. Rather, these methods are more properly classified as actor-critic methods.

3.8.2 Policies vs. Values

Much attention in the reinforcement learning community has been given recently to the question of learning policies versus learning value functions. Over the past twenty years or so, the Value Function approach has dominated the field. The approach has worked well in many applications, and a number of convergence theorems exist that prove that the approach will work under certain conditions.

However, the value function approach suffers from several limitations. The original formulation of Q-Learning is in the context of discrete state and action spaces. As such, in many practical situations it suffers from the "curse of dimensionality". When Q-Learning is extended to function approximators, it has been shown in many cases that there are simple Markov Decision Processes for which the algorithms fail to converge [2]. Also, the policies derived from a Q-Learning approach tend to be brittle, that is, small changes in the value function can produce large changes in the policy. For finance in particular, the presence of large amounts noise and nonstationarity in the datasets can cause severe problems for a value function approach.[15]

I find the Recurrent Reinforcement Learning algorithm to be a simpler and more efficient approach. Since the policy is represented directly, a much simpler functional form is often adequate to solve the problem. A significant advantage of the RRL approach is the ability to produce real valued actions (eg. portfolio weights) naturally without resorting to the discretization necessary in the Q-Learning case. Constraints on actions are also much easier to represent given the policy representation. Other advantages are that the RRL algorithm is more robust to the large amounts of noise that exists in financial data, and is able to quickly adapt to nonstationary market conditions.

[15] Brown [18] provides a nice example that demonstrates the brittleness of Q-Learners in noisy environments.

3.8.3 An Example

I present an example of how an increase in complexity occurs when a policy is represented implicitly through the use of a value function. I start with the most simple trading problem: a trader that makes decisions to buy and sell a single asset where there are no transaction costs or trading frictions. The asset returns r_t are from a binomial process in {-1, +1}. To make matters even more simple, I will assume that the next period's return r_{t+1} is known in advance. Given these conditions, the optimal policy does not require knowledge of future rewards, so the Q-Learning discount parameter γ will be set to 0. I will measure the complexity of the solution by counting the number of *tanh* units that are required to implement a solution using a single function approximator.

It is obvious that the policy function is trivial. The optimal policy is to take the action $a_t = r_{t+1}$. In terms of model structure, a single *tanh* unit would suffice. On the other hand, if learning the value function before taking actions, in this case the value function has the form of the XOR function. As shown in Figure 3.4, the value function is +1 when the proposed action a has the same sign as r_{t+1} and -1 otherwise. Because of the binomial return process, this problem can be solved using only two *tanh* units. Due to the value function representation of the problem, the complexity of the solution has doubled.

This doubling of model complexity is by comparison minor if the problem is made a little more realistic by allowing returns to be drawn from a continuous real-valued distribution. The complexity of the policy function has not increased, $a_t = \text{sign}(r_{t+1})$. However the value function's increase in complexity is potentially enormous. Since returns are now real valued, approximating the value function to an arbitrarily small precision requires an arbitrarily large model.

3.9 Results

This section presents empirical results for several simulations based on the various techniques discussed in this chapter related to the RRL algorithm. First, controlled experiments using artificial price series are presented to test the RRL algorithm's ability to learn profitable trading strategies, to maximize risk adjusted return (as measured by the Sharpe

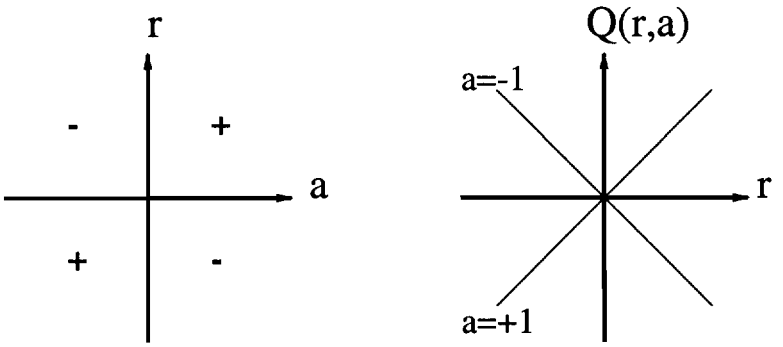

Figure 3.4: A representation of the value function to be learned by the Q-Learning algorithm for the example given in the text (Section 3.8). The function represents the Q-value, $Q(r, a)$, which is the value from taking action "a" in state "r". The figure on the left shows the value function for the case of discrete, binary returns. The Q-function has the form of the XOR problem, while the optimal policy is simply $a = r$. The figure on the right shows the value function when returns are real-valued (note the change in axes). The Q-function now becomes arbitrarily hard to represent accurately using a single function approximator of *tanh* units while the optimal policy is still very simple, $a = \text{sign}(r)$.

ratio), and to respond appropriately to varying transaction costs. The second problem demonstrates the ability of RRL to discover structure in a real financial price series, the half-hourly US Dollar / British Pound exchange rate. For this problem, the RRL trader attempts to avoid downside risk by maximizing the Downside Deviation Ratio. Next, I compare the performance of traders based on RRL and Q-Learning for a second real-world problem, trading the monthly S&P 500 stock index. Over the 25 year test period, I find that the RRL-Trader outperforms the Q-Trader, and that both outperform a buy and hold strategy. Further discussion of the Q-Trader vs. RRL-Trader performance is presented in Section 3.9.5. Finally, I present simulation results for a portfolio management system with three artificial assets. The portfolio management system is shown to adjust its behavior in response to changes in the environment and in response to the differing characteristics of the assets being managed.

3.9.1 Trader Simulation

In this section I demonstrate the use of the RRL algorithm to optimize trading behavior using the differential Sharpe Ratio (Equation (3.14)) in the presence of transaction costs. More extensive results are presented in Moody et al. [75]. There, the authors find that maximizing the differential Sharpe ratio yields more consistent results than maximizing profits, and that both methods outperform trading systems based on forecasts.

The RRL-Traders studied here take {long, short} positions and have recurrent state similar to that described in Section 3.4.1. To enable controlled experiments, the data used in this section are artificial price series that are designed to have tradeable structure. These experiments demonstrate that (a) RRL is an effective means of learning trading strategies, and (b) trading frequency is reduced as expected as transaction costs increase.

Data

Following the procedure in Moody & Wu [74], I generate log price series as random walks with autoregressive trend processes. The two parameter model is thus:

$$p(t) = p(t-1) + \beta(t-1) + k\,\epsilon(t) \tag{3.48}$$

$$\beta(t) = \alpha\beta(t-1) + \nu(t)\ , \tag{3.49}$$

where α and k are constants, and $\epsilon(t)$ and $\nu(t)$ are normal random deviates with zero mean and unit variance. The artificial price series is defined as

$$z(t) = \exp\left(\frac{p(t)}{R}\right) \tag{3.50}$$

where R is a scale defined as the range of $p(t)$: $\max(p(t)) - \min(p(t))$ over a simulation with 10,000 samples.[16]

For the results presented here, I set the parameters of the price process to $\alpha = 0.9$ and $k = 3$. The artificial price series are trending on short time scales and have a high level of noise. A realization of the artificial price series is shown in the top panel of Figure 3.5.

[16]This is slightly more than the number of hours in a year (8760), so the series could be thought of as representing hourly prices in a 24 hour artificial market. Alternatively, a series of this length could represent slightly less than five years of hourly data in a market that trades about 40 hours per week.

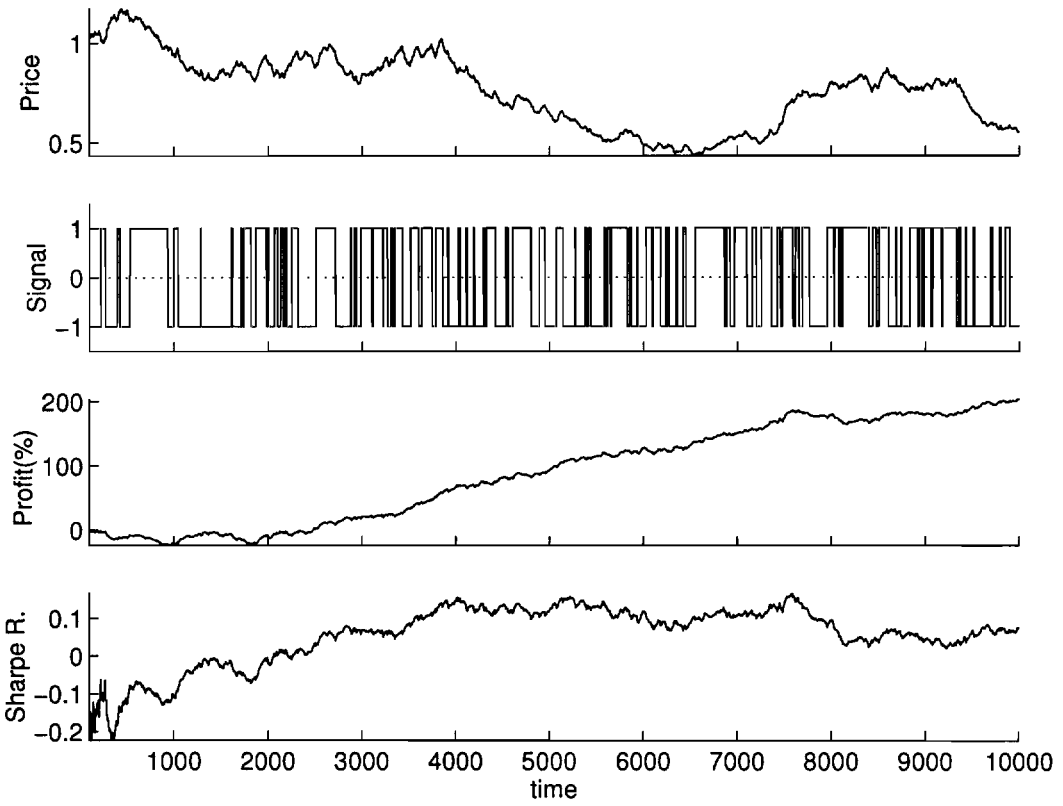

Figure 3.5: Artificial prices (top panel), trading signals (second panel), cumulative sums of profits (third panel) and the moving average Sharpe ratio with $\eta = 0.01$ (bottom panel). The system performs poorly while learning from scratch during the first 2000 time periods, but its performance remains good thereafter.

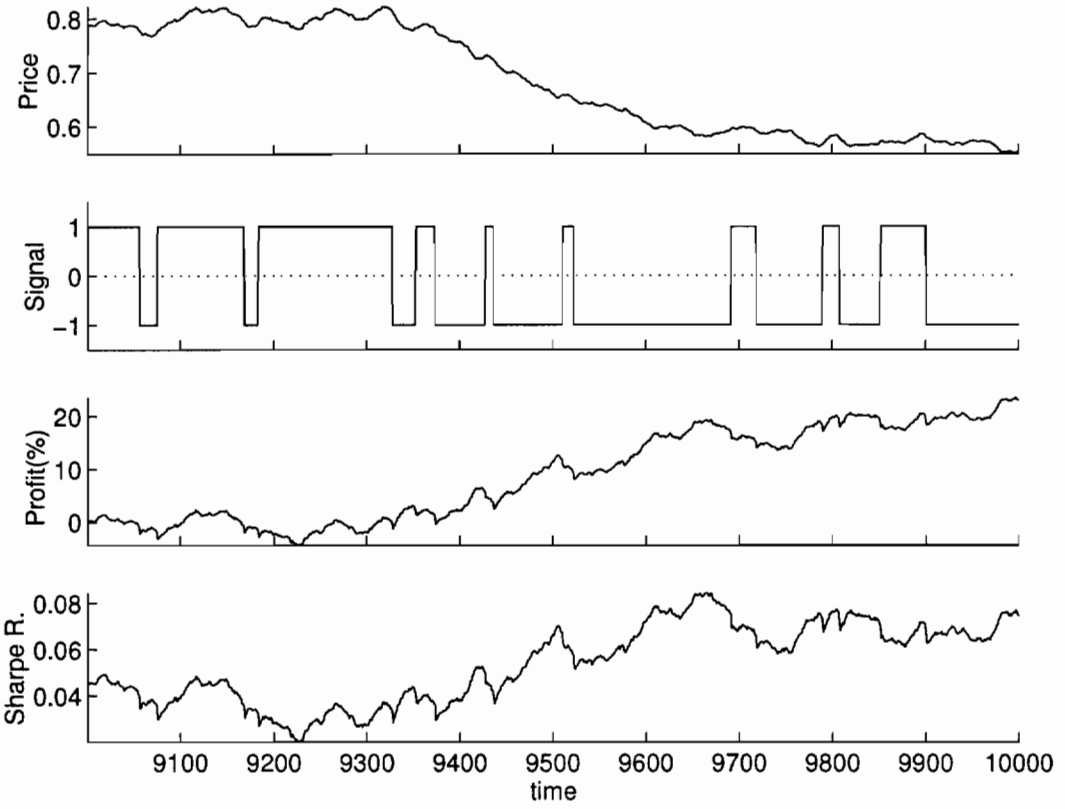

Figure 3.6: An expanded view of the last thousand time periods of Figure 3.5. The exponential moving Sharpe ratio has a forgetting time scale of $1/\eta = 100$ periods. A smaller η would smooth the fluctuations out.

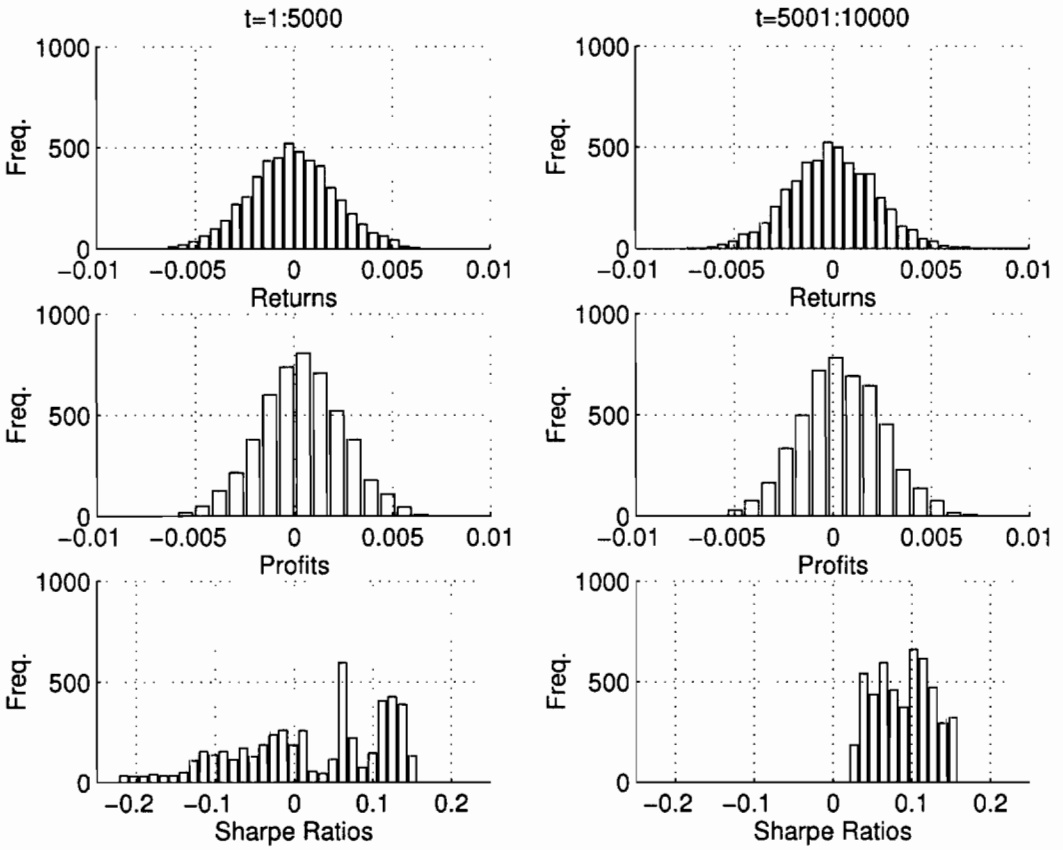

Figure 3.7: Histograms of the price changes (top), trading profits per time period (middle) and Sharpe ratios (bottom) for the simulation shown in Figure 3.5. The left column is for the first 5,000 time periods, and the right column is for the last 5,000 time periods. The transient effects during the first 2000 time periods for the real-time recurrent learning are evident in the lower left graph.

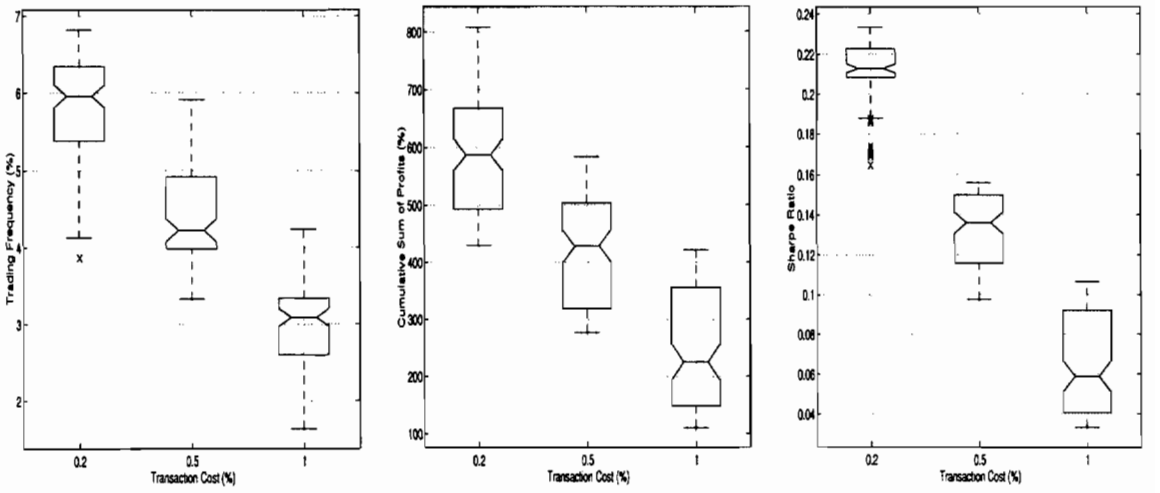

Figure 3.8: Boxplots of trading frequency, cumulative sums of profits and Sharpe ratios vs. transaction costs. The results are obtained over 100 trials with various realizations of artificial data and initial system parameters. Increased transaction costs reduce trading frequency, profits and Sharpe ratio, as expected. The trading frequency is the percentage of the number of time periods during which trades occur. All figures are computed on the last 9,000 points in the data set.

Simulated Trading Results

Figures 3.5, 3.6 and 3.7 show results for a single simulation for an artificial market as described above. For these experiments, the RRL-Traders are single threshold units with an autoregressive input representation. The inputs at time t are constructed using the previous eight returns.

The RRL-Traders are initialized randomly at the beginning, and adapted using real-time recurrent learning to optimize the differential Sharpe ratio (3.14). The transaction costs are fixed at a half percent during the whole real-time learning and trading process. Transient effects of the initial learning while trading process can be seen in the first 2000 time steps of Figure 3.5 and in the distribution of differential Sharpe ratios in the lower left panel of Figure 3.7.

Figure 3.8 shows box plots summarizing test performances for ensembles of 100 experiments.

In these simulations, the 10,000 data samples are partitioned into an initial training

set consisting of the first 1,000 samples and a subsequent test data set containing the last 9,000 samples. The RRL-Traders are first optimized on the training data set for 100 epochs and adapted on-line throughout the whole test data set. Each trial has different realizations of the artificial price process and different randomly-chosen initial trader parameter values. I vary the transaction cost from 0.2%, 0.5% to 1%, and observe the trading frequency, cumulative profit and Sharpe ratio over the test data set. As shown, in all 100 experiments, positive Sharpe ratios are obtained. As expected, trading frequency is reduced as transaction costs increase.

3.9.2 Simulations for Maximizing the Downside Deviation Ratio

Artificial Price Series with Skewed Returns

I generate price series as random walks with autoregressive trend processes with *skewed* returns distributions. The two parameter model for an individual asset is thus:

$$\begin{aligned} p(t) &= p(t-1) + \beta(t-1) + k\,\epsilon(t) \\ \beta(t) &= \alpha\,\beta(t-1) + \nu(t) \ , \end{aligned} \tag{3.51}$$

where α and k are constants, and $\epsilon(t)$ and $\nu(t)$ are iid random deviates drawn from a demeaned gamma distribution with shape parameter 5. The artificial price series is defined as

$$z(t) = \exp\left(\frac{p(t)}{R}\right) \tag{3.52}$$

where R is a scale defined as the range of $p(t)$: $\max(p(t)) - \min(p(t))$ over a simulation with 10,000 samples. Figure 3.9 shows an example of an artificially created time series, and a histogram of the returns distribution that generated it.

Long/Neutral/Short Trading System

In this section I compare trading systems that can take long, neutral and short positions in a single asset. The dataset used consists of 10 different realizations of artificially created data described in Section 3.9.2. One trading system is trained to maximize the Downside Deviation ratio, "DDR", and the other is trained to maximize the Sharpe ratio, "SR". The

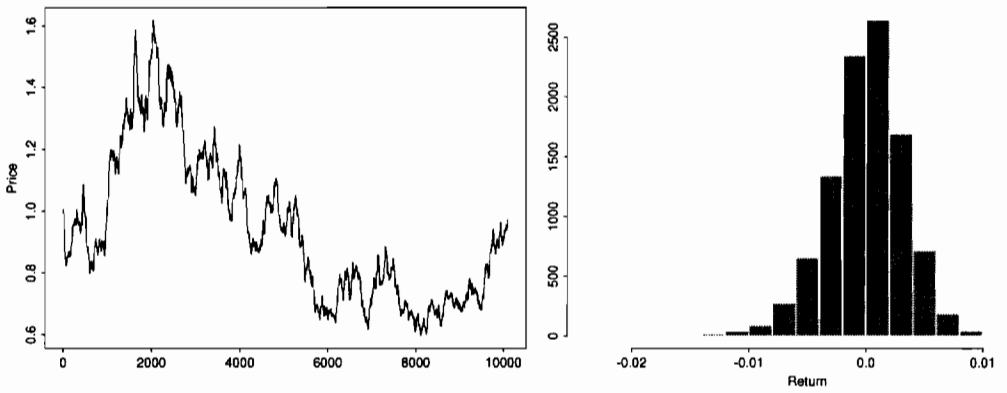

Figure 3.9: An example of the type of artificial price series used in the experiments here and the negatively skewed returns distribution that generated it. The returns are generated from a gamma distribution with shape parameter 5. The means of the distributions are removed, and here the sign of the distribution is adjusted to produce a negatively skewed distribution.

left panel in Figure 3.10 shows boxplots of the Sharpe ratio for 100 trading systems trained from random initial weight configurations. The right panel shows the summary results when measuring the Downside Deviation ratio. The plots show that the trading systems are successful in maximizing the performance ratios they were trained to maximize.

A very interesting property of the systems here that were trained to maximize the Downside Deviation ratio is that the maximum drawdowns incurred by these traders are consistently less than the drawdowns incurred by the Sharpe ratio traders. Figures 3.11 and 3.12 show a comparison of drawdowns between the "DDR" and "SR" trading systems. Figure 3.11 compares the histograms of maximum drawdowns for each trading system. This figure shows that the worst drawdowns in the histogram for the "DDR" trading system are truncated at approximately half the magnitude as that of the "SR" trading system. Figure 3.12 shows an excerpt of the behavior of the two systems on a sample test period. The top panel shows the underwater curves for the trading systems. An underwater curve shows the magnitude of the drawdowns, and is equal to 0 when the system is "above water", ie. when a new equity peak is achieved. The difference in behavior during drawdowns can be clearly seen here as the "DDR" trader maintains a

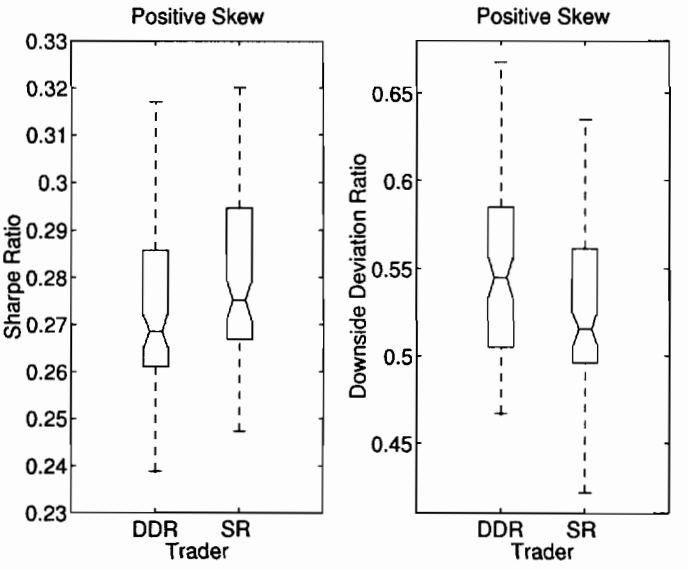

Figure 3.10: Boxplots of performance results on the artificial data series for ensembles of 100 trading systems. The set of systems labeled "DDR" were trained to maximize the Downside Deviation ratio, and the set labeled "SR" were trained to maximize the Sharpe ratio. The left panel shows summary statistics of the final Sharpe ratios and the right panel shows the summary statistics of final Downside Deviation ratios calculated for each of the 100 trials. The notches shown on the boxplots are robust confidence intervals on the medians of the distributions. The plots show that the systems trained to maximize a certain performance function do, on average, significantly outperform (according to their performance measure) trading systems trained to maximize other performance functions. The 100 trials were composed of trading system trained from 10 different random initializations for each of 10 different artificially generated price series.

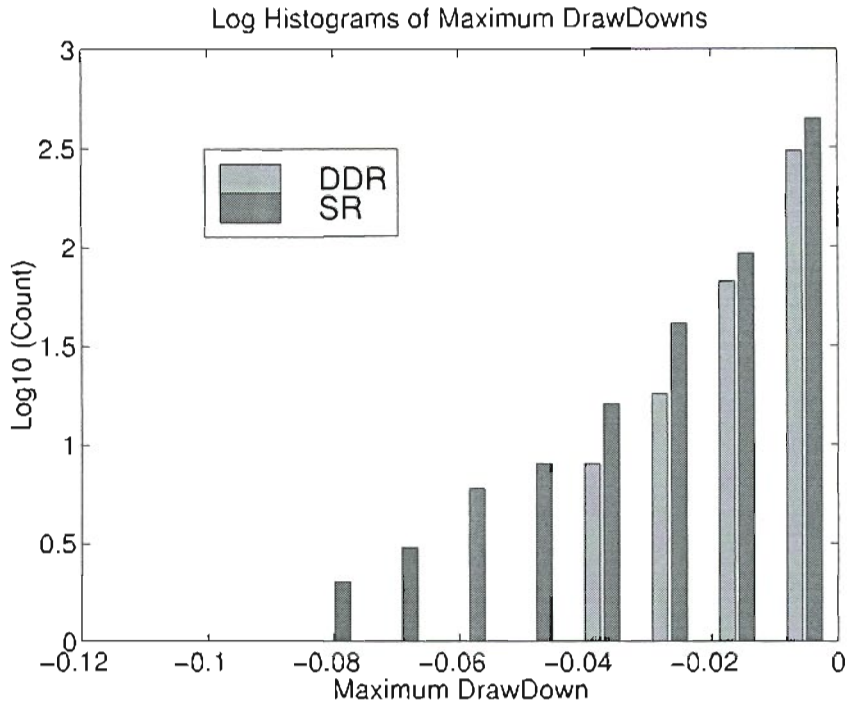

Figure 3.11: A comparison of the histograms of maximum drawdown incurred by the "DDR" and "SR" trading systems. The magnitudes of the worst drawdowns for the "DDR" system are only half the size of those for the "SR" trading system.

neutral position during the worst of the "SR" trader's drawdowns. The lower panel shows a moving average calculation of the Downside and Standard Deviations calculated from the equity curves in the top panel. These are the risk penalty terms in the DDR and SR performance functions respectively. Of particular interest is the time period between 100 and 150, where even though the "SR" trading system is recovering from a severe drawdown, the penalty term (the Standard Deviation) is increasing. Examination of the Downside Deviation shows an increase in penalty only when the drawdown becomes worse.

3.9.3 US Dollar/British Pound Foreign Exchange Trading System

In this section, the use of the downside deviation ratio to train trading systems is demonstrated on a real world data set. A {long, short, neutral} trading system is trained to trade

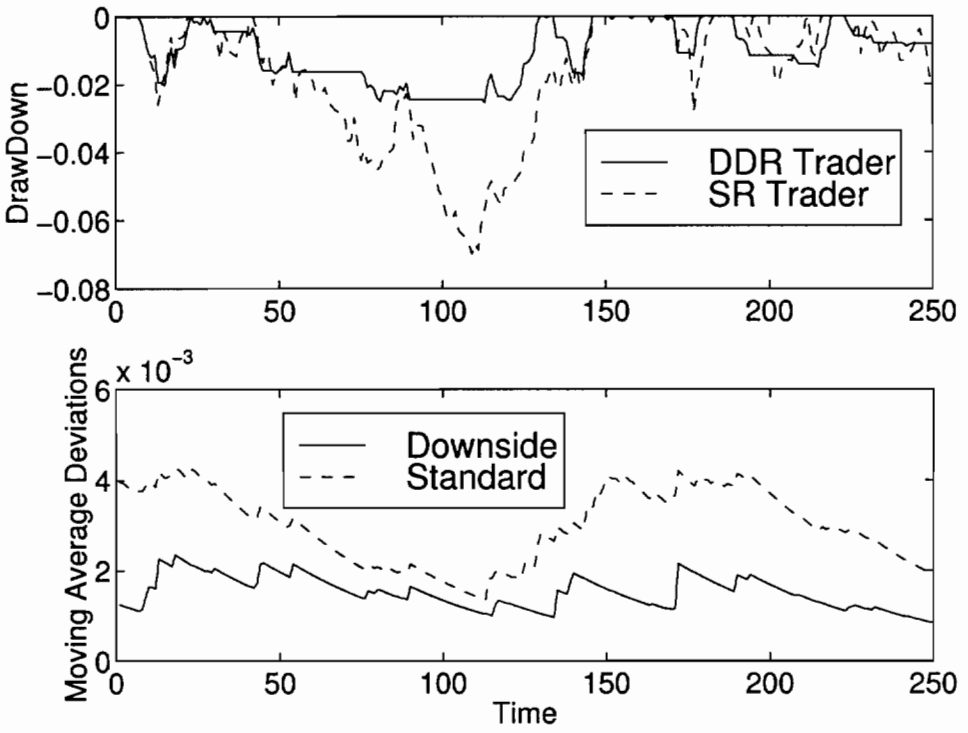

Figure 3.12: A closeup of the behavior of the "DDR" and "SR" trading systems. The top panel shows the underwater curves for the systems. The "DDR" system avoids the large drawdowns incurred by the "SR" system. The bottom panel shows a moving average calculation of the Downside Deviation and the standard deviation, the penalty terms of the two utility measures. The standard deviation, used as a penalty term in the Sharpe ratio, increases even though the "SR" system is recovering from a severe drawdown. On the other hand, the Downside Deviation only increases as the drawdown worsens for the "DDR" system.

the half-hourly US Dollar / British Pound foreign exchange (FX) rate.[17] The dataset used here consists of the first 8 months of quotes from the 24 hour, 5-days a week foreign exchange market during 1996.[18] Both bid and ask prices are in the dataset, and the trading system is required to incur the transaction costs of trading through the bid/ask prices. The trader is trained via the Recurrent Reinforcement Learning algorithm to maximize the Differential Downside Deviation Ratio (3.30).

The network models consist of a single tanh unit. The inputs to the models are based on the {1, 2, 4, 8, 16, 32, 48, 96, 240} time step backward differences of the British Pound FX rate. These differences correspond to returns on time scales of {30 min, 60 min, 2 hours, 4 hours, 8 hours, 16 hours, 24 hours, 2 days, 5 days }, and are intended to capture daily and weekly seasonal structure (if present). Note that the currency market has three major business time zones (East Asia, Europe, and North America). The return series are further processed by splitting each into two signals, one containing the positive values (or 0 when the value is negative), and the other containing the negative values (or 0 when the value is positive). This transformation is meant to allow the system to more easily react differently to up and down market conditions. These 18 signals are then reduced using a principal component analysis (using the training data) to 12 dimensions. This reduction retains 90% of the variance of the inputs. As the networks contain a bias weight and a recurrent input, there are a total of 14 network parameters.

The top panel in Figure 3.13 shows the US Dollar/British Pound price series for the 8 month period. The trading system is initially trained on the first 2000 data points, and then produces trading signals for the next 2 week period (480 data points). The training window is then shifted forward to include the just tested on data, is retrained, and its trading signals are recorded for the next 2 week out-of-sample time period. This process for generating out-of-sample trading signals continues for the rest of the data set.

The second panel in Figure 3.13 shows the out-of-sample trading signal produced by the trading system, and the third panel displays the equity curve achieved by the trader. The bottom panel shows a moving average calculation of the Sharpe Ratio over the trading

[17]The experiments described in this section were first reported in Moody & Saffell [70].

[18]The data is part of the Olsen & Associates HFDF96 dataset, obtainable by contacting www.olsen.ch.

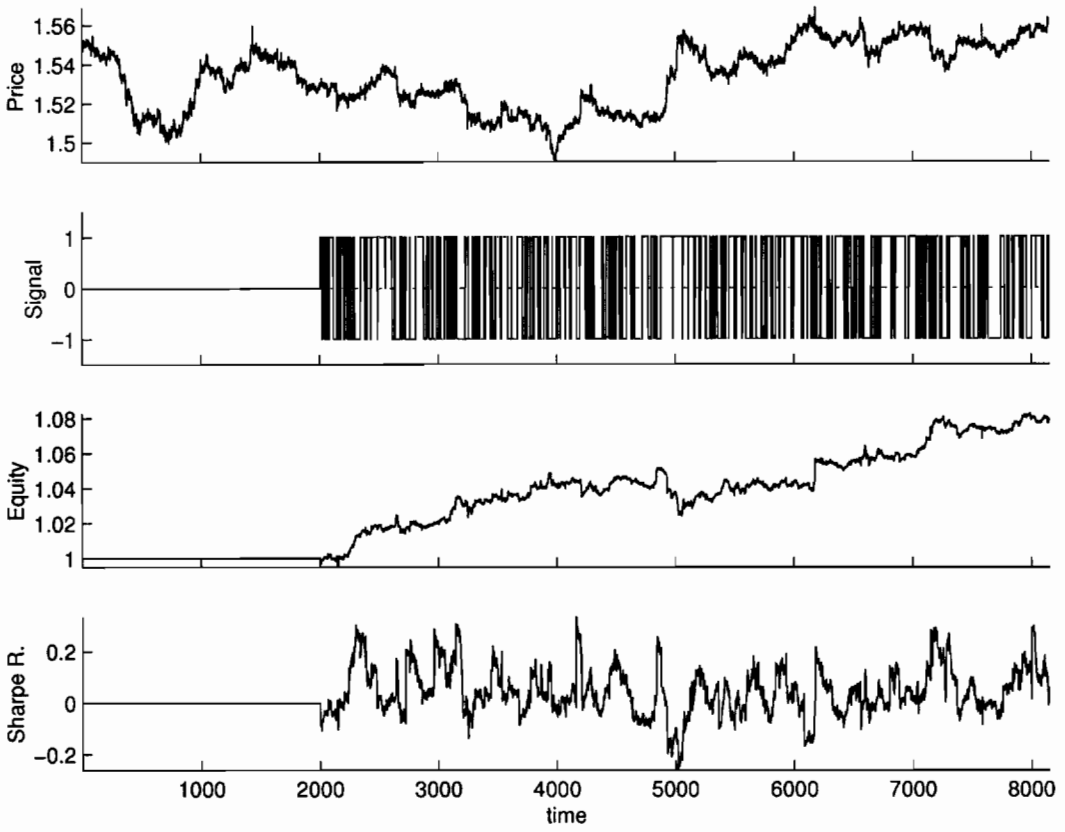

Figure 3.13: {Long, short, neutral} trading system of the US Dollar/British Pound that uses the bid/ask spread as transaction costs. The data consists of half-hourly quotes for the 5 day per week, 24 hour interbank FX market. The time period shown is the first 8 months of 1996. The trader is optimized via Recurrent Reinforcement Learning to maximize the Differential Downside Deviation Ratio. The first 2000 data points (approximately two months) are used for training and validation. The trading system achieves an annualized 15% return with an annualized Sharpe Ratio of 2.3 over the approximately 6 month long out-of-sample test period. On average, the system makes a trade once every 5 hours.

period with a time constant of 0.01. The trading system achieves an annualized 15% return with an annualized Sharpe Ratio of 2.3 over the approximately 6 month long test period. On average, the system makes a trade once every 5 hours.

These FX simulations demonstrate the ability of the RRL algorithm to discover structure in a real-world financial price series. However one must be cautious when extrapolating from simulated performance to what can be achieved in actual real-time trading. One problem is that the data set consists of indicative quotes which are not necessarily representative of the price at which the system would have actually been able to transact. A related possibility is that the system is discovering market microstructure effects that are not actually tradeable in real-time. Also, the simulation assumes that the Pound is tradeable 24 hours a day during the 5-day trading week. Certainly a real-time trading system will suffer additional penalties when trying to trade during off-peak, low liquidity trading times. An accurate test of the trading system would require live trading with a foreign exchange broker or directly through the interbank FX market in order to verify real time transactable prices and profitability.

3.9.4 S&P 500 / T-Bill Asset Allocation

In this section I compare the use of Recurrent Reinforcement Learning to the Advantage Updating formulation of the Q-Learning algorithm for building a trading system. These comparative results were presented previously at NIPS*98 [69]. The long/short trading systems trade the S&P 500 Stock Index, in effect allocating assets between the S&P 500 and 3-month Treasury Bills. When the traders are long the S&P 500, no T-Bill interest is earned, but when the traders are short stocks (using standard 2:1 leverage), they earn twice the T-Bill rate. I use the Advantage Updating refinement instead of the standard Q-Learning algorithm, because I found it to yield better trading results than the standard formulation for this problem. See Section 3.7.2 for a description of the representational advantages of the approach.

The S&P 500 target series is the total return index computed monthly by reinvesting dividends. The S&P 500 indices with and without dividends reinvested are shown in Figure 3.14 along with the 3-month Treasury Bill and S&P 500 dividend yields. The 84

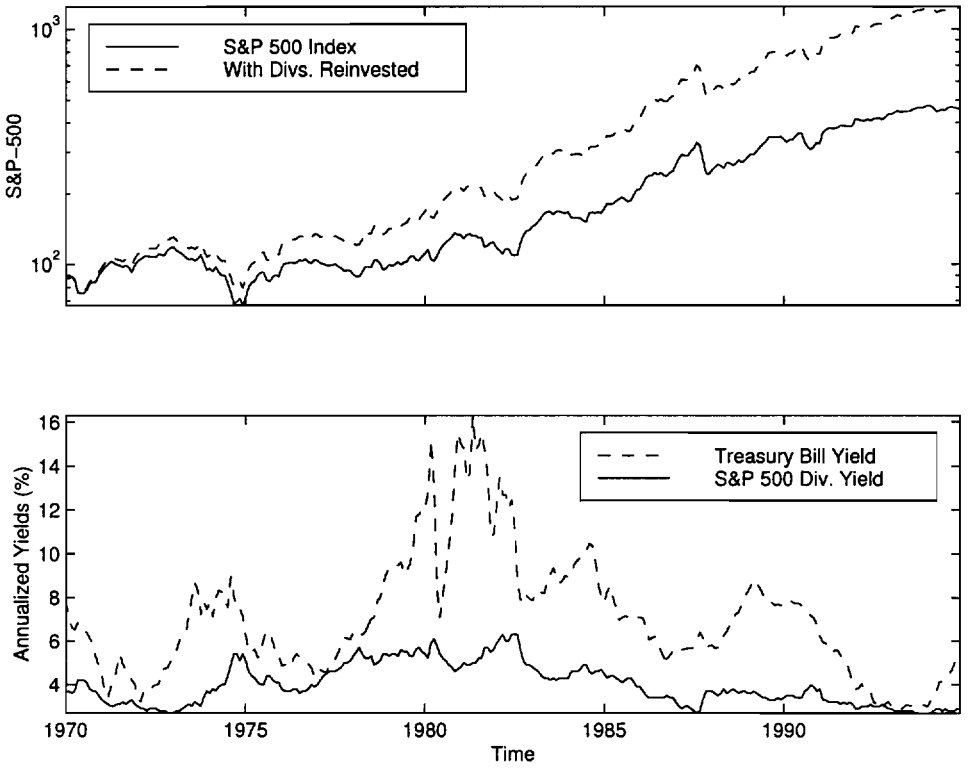

Figure 3.14: Time series that influence the return attainable by the S&P 500 / TBill asset allocation system. The top panel shows the S&P 500 series with and without dividends reinvested. The bottom panel shows the annualized monthly Treasury Bill and S&P 500 dividend yields.

monthly input series used in the trading systems include both the financial and macroeconomic data listed previously in Table 2.1. All data are obtained from Citibase,[19] and the macroeconomic series are lagged by one month to reflect reporting delays.

A total of 45 years of monthly data are used, from January 1950 through December 1994. The first 20 years of data are used only for the initial training of the system. The test period is the 25 year period from January 1970 through December 1994. The experimental results for the 25 year test period are true *ex ante* simulated trading results.

[19]Citibase historical data is obtainable from www.fame.com.

Simulation Details

The models trained here use many of the components of the methodology presented in Chapter 2. This methodology used here includes sliding window retraining, early stopping, weight decay regularization, input dimension reduction using principal components, and committee voting.

For each year during 1970 through 1994, the system is trained on a moving window of the previous 20 years of data. For 1970, the system is initialized with random parameters. For each of the 24 subsequent years, the previously learned parameters are used to initialize the training. In this way, the system is able to adapt to changing market and economic conditions. Within the moving training window, the RRL-Trader systems use the first 10 years for stochastic optimization of system parameters, and the subsequent 10 years for validating early stopping of training. The RRL-Trader networks use a single *tanh* unit, and are regularized using quadratic weight decay during training with a regularization parameter of 0.01. The 84 inputs are processed using a principal component analysis performed using the data prior to 1970, and the top 15 principal components were used as inputs to the models. Including a bias weight and the recurrent input, the RRL networks have a total of 17 network parameters.

The Q-Trader systems use a bootstrap sample of the 20 year training window for training, and the final 10 years of the training window are used for validating early stopping of training. For the results reported, the networks are two-layer feedforward networks with 30 *tanh* units in the hidden layer. As discussed previously, the Advantage Updating form of Q-Learning was used for this problem, and so required two separate networks of 30 units each to represent the Q-function. The networks are trained initially with the γ discounting factor set to 0 to learn the immediate rewards, and then γ is set to 0.75 to allow the systems to learn discounted future rewards. I find decreasing performance when the value of γ is adjusted to values higher than 0.75. The inputs are similarly the top 15 principal components as for the RRL-Trader.

To investigate the bias / variance tradeoff for the Q-Traders, I tried networks of size 10, 20, 30 and 40 hidden units. The 30 unit traders performed significantly better *out of*

sample than traders with smaller or larger networks. The 20-unit traders were significantly better than the 10-unit traders, suggesting that the smaller networks could not represent the Q function adequately (high model bias). The degradation in performance observed for the 40 unit nets suggests possible overfitting (increased model variance).[20]

Including the bias weights and the additional action input to the Advantage network, the entire Q-Trader system used here has a total of 1052 network parameters. This is a considerable increase over the complexity of the RRL-Trader systems (17 parameters). Even if the standard Q-Learning framework was used (though again, the Advantage Updating framework produced better results that the standard formulation *out of sample*) it would still have 541 network parameters.

S&P Experimental Results

Figure 3.15 shows box plots summarizing the test performance for the full 25-year test period of the trading systems with various realizations of the initial system parameters over 30 trials for the RRL-Trader system, and 10 trials for the Q-Trader system[21]. The transaction cost is set at 0.5%. Profits are reinvested during trading, and multiplicative profits are used when calculating the wealth. The notches in the box plots indicate robust estimates of the 95% confidence intervals on the hypothesis that the median is equal to the performance of the buy and hold strategy. The horizontal lines show the performance of the RRL-Trader voting, Q-Trader voting and buy and hold strategies for the same test period. Note that in this case there is a big win for the committee result over the average performance of the individual committee members. The total profits of the buy and hold strategy, the Q-Trader voting strategy and the RRL-Trader voting strategy are 1348%, 3359% and 5860% respectively. The corresponding annualized monthly Sharpe ratios 0.34, 0.63 and 0.83 respectively.[22] Remarkably, the superior results for the RRL-Trader are based on networks with a single thresholded *tanh* unit, while those for the

[20] Fewer runs were done with 40-unit traders due to the excessive computation time. Further experiments with network sizes larger than 30 would be needed to more accurately assess the possibility of overfitting.

[21] Ten trials were done for the Q-Trader system due to the amount of computation required in training the systems

[22] The Sharpe ratios calculated here are for the returns in excess of the 3-month treasury bill rate.

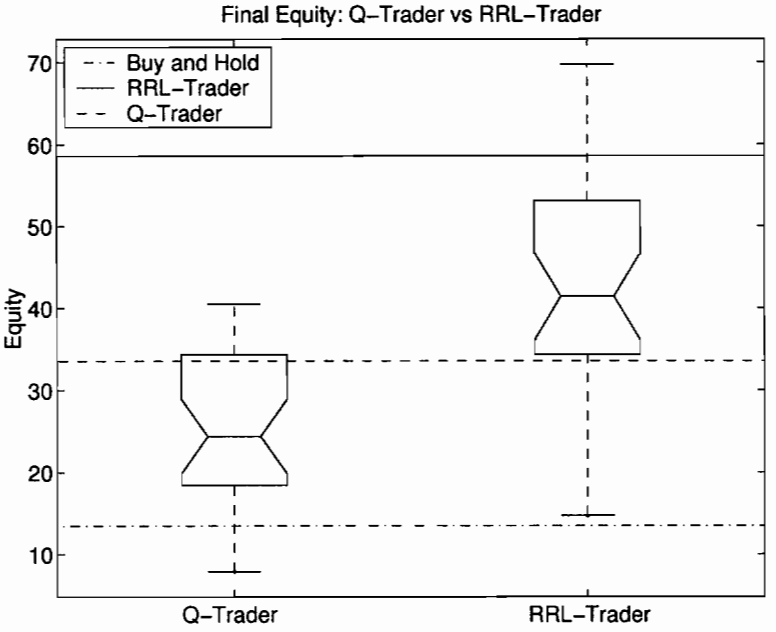

Figure 3.15: Test results for ensembles of simulations using the S&P 500 stock index and 3-month Treasury Bill data over the 1970-1994 time period. The boxplots show the performance for the ensembles of RRL-Trader and Q-Trader trading systems. The horizontal lines indicate the performance of the systems and the buy and hold strategy. The solid curves correspond to the RRL-Trader system performance, dashed curves to the Q-Trader system and the dashed and dotted curves indicate the buy and hold performance. Both systems significantly outperform the buy and hold strategy.

Q-Trader required networks with 30 hidden *tanh* units.[23]

Figure 3.16 shows results for following the strategy of taking positions based on a majority vote of the ensembles of trading systems compared with the buy and hold strategy. The trading systems go short the S&P 500 during critical periods, such as the oil price shock of 1974, the tight money periods of the early 1980's, the market correction of 1984, and the 1987 crash. This ability to take advantage of high treasury bill rates or to avoid periods of substantial stock market loss is the major factor in the long term success of these trading models. One exception is that the RRL-Trader trading system remains long during the 1991 stock market correction associated with the Persian Gulf war, a political

[23] As discussed in the Section 3.9.4, care was taken to avoid both underfitting and overfitting in the Q-Trader case, and smaller nets performed substantially worse.

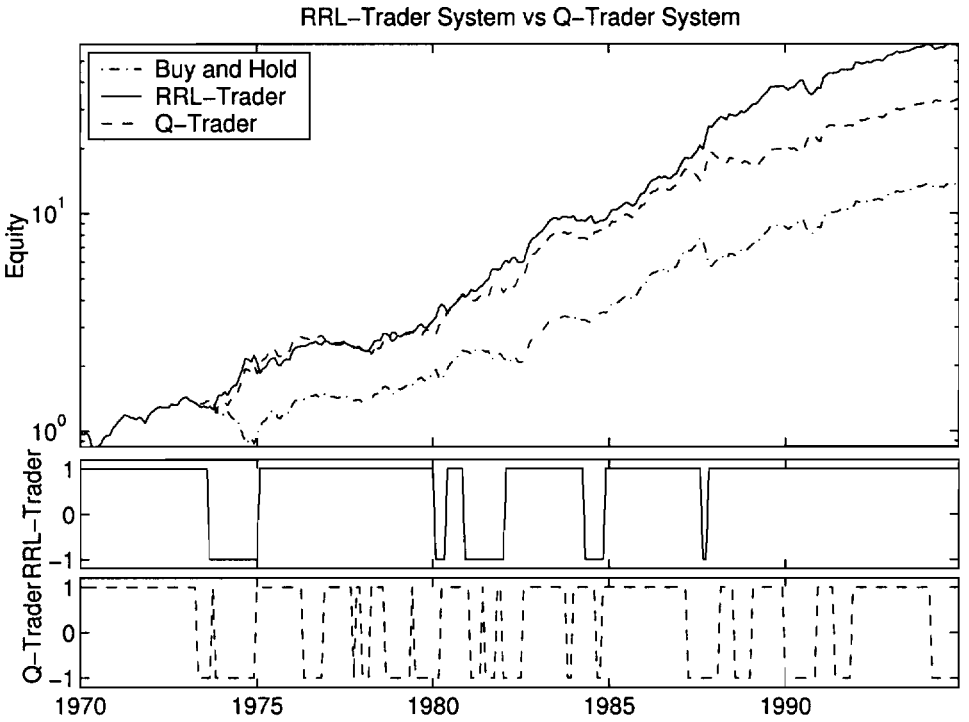

Figure 3.16: Test results for ensembles of simulations using the S&P 500 stock index and 3-month Treasury Bill data over the 1970-1994 time period. Shown are the equity curves associated with the systems and the buy and hold strategy, as well as the trading signals produced by the systems. The solid curves correspond to the RRL-Trader system performance, dashed curves to the Q-Trader system and the dashed and dotted curves indicate the buy and hold performance. Both systems significantly outperform the buy and hold strategy. In both cases, the traders avoid the dramatic losses that the buy and hold strategy incurred during 1974 and 1987.

event, though the Q-Trader system is fortunately short during the correction. On the whole though, the Q-Trader system trades much more frequently than the RRL-Trader system, and in the end does not perform as well on this data set.

From these results I find that both trading systems outperform the buy and hold strategy, as measured by both accumulated wealth and Sharpe ratio. These differences are statistically significant and support the proposition that there is predictability in the U.S. stock and treasury bill markets during the 25 year period 1970 through 1994. A more detailed presentation of the RRL-Trader results appears in Moody et al. [75]. Further

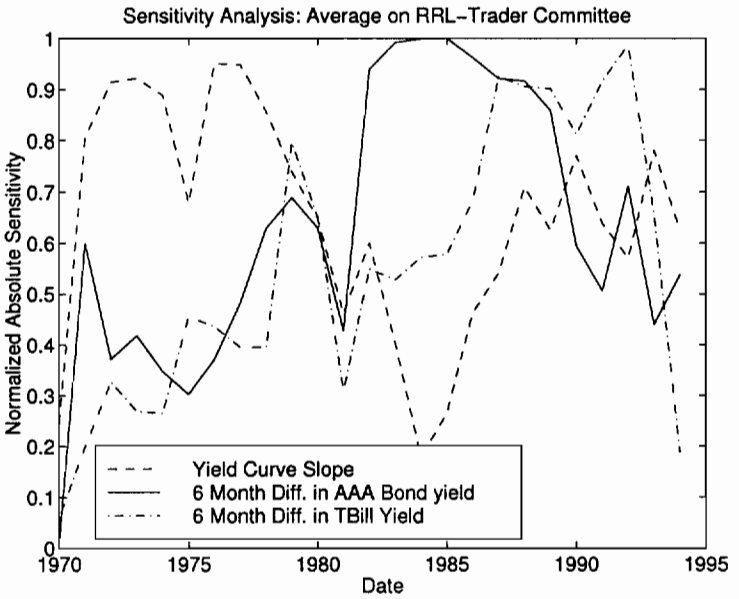

Figure 3.17: Sensitivity traces for three of the inputs to the RRL-Trader trading system averaged over the ensemble of traders. The nonstationary relationships typical among economic variables is evident from the time-varying sensitivities.

discussion of the Q-Trader vs. RRL-Trader performance is presented in Section 3.9.5.

Model Insight Through Sensitivity Analysis

A sensitivity analysis of the RRL-Trader systems was performed in an attempt to determine on which economic factors the traders are basing their decisions. Figure 3.17 shows the absolute normalized sensitivities for three of the more salient input series as a function of time, averaged over the 30 members of the RRL-Trader committee. The sensitivity of input i is defined as:

$$S_i = \frac{\left|\frac{dF}{dx_i}\right|}{\max_j \left|\frac{dF}{dx_j}\right|} \quad , \tag{3.53}$$

where F is the unthresholded trading output of the policy function and x_i denotes input i.

The time-varying sensitivities in Figure 3.17 emphasize the nonstationarity of economic relationships. For example, the yield curve slope (which measures inflation expectations)

is found to be a very important factor in the 1970's, while trends in long term interest rates (measured by the 6 month difference in the AAA bond yield) becomes more important in the 1980's, and trends in short term interest rates (measured by the 6 month difference in the treasury bill yield) dominate in the early 1990's.

3.9.5 Discussion of the S&P 500 / T-Bill Results

For the S&P 500 / T-Bill asset allocation problem described in Section 3.9.4, I find that RRL offers advantages over Q-Learning in performance, interpretability and computational efficiency. Over the 25 year test period, the RRL-Trader produced significantly higher profits (5860% vs. 3359%) and Sharpe ratios (0.83 vs. 0.63) than did the Q-Trader. The RRL-Trader learns a stable and robust trading strategy, maintaining its positions for extended periods. The frequent switches in position by the Q-Trader suggests that it is more sensitive to noise in the inputs. Hence, the strategy it has learned is brittle.

Regarding interpretability, I find the value function representation to be obscure. While the change in the policy as implemented by the RRL algorithm is directly related to changes in the inputs, for the value function the effect on policy is not so clear. While the RRL-Trader has an almost linear policy representation (a net with just a single *tanh* unit), the Q-Trader's policy is the *argmax* of a two layer network for which the policy is an *input*. The brittle behavior of the Q-Trader is probably due to the complexity of the learned Q-function with respect to the inputs and actions. The problem representation for the Q-Trader thus reduces explanatory value.

The sensitivity analysis presented for the RRL-Trader strategy in Section 3.9.4 was easy to formulate and implement. It enables identification of the most important explanatory variables, and to observe how their relative saliency varies slowly over time. For the Q-Trader, however, a similar analysis is not straightforward. The possible actions are represented as *inputs* to the Q-function network, with the chosen action being determined by the *argmax*. While I can imagine proxies for a sensitivity analysis in a simple two action {long, short} framework, it is not clear how to perform a sensitivity analysis for actions versus inputs in general for a Q-Learning framework. This reduces the explanatory value of a Q-Trader.

Since the {long, short} Q-Trader is implemented using a neural network function approximator, Bellman's curse of dimensionality has a relatively small impact on the results of the experiments presented here. The input dimensionality of the Q-Trader is increased by only one, and there are only two actions to consider. However, in the case of a portfolio management or multi-sector asset allocation system, the dimensionality problem becomes severe. Portfolio management requires a continuous weight for each of N assets included in the portfolio. This increases the input dimension for the Q-Trader by N relative to the RRL-Trader. Then, in order to facilitate the *argmax* discovery of actions, only discrete action sets can be used. The number of discrete actions that must be considered is exponential in N. Another issue is the possible loss of utility that results due to the finite resolution of action choices.

In terms of efficiency, the advantage updating representation used for the Q-Trader required two networks each with 30 *tanh* units. In order to reduce run time, the simulation code was written in C. Still, each run required approximately 25 hours to complete using a Pentium Pro 200 running the Linux operating system. The RRL networks used a single *tanh* unit, and were implemented as uncompiled Matlab code. Even given this unoptimized coding, the RRL simulations were 150 times faster, taking only 10 minutes.

3.9.6 Portfolio Management Simulation

As was stated previously, investors typically hold multiple assets at any given time. It is important to make trading decisions in the context of the portfolio as a whole. In this section, RRL is used to train a portfolio management system that make trading decisions for three artificial assets.

Portfolio System and Data

The portfolio management system is allowed to invest proportions of its wealth among three different securities with the restrictions that it must be fully invested at each time step, and that no short selling is allowed. The output of the portfolio management system

is a set of portfolio weights $\{F_t^1, F_t^2, F_t^3\}$, with the conditions that

$$F_t^a \geq 0 \text{ and } \sum_{a=1}^{3} F_t^a = 1 \ . \tag{3.54}$$

The recurrent state of the systems is similar to that described in Section 3.5.

The artificial assets are created using the random walk with trend model described in Section 3.9.1. When generating m price series according to this model, p, β, ϵ and ν become m dimensional vectors and α and k become $m \times m$ matrices. For these experiments I have $m = 3$, and set α to be a diagonal matrix with elements $\{0.85, 0.9, 0.95\}$ and k to be diagonal with elements $\{3, 3, 3\}$. Thus the series have different degrees of predictability and are uncorrelated with one another. Examples of the artificial price series are shown in the top panel of Figure 3.18.

Using the portfolio management system, I compare training to maximize the differential Sharpe ratio and training to maximize profits. I find for a variety of transaction costs, that on average, training to maximize the differential Sharpe ratio outperforms training to maximize profits.

Simulated Trading Results

Figure 3.18 shows a section of a single simulation of the portfolio management system. The trading system starts from a random initial configuration and is then adapted to optimize the differential Sharpe ratio. The transaction costs during this simulation are set at 0.5%.

Figure 3.19 shows box plots summarizing test performances for ensembles of 100 experiments. In these simulations, the trading system is initialized to a random starting condition and then adapted on-line throughout the entire data set. The simulation ensembles include 10 different initializations for each of 10 different realizations of the artificial price series. I vary the transaction costs from 0.2%, 0.5% to 1%, and observe the trading frequency, cumulative profits and Sharpe ratio on the data set. The figures show that the behavior of the portfolio management system is similar to that of the long/short trader in response to increasing transaction cost. Also, as the middle panels of Figure 3.18 demonstrate, the portfolio system tends to saturate the portfolio weights and take long/neutral

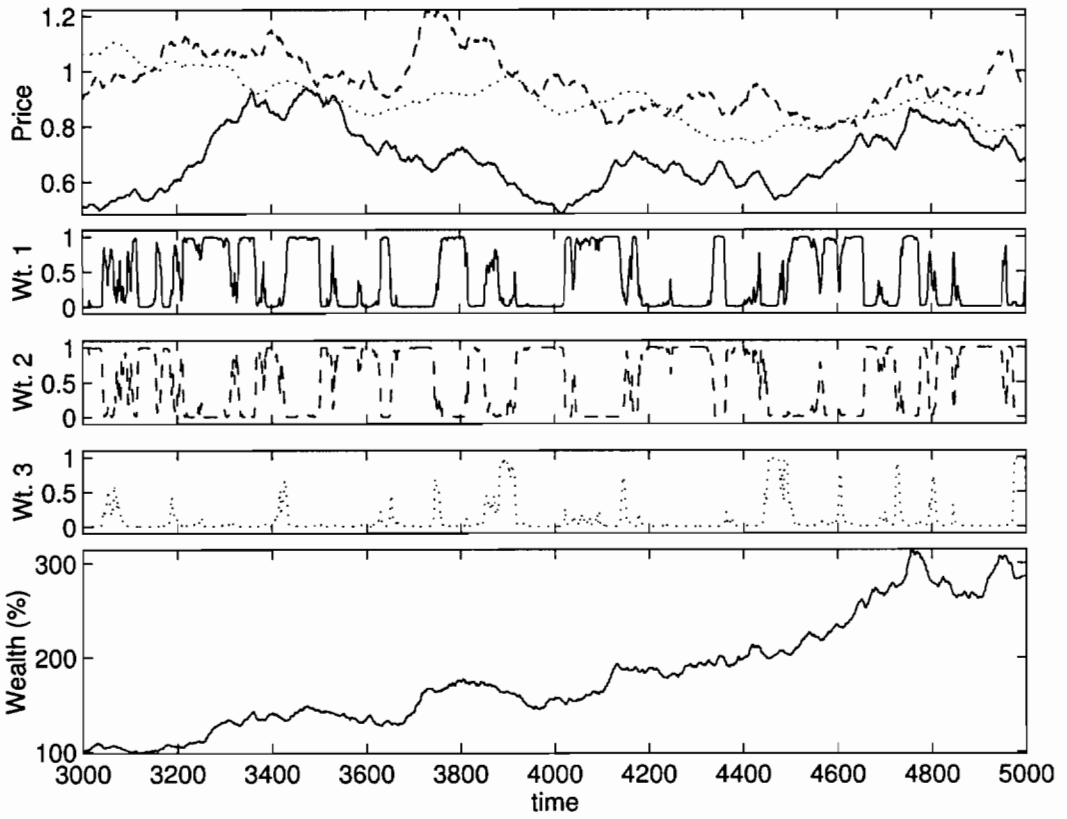

Figure 3.18: An expanded view of 2000 time periods from a simulation of the portfolio management system with transaction costs = 0.5%. The top panel shows the three artificial price series used in the simulation. The middle three panels show the corresponding portfolio weights chosen by the trading system at each time step. Note that the smoothest price series is also the least invested in, and that the portfolio system is required to be fully invested at all times. The bottom panel shows the cumulative wealth over this time period. The portfolio system tripled its wealth during this time period even though the price series showed almost no net gain during the period.

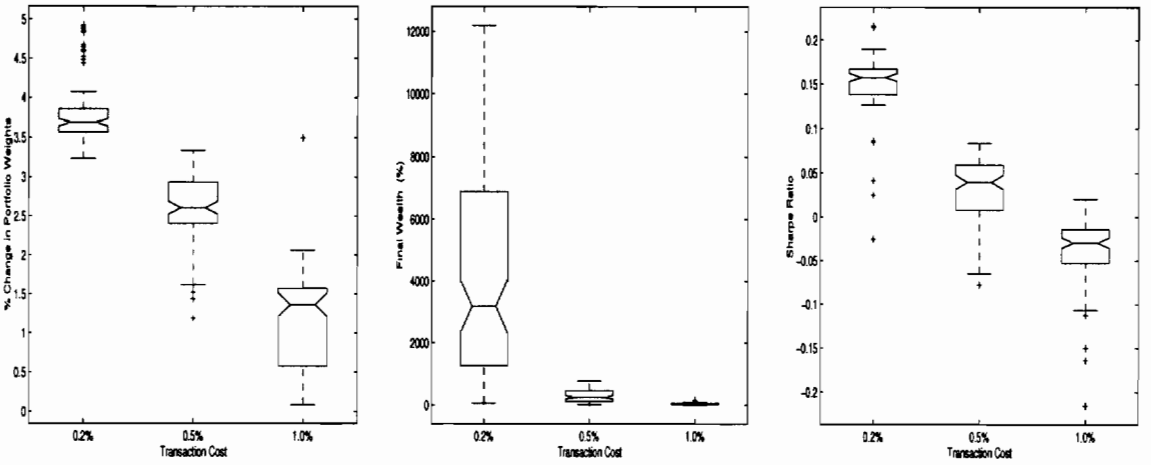

Figure 3.19: Boxplots of the average percent change in the portfolio weights, cumulative profits and Sharpe ratios vs transaction costs for the "Max.SR" portfolio management system. The results are obtained over 100 trials with various realizations of artificial data and initial system parameters. Increased transaction costs reduce the amount of change in portfolio weights, profits and Sharpe ratio, as expected. The change in portfolio weights reported here is the average of the time averages of the changes in each of the three portfolio weights. All figures are computed on the last 9,000 points in the data set.

positions in the individual securities.

Figure 3.20 compares training to maximize the differential Sharpe ratio ("Max.SR") and training to maximize cumulative profits ("Max.Profit"). Statistics are collected over an ensemble of 100 experiments as described previously. As the transactions costs increase, the "Max.SR" system actually outperforms the "Max.Profit" system in terms of average final wealth. While both systems are attempting to maximize profit, it appears that in these examples, concurrently minimizing risk can have tangible effects on actual as well as risk-adjusted profits.

3.10 Discussion

The research presented here shows that the RRL algorithm is successful in optimizing trader behavior relative to different utility functions. The use of downside performance measures, which more accurately reflects actual investor preference, had interesting effects

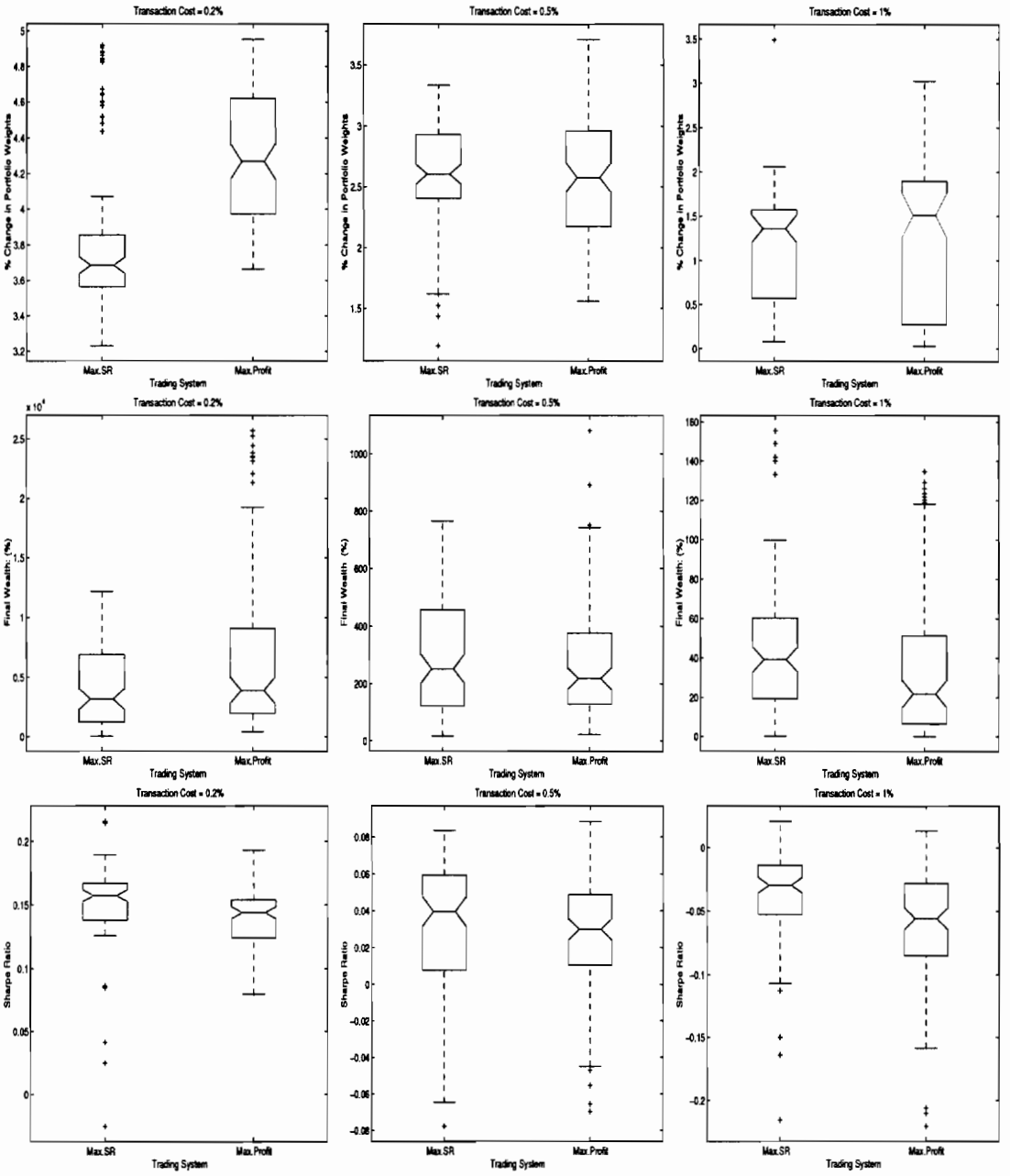

Figure 3.20: Boxplots of the percent change in portfolio weights, the log base 10 of final profits and the Sharpe ratios of the two trading systems, "Max.SR" and "Max.Profit". The change in portfolio weights reported here is the average of the average change of each of the three portfolio weights. Transaction costs are $0.2\%, 0.5\%$ and 1%. The results are obtained over 100 trials with various realizations of artificial data and initial system parameters.

on trading performance. Significant differences in behavior were apparent when comparing systems trained to optimize performance ratios based on downside risk measure to systems trained using more traditional symmetric risk measures. The systems trained using downside risk measures learn to cut their losses much more quickly than those using more traditional risk measures. This will have a significant impact on the feasibility of applying RRL to actual investment scenarios.

This work also extended the RRL single-asset model to a portfolio management system capable of trading multiple assets. The portfolio systems perform well in simulations using artificial price series. Similarly to the single asset case, the system's trading behavior changes as the costs of trading increase. Also, the systems show the ability to produce allocations that take into account the predictability of the underlying assets.

I also presented a discussion of the difference between representing policy functions and value functions. Relative to Q-Learning, RRL enables a simpler problem representation, avoids Bellman's curse of dimensionality and offers compelling advantages in efficiency. The two types of RL are compared using an S&P-500 stock index asset allocation problem. The RRL trader uses elements of the robust training methodology developed for the macroeconomic forecasting problem of Chapter 2. While the supervised forecasters were not able to find significant, nontrivial forecastable structure in the S&P-500 at the 3 and 12-month horizons as presented in Chapter 2, the RRL asset management system was able to identify a profitable trading strategy. This highlights the need to optimize systems directly using the correct utility function. Results were also presented for another real world dataset, an intra-daily British Pound foreign exchange trading system. Using RRL, the trader finds and exploits structure in the exchange rate data during 1996. These results for the RRL algorithm on actual financial dataset show that the algorithm has great promise for producing trading systems that investors can actually use in real life to manage their assets.

Chapter 4

Summary & Conclusions

In this thesis, I investigate the use of machine learning methods, including supervised and reinforcement learning algorithms, to analyze and forecast economic and financial time series. The algorithms are data driven, time series based approaches that produce quantitative forecasts and actions that result in measurable performance gains relative to standardly used techniques. As such, these algorithms have the potential to support real decision making in business, financial, and governmental institutions.

4.1 Supervised Learning for Forecasts

In Chapter 2, I presented a comprehensive methodology for forecasting macroeconomic and financial time series using nonlinear neural networks. The methodology includes several types of techniques that are in common use nowadays in the neural network community, but that are still novel for this problem domain. These include the use of bagging committees, model complexity pruning via Principal Component Pruning, and regularized nonlinear models. My goal is to produce a robust methodology for training nonlinear models using data series with limited history, poor signal to noise ratios, nonstationarities, and possible nonlinear structure. I also address the question of whether there is detectable nonlinear structure in the data.

I presented results for 3 and 12-month forecasts of macroeconomic and financial time series using the methodology. There is non-trivial predictability in the forecasts for all series considered except the Aaa-bond yield, the S&P-500, and the yield-curve slope. There is also evidence of nonlinear structure in the Index of Industrial Production and

possibly Housing Starts. While most theories of the economy are nonlinear, it seems difficult in practice to exploit these nonlinearities. There are a variety of factors that could make nonlinearity difficult to detect in the remaining series. Among them are the large amounts of noise in the series, the resulting need for heavy regularization, and the types of inputs being used. Many of the series included in this study are higher level, more broad based series than the ones that structural economic theories use as explanatory variables. However the focus of this study was on the broader indicators of economic activity.

One feature the forecast models exhibit is a large amount of nonstationarity in the learned relationship between the target and input variables. Swanson and White [107] conclude that nonstationarity is not a factor in the models they build because the largest available training window they used produced lower error rates than the smaller windows they tried. This result is more likely due to the extreme amounts of noise in the "first-reported" quarterly data they used rather than a statement about actual nonstationarity in the underlying series. In this work, the sensitivity analysis shows that the relationships between variables learned by the models changes significantly over the course of the test period.

The results show that applying nonlinear estimation and model selection techniques to linear models results in large improvements over more standard linear model fitting techniques. This includes regularized models such as the Bayesian Vector Autoregression. This result argues strongly for the use of appropriate linear models for comparison when fitting nonlinear models to these types of series. It could be that positive results in the literature from using nonlinear network models are due more to the estimation methodology than to actual nonlinear structure.

When making directional forecasts, the research showed that incorporated class-ordering information in the target representation via use of a thermometer code representation yielded significantly better results than those using a naive unary representation. Also, the regression network models trained on point forecasts perform as well in terms of directional forecasts as do the more complex classification models trained to directly forecast class labels, indicating that the additional complexity was not useful given the relatively limited history of the data.

4.1.1 Future Research Directions

Comparing the performance of models is challenging because errors are non-normal and serially correlated. The current best test is the Diebold-Mariano test. This test can have problems distinguishing between models, even when there is a relatively large difference in average error. Research is needed to develop more powerful hypothesis tests for these types of challenging series. Possible extensions could involve developing a test based on the stationary bootstrap to compare predictive accuracy.

Input selection is always a crucial issue. As was mentioned earlier, a preliminary study of the Delta Test input selection technique yielded very promising results, though the specific technique turned out not to be robust enough to be useful. Possible directions of research for input selection include sensitivity based selection or even the use of heterogeneous committees, each with its own limited input set.

4.2 Reinforcement Learning for Trading

In Chapter 3, I presented extensions and enhancements to the Recurrent Reinforcement Learning algorithm [74] that will make the algorithm more useful for investment management. These enhancements include extending the use of the algorithm to multiple asset portfolios, and extending the types of utility functions to include downside risk measures which more accurately reflect the preferences of the typical investor. The results demonstrated that the techniques are viable on real world datasets, and also compared the policy gradient approach of RRL with a popular value function algorithm called Q-Learning. Not only is the RRL trader more profitable, but it also produces more robust and interpretable trading decisions than the Q-Learning trader.

The research presented here shows that the RRL algorithm is successful in optimizing trader behavior relative to different utility functions. The use of downside performance measures, which more accurately reflects actual investor preference, has interesting effects on trading performance. Significant differences in behavior were apparent when comparing systems trained to optimize performance ratios based on downside risk measure to systems trained using more traditional symmetric risk measures. The systems trained

using downside risk measures learn to cut their losses much more quickly than those using more traditional risk measures. This will have a significant impact on the feasibility of applying RRL to actual investment scenarios.

This work also extended the RRL single-asset model to a portfolio management system capable of trading multiple assets. The portfolio systems perform well in simulations using artificial price series. Similarly to the single asset case, the system's trading behavior changes as the costs of trading increase. Also, the systems show the ability to produce allocations that take into account the predictability of the underlying assets.

I also presented a discussion of the difference between representing policy functions and value functions. Relative to Q-Learning, RRL enables a simpler problem representation, avoids Bellman's curse of dimensionality, and offers compelling advantages in efficiency. The two types of RL are compared using an S&P-500 stock index asset allocation problem. The RRL trader also uses elements of the robust training methodology developed for the macroeconomic forecasting problem of Chapter 2. While the supervised forecasters were not able to find nontrivial forecastable structure in the S&P-500 at the 3 and 12-month horizons as presented in Chapter 2, the RRL asset management system was able to identify a profitable trading strategy. This highlights the need to optimize systems directly using the correct utility function. Results were also presented for another real world dataset, an intra-daily British Pound foreign exchange trading system. Using RRL, the trader finds and exploits structure in the exchange rate data during 1996. These results for the RRL algorithm on actual financial datasets show that the algorithm has great promise for producing systems that can be used in the real world to manage investment assets.

4.2.1 Future Research Directions

There is much research that could be done regarding the use of Recurrent Reinforcement Learning with portfolios. RRL needs to be tested with larger portfolios that have more complex correlation structure, and the robustness of allocations produced needs to be characterized. The incorporation of Black-Litterman [11] type priors to help stabilize the allocations may be useful. Also, the allocation constraints could be relaxed to allow short positions to be taken, creating a comprehensive framework for multi-asset trading systems.

Bibliography

[1] ANDERS, U., AND KORN, O. Model selection in neural networks. *Neural Networks 12*, 2 (March 1999), 309–323.

[2] BAIRD, L., AND MOORE, A. Gradient descent for general reinforcement learning. In *Advances in Neural Information Processing Systems* (1999), S. A. S. Michael S. Kearns and D. A. Cohn, Eds., vol. 11, MIT Press, pp. 968–974.

[3] BAIRD, L. C. Advantage updating. Tech. Rep. WL-TR-93-1146, Wright Laboratory, Wright-Patterson Air Force Base, OH 45433-7301, 1993.

[4] BARTO, A. G. *Handbook of Intelligent Control.* Van Nostrand Reinhold, New York, 1992, ch. 12, pp. 469–492.

[5] BARTO, A. G., SUTTON, R. S., AND ANDERSON, C. W. Neuronlike adaptive elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics 13*, 5 (September 1983), 12.

[6] BAXTER, J., AND BARTLETT, P. L. Direct gradient-based reinforcement learning: I. Gradient estimation algorithms. Tech. rep., Computer Sciences Laboratory, Australian National University, 1999.

[7] BELLMAN, R. E. *Dynamic Programming.* Princeton University Press, Princeton, NJ, 1957.

[8] BENGIO, Y. Training a neural network with a financial criterion rather than a prediction criterion. In *Decision Technology for Financial Engineering* (London, 1997), A. Weigend, Y. Abu-Mostafa, and A. N. Refenes, Eds., World Scientific.

[9] BERTSEKAS, D. P. *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, MA, 1995.

[10] BERTSEKAS, D. P., AND TSITSIKLIS, J. N. *Neuro-Dynamic Programming.* Athena Scientific, 1996.

[11] BLACK, F., AND LITTERMAN, R. Asset allocation: Combining investor views with market equilibrium. *Journal of Fixed Income 1* (September 1991), 7–18.

[12] Box, G. E. P., and Cox, D. R. An analysis of transformations. *Journal of the Royal Statistical Society B 26* (1964), 211–234.

[13] Box, G. E. P., and Jenkins, G. M. *Time Series Analysis: Forecasting and Control.* Prentice Hall, 1970.

[14] Breeden, D. T. Intertemporal portfolio theory and asset pricing. In *Finance*, J. Eatwell, M. Milgate, and P. Newman, Eds. The New Palgrave, Macmillan Press, New York, 1987, pp. 180–193.

[15] Breiman, L. Bagging predictors. *Machine Learning 24*, 2 (August 1996), 123–140.

[16] Brennan, M. J., Schwartz, E. S., and Lagnado, R. Strategic asset allocation. *Journal of Economic Dynamics and Control 21* (1997), 1377–1403.

[17] Bridle, J. S. Probabilistic interpretation of feedforward classification network output, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, Architectures and Applications*, Fogelman-Soulie and Herault, Eds., NATO ASI Series. Springer, 1990.

[18] Brown, T. X. Evaluating value functions can be arbitrarily harder than evaluating policies. (http://ece.colorado.edu/~timxb/timxb/publications/0012slide.pdf) Presented at Neural Information Processing Systems Workshop, Breckenridge, CO., December 2000.

[19] Clark, W. A., and Farley, B. G. Simulation of self-organizing systems by digital computer. *IRE Transactions on Information Theory 4* (1954), 76–84.

[20] Clark, W. A., and Farley, B. G. Generalization of pattern recognition in a self-organizing system. In *Proceedings of the 1955 Western Joint Computer Conference* (1955), pp. 86–91.

[21] Clemen, R. T. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5 (1989), 559–583.

[22] Cox, J. C., Ross, S. A., and Rubinstein, M. Option pricing: A simplified approach. *Journal of Financial Economics 7* (October 1979), 229–263.

[23] Crites, R. H., and Barto, A. G. Improving elevator performance using reinforcement learning. In *Advances in NIPS* (1996), D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8, pp. 1017–1023.

[24] DIEBOLD, F. X. The past, present and future of macroeconomic forecasting. *Journal of Economic Perspectives 12* (1998), 175–192.

[25] DIEBOLD, F. X., AND MARIANO, R. S. Comparing predictive accuracy. *Journal of Business & Economic Statistics 13*, 3 (July 1995), 134–144.

[26] DIEBOLD, F. X., AND RUDEBUSCH, G. D. Forecasting output with the composite leading index: A real time analysis. *Journal of the American Statistical Association* (1991), 603–610.

[27] DOAN, T., LITTERMAN, R., AND SIMS, C. Forecasting and conditional projection using realistic prior distributions. *Econometric Reviews 3* (1984), 1–144.

[28] DUFFIE, D. *Security Markets: Stochastic Models.* Academic Press, 1988.

[29] DUFFIE, D. *Dynamic Asset Pricing Theory*, 2 ed. Princeton University Press, 1996.

[30] EFRON, B., AND TIBSHIRANI, R. J. *An Introduction to the Bootstrap.* No. 57 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1993.

[31] ELTON, E. J., AND GRUBER, M. J. Dynamic programming applications in finance. *Journal of Finance 26*, 2 (1971), 437–506.

[32] ENDERS, W. *Applied Econometric Time Series.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, 1995.

[33] EUBANK, R. L. *Spline Smoothing and Nonparametric Regression.* Marcel Dekker, Inc., 1988.

[34] FRANSES, P. H. *Time Series Models for Business and Economic Forecasting.* Cambridge University Press, 1998.

[35] GEMAN, S., BIENENSTOCK, E., AND DOURSAT, R. Neural networks and the bias/variance dilemma. *Neural Computation 4* (1992), 1–58.

[36] GOLDEN, R. M. *Mathematical Methods for Neural Network Analysis and Design.* MIT Press, 1996.

[37] GRANGER, C. W. J., AND NEWBOLD, P. *Forecasting Economic Time Series*, 2nd ed. Academic Press, San Diego, California, 1986.

[38] GRANGER, C. W. J., AND TERASVIRTA, T. *Modelling Nonlinear Economic Relationships.* Oxford University Press, 1993.

[39] GRUDIC, G. Z., AND UNGAR, L. H. Localizing policy gradient estimates to action transitions. In *Seventeenth International Conference on Machine Learning* (2000).

[40] HAMILTON, J. D. *Time Series Analysis.* Princeton University Press, 1994.

[41] HAYKIN, S. *Neural Networks: A Comprehensive Foundation.* IEEE Press, 1994.

[42] HERTZ, J., KROGH, A., AND PALMER, R. G. *Introduction to the Theory of Neural Computation.* Addison-Wesley Publishing Company, 1991.

[43] HOERL, A., AND KENNARD, R. Ridge regression: Applications to nonorthogonal problems. *Technometrics 12* (1970), 69–82.

[44] HOERL, A., AND KENNARD, R. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics 12* (1970), 55–67.

[45] JARROW, R. A. *Modelling Fixed Income Securities and Interest Rate Options.* McGraw-Hill, 1996.

[46] KAELBLING, L. P., LITTMAN, M. L., AND MOORE, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research 4* (1996), 237–285.

[47] KANG, J., CHOEY, M., AND WEIGEND, A. Nonlinear trading models and asset allocation based on Sharpe ratio. In *Decision Technology for Financial Engineering* (London, 1997), A. Weigend, Y. Abu-Mostafa, and A. N. Refenes, Eds., World Scientific.

[48] KEARNS, M., MANSOUR, Y., NG, A. Y., AND RON, D. An experimental and theoretical comparison of model selection methods. *Machine Learning 27*, 1 (April 1997), 7–50.

[49] KONDA, V. R., AND TSITSIKLIS, J. N. Actor-critic algorithms. In *Advances in Neural Information Processing Systems* (2000), S. A. Solla, T. K. Leen, and K.-R. Muller, Eds., vol. 12, MIT Press, pp. 1008–1014.

[50] LEVIN, A. U., LEEN, T. K., AND MOODY, J. E. Fast pruning using principal components. In *Advances in Neural Information Processing Systems 6* (1994), J. Cowan, G. Tesauro, and J. Alspector, Eds., Morgan Kaufmann Publishers, San Francisco, CA, pp. 35–42.

[51] LITTERMAN, R. B. Forecasting with Bayesian vector autoregressions – five years of experience. *Journal of Business and Economic Statistics 4*, 1 (1986), 25–38.

[52] LONGSTAFF, F., AND SCHWARTZ, E. Valuing American options by simulation: A simple least squares approach. *Review of Financial Studies 14*, 1 (2001), 113–147.

[53] MARBACH, P., AND TSITSIKLIS, J. N. Simulation-based optimization of Markov reward processes. In *IEEE Conference on Decision and Control* (1998), vol. 3, pp. 2698–2703.

[54] MARBACH, P., AND TSITSIKLIS, J. N. Simulation-based optimization of Markov reward processes. *IEEE Transactions on Automatic Control 46*, 6 (February 2001), 191–209.

[55] MARKOWITZ, H. *Portfolio Selection: Efficient Diversification of Investments.* New York: Wiley, 1959.

[56] MCNEES, S. K. Forecasting accuracy of alternative techniques: A comparison of U.S. macroeconomic forecasts. *Journal of Business & Economic Statistics 4*, 1 (1986), 5–23.

[57] MERTON, R. C. Lifetime portfolio selection under uncertainty: The continuous-time case. *Review of Economics and Statistics 51* (August 1969), 247–257.

[58] MERTON, R. C. Optimum consumption and portfolio rules in a continuous-time model. *Journal of Economic Theory 3* (December 1971), 373–413.

[59] MERTON, R. C. *Continuous-Time Finance.* Blackwell Publisher Inc, 1990.

[60] MILLER, I., AND MILLER, M. *John E. Freund's Mathematical Statistics.* Prentice Hall, 1998.

[61] MILLS, T. C. *The Econometric Modelling of Financial Time Series*, second ed. Cambridge University Press, 1993.

[62] MOODY, J. Challenges of economic forecasting: Noise, nonstationarity, and nonlinearity. Invited talk presented at Machines that Learn, Snowbird Utah, April 1994, Unpublished.

[63] MOODY, J. Note on generalization, regularization, and architecture selection in nonlinear learning systems. In *Proceedings of the First IEEE-SP Workshop on Neural Networks for Signal Processing* (Los Alamitos, CA, 1991), IEEE Computer Society Press, pp. 1–10.

[64] MOODY, J. The effective number of parameters: An analysis of generalization and regularization in nonlinear learning systems. In *Advances in Neural Information*

Processing Systems 4 (1992), J. E. Moody, S. J. Hanson, and R. P. Lippmann, Eds., Morgan Kaufmann Publishers, San Mateo, CA, pp. 847–854.

[65] MOODY, J. Prediction risk and neural network architecture selection. In *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, V. Cherkassky, J. Friedman, and H. Wechsler, Eds. Springer-Verlag, 1994.

[66] MOODY, J. Economic forecasting: Challenges and neural network solutions. (ftp.cse.ogi.edu/pub/neural/papers/moody95.macroeconomic.ps.Z) Keynote talk presented at the International Symposium on Artificial Neural Networks, Hsinchu, Taiwan, 1995.

[67] MOODY, J. Forecasting the economy with neural nets: A survey of challenges and solutions. In *Neural Networks: Tricks of the Trade*, G. B. Orr and K.-R. Muller, Eds. Springer Verlag, 1998, ch. 16, pp. 347–371.

[68] MOODY, J., LEVIN, A., AND REHFUSS, S. Predicting the U.S. index of industrial production. *Neural Network World 3*, 6 (1993), 791–794. Special Issue: Proceedings of Parallel Applications in Statistics and Economics '93.

[69] MOODY, J., AND SAFFELL, M. Reinforcement learning for trading. In *Advances in Neural Information Processing Systems* (1999), S. A. S. Michael S. Kearns and D. A. Cohn, Eds., vol. 11, MIT Press, pp. 917–923.

[70] MOODY, J., AND SAFFELL, M. Minimizing downside risk via stochastic dynamic programming. In *Computational Finance 1999* (2000), A. W. L. Yaser S. Abu-Mostafa, Blake LeBaron and A. S. Weigend, Eds., MIT Press, pp. 403–415.

[71] MOODY, J., AND SAFFELL, M. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks 12*, 4 (July 2001).

[72] MOODY, J., SAFFELL, M., LIAO, Y., AND WU, L. Reinforcement learning for trading systems and portfolios: Immediate vs future rewards. In *Decision Technologies for Financial Engineering* (Netherlands, 1998), A. N. Refenes, N. Burgess, and J. Moody, Eds., Kluwer. This volume is the proceedings for the 1997 Computational Finance conference held at London Business School.

[73] MOODY, J., AND UTANS, J. Architecture selection strategies for neural networks: Application to corporate bond rating prediction. In *Neural Networks in the Capital Markets*, A. N. Refenes, Ed. John Wiley & Sons, 1994, pp. 277–307.

[74] MOODY, J., AND WU, L. Optimization of trading systems and portfolios. In *Neural Networks in the Capital Markets* (London, 1997), A. S. Weigend, Y. Abu-Mostafa, and A. N. Refenes, Eds., World Scientific.

[75] MOODY, J., WU, L., LIAO, Y., AND SAFFELL, M. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting 17* (1998), 441–470.

[76] MOORE, A. W., AND ATKESON, C. G. Prioritized sweeping: Reinforcement learning with less data and less real time. *Machine Learning 13* (1993), 103–130.

[77] MOZER, M. C., AND SMOLENSKY, P. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in Neural Information Processing Systems 1* (1990), D. S. Touretzky, Ed., Morgan Kaufmann Publishers, San Mateo, CA.

[78] NARENDRA, K. S., AND PARTHASARATHY, K. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks 1*, 1 (1990), 4–27.

[79] NATTER, M., HAEFKE, C., SONI, T., AND OTRUBA, H. Adaptive methods in macroeconomic forecasting. *International Journal of Intelligent Systems in Accounting, Finance and Management 6*, 1 (March 1997), 1–10.

[80] NAWROCKI, D. Optimal algorithms and lower partial moment: Ex post results. *Applied Economics 23* (1991), 465–470.

[81] NAWROCKI, D. The characteristics of portfolios selected by n-degree lower partial moment. *International Review of Financial Analysis 1* (1992), 195–209.

[82] NAWROCKI, D. A brief history of downside risk measures. *Journal of Investing* (Fall 1999), 9–26.

[83] NELSON, C. R. *The Investor's Guide to Economic Indicators.* John Wiley and Sons, New York, 1987.

[84] NEUNEIER, R. Optimal asset allocation using adaptive dynamic programming. In *Advances in NIPS* (1996), D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8, pp. 952–958.

[85] NEUNEIER, R., AND MIHATSCH, O. Risk sensitive reinforcement learning. In *Advances in Neural Information Processing Systems* (1999), M. S. Kearns, S. A. Solla, and D. A. Cohn, Eds., vol. 11, MIT Press, pp. 1031–1037.

[86] PENG, J., AND WILLIAMS, R. J. Efficient learning and planning within the Dyna framework. *Adaptive Behavior 1*, 4 (1993), 437–454.

[87] PI, H., AND PETERSON, C. Finding the embedding dimension and variable dependencies in time series. *Neural Computation 6* (1994), 509–520.

[88] PLOCEK, J. E. *Economic Indicators: How America Reads Its Financial Health.* New York Institute of Finance, New York, 1991.

[89] REHFUSS, S. Macroeconomic forecasting with neural networks. Unpublished simulations., 1994.

[90] ROBERTSON, J. C., AND TALLMAN, E. W. Data vintages and measuring forecast model performance. *Economic Review*, 4 Q (1998), 4–20.

[91] RUMELHART, D., HINTON, G., AND WILLIAMS, R. Learning internal representations by error propagation. In *Parallel Distributed Processing: Exploration in the microstructure of cognition*, D. Rumelhart and J. McClelland, Eds. MIT Press, Cambridge, MA, 1986, ch. 8, pp. 319–362.

[92] SAMUEL, A. L. Some studies in machine learning using the game of checkers. *IBM Journal on Research and Development 3* (1959), 211–229.

[93] SAMUEL, A. L. Some studies in machine learning using the game of checkers. II – Recent progress. *IBM Journal on Research and Development 11* (1967), 601–617.

[94] SAMUELSON, P. A. Asset allocation could be dangerous to your health. *The Journal of Portfolio Management* (Spring 1990), 5–8.

[95] SATCHELL, S., AND TIMMERMANN, A. An assessment of the economic value of non-linear foreign exchange rate forecasts. *Journal of Forecasting 14* (1995), 477–497.

[96] SHARPE, W. F. Mutual fund performance. *Journal of Business* (Jan 1966), 119–138.

[97] SOLLA, S., CUN, Y. L., AND DENKER, J. Optimal brain damage. In *Advances in Neural Information Processing Systems* (1990), D. S. Touretzky, Ed., vol. 2, Morgan Kaufmann Publishers Inc., pp. 598–605.

[98] SORTINO, F., AND FORSEY, H. On the use and misuse of downside risk. *The Journal of Portfolio Management 22* (1996), 35–42.

[99] SORTINO, F., AND VAN DER MEER, R. Downside risk – capturing what's at stake in investment situations. *The Journal of Portfolio Management 17* (1991), 27–31.

[100] STORK, D., AND HASSIBI, B. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in Neural Information Processing Systems* (1993), T. J. Sejnowski, G. E. Hinton, and D. S. Touretzky, Eds., vol. 5, Morgan Kaufmann Publishers Inc., pp. 164–171.

[101] STUTELY, R. *The Economist Guide to Economic Indicators.* John Wiley and Sons, New York, 1997.

[102] SUTTON, R. S. Temporal credit assignment in reinforcement learning. Ph.D. thesis, University of Massachusetts, Amherst, 1984.

[103] SUTTON, R. S. Learning to predict by the method of temporal differences. *Machine Learning 3*, 1 (1988), 9–44.

[104] SUTTON, R. S., AND BARTO, A. G. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA, 1997.

[105] SUTTON, R. S., MCALLESTER, D., SINGH, S., AND MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems* (2000), T. K. L. Sara A. Solla and K.-R. Muller, Eds., vol. 12, MIT Press, pp. 1057–1063.

[106] SWANSON, N., AND WHITE, H. A model selection approach to real-time macroeconomic forecasting using linear models and artificial neural networks. Discussion paper, Department of Economics, Pennsylvania State University, 1995.

[107] SWANSON, N., AND WHITE, H. A model selection approach to real-time macroeconomic forecasting using linear models and artificial neural networks. *Review of Economics and Statistics 79*, 4 (1997), 540–550.

[108] SWANSON, N. R., AND FRANSES, P. H. Nonlinear econometric modelling: A selective review. In *Nonlinear Time Series Analysis of Economic and Financial Data*, P. Rothman, Ed. Kluwer Academic Publishers, Boston, 1999, ch. 4, pp. 87–109.

[109] TESAURO, G. TD-Gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation 6*, 2 (1994), 215–219.

[110] TESAURO, G. Temporal difference learning and TD-Gammon. *Communications of the ACM 38*, 3 (1995), 58–68.

[111] TIMMERMANN, A., AND PESARAN, H. Predictability of stock returns: Robustness and economic significance. *Journal of Finance 50* (1995), 1201–1228.

[112] TSITSIKLIS, J. N., AND ROY, B. V. Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives. *IEEE Transactions on Automatic Control 44*, 10 (October 1999), 1840–1851.

[113] TSITSIKLIS, J. N., AND ROY, B. V. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks 12*, 4 (July 2001), 694–703.

[114] UTANS, J., MOODY, J., REHFUSS, S., AND SIEGELMANN, H. Input variable selection for neural networks: Application to predicting the U.S. business cycle. In *Proceedings of Computational Intelligence for Financial Engineering* (1995), pp. 118–122.

[115] VAN ROY, B. Temporal-difference learning and applications in finance. In *Computational Finance 1999* (2001), Y. S. Abu-Mostafa, B. LeBaron, A. W. Lo, and A. S. Weigend, Eds., MIT Press, pp. 447–461.

[116] WAHBA, G. *Spline Models for Observational Data.* No. 59 in Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1990.

[117] WATKINS, C. J., AND DAYAN, P. Technical note: Q-Learning. *Machine Learning 8*, 3 (1992), 4.

[118] WATKINS, C. J. C. H. *Learning with Delayed Rewards.* PhD thesis, Cambridge University, Psychology Department, 1989.

[119] WERBOS, P. Back-propagation through time: What it does and how to do it. *IEEE Proceedings 78*, 10 (Oct. 1990), 1550–1560.

[120] WERBOS, P. Neurocontrol and supervised learning: An overview and evaluation. In *Handbook of Intelligent Control*, D. A. White and D. A. Sofge, Eds. Van Nostrand Reinhold, New York, 1992, ch. 3, pp. 65–90.

[121] WHITE, H. Personal communication. Unpublished. 1996.

[122] WIDROW, B., AND HOFF, M. E. Adaptive switching circuits. In *IRE WESCON Convention Record* (1960), pp. 96–104.

[123] WILLIAMS, R. J. Toward a theory of reinforcement-learning connectionist systems. Tech. Rep. NU-CCS-88-3, College of Computer Science, Northeastern University, Boston, MA, 1988.

[124] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning 8* (1992), 229–256.

[125] WILLIAMS, R. J., AND ZIPSER, D. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation 1* (1989), 270–280.

[126] WINKLER, R. L., AND MAKRIDAKIS, S. The combination of forecasts. *Journal of the Royal Statistical Society, A*, 146 (1983), 150–157.

[127] WU, L., AND MOODY, J. A smoothing regularizer for feedforward and recurrent networks. *Neural Computation 8*, 3 (1996), 461–489.

[128] ZELLNER, A., AND PALM, F. Time series analysis and simultaneous equation econometric models. *Journal of Econometrics 2* (1974), 17–54.

[129] ZHANG, W., AND DIETTERICH, T. G. High-performance job-shop scheduling with a time-delay td(λ) network. In *Advances in NIPS* (1996), D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, Eds., vol. 8, pp. 1024–1030.

Appendix A

Tables of Results

This appendix lists tables of results for the macroeconomic prediction problem presented in Chapter 2 along with a variety of performance metrics for evaluating the performance of the predictive models.

A.1 Performance Metrics

For these results I use three primary performance metrics: Normalized Root Mean Square Error, (NRMSE), Normalized Mean Absolute Error (NMAE), and Misclassification Rate. All metrics are averages over the entire test set, 1980 – 1990. In the following formulas, t is the target, p is the prediction, and $\langle\cdot\rangle$ gives the expected value over the test period. The NRMSE is the root mean square error, normalized by the variance of the target series over the test period,

$$\sqrt{\langle(t-p)^2\rangle / \langle(t-\bar{t})^2\rangle}.$$

This definition of NRMSE can be interpreted as "fraction of unexplained variance". The NMAE is the average absolute error, normalized by the mean absolute deviation of the target series over the test period,

$$\langle|t-p|\rangle / \langle|t-\bar{t}|\rangle.$$

The NMAE behaves similarly to NRMSE, but is more robust to outliers in the data. The Misclassification Rate is the percent of the predictions which have the opposite sign of the target. This is a measure of how well the model predicts the direction of change of

the series. The Misclassification Rate can be calculated from the Confusion Matrix. The Confusion Matrix is a table that contains a count of the realized combinations of up and down predictions with up and down targets:

$$\begin{array}{cc} & \text{target} \\ & \begin{array}{cc} \textit{up} & \textit{down} \end{array} \\ \text{predicted} \begin{array}{c} \textit{up} \\ \textit{down} \end{array} & \begin{pmatrix} 12 & 5 \\ 7 & 14 \end{pmatrix} \end{array}$$

The Misclassification Rate is the sum of the off-diagonal elements divided by the total number of elements. In the previous example, the Misclassification Rate is $(7+5)/(12+14+7+5) = 0.3158$. If a target is zero, then that point is not included in the calculation of the confusion matrix. If the prediction of a given model is zero then that point is not included in the calculation of the confusion matrix for that model.

The Confusion Matrix calculated for the point forecast results uses a return of 0 as the dividing line between up and down returns. In the case of quintile directional forecasts (for the classification and quantized regression models), the median return is used as the dividing line. Thus an up target corresponds to a target in the top two quintiles of the returns distribution, and a down target corresponds to a target return in the bottom two quintiles of the returns distribution. The middle quintile (around the median) is not included as an up or down return.

A.2 Tables of Results

The following tables list numerical results of the performance measures described previously for the point and direction forecasts. The series are listed in alphabetical order by the Citibase designation (Table 2.1), and for each series, results for the 3-month and 12-month forecasts are given.

For each series-horizon combination, there are a set of three tables: the first for point forecasts, the second for the quantized point forecasts evaluated against the classification targets, and the third for the direct classification forecasts. In each table, the best model, according to each performance measure, is marked with a '*'. The models which are not

significantly worse than the best at the 5% significance level (according to the Diebold-Mariano test as described in Section 2.6.3) are also marked with a '*'.

The significance of the regression models results in the top table on each page are tested relative to each other. The significance of the quantized regression and classification models results in the bottom two tables on each page are tested relative to each other as a whole.

Refer to Section 2.5 for descriptions of the models used.

Table A.1: DLEAD.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.001	1.459	1.069	0.554*	0.928	0.492*	0.579
NRMSE	1.000	1.208	1.034	0.744*	0.964	0.701*	0.761
NMAE	0.998	1.125	1.003	0.832*	1.013*	0.759*	0.801*
Confusion Matrix	73 45 0 0	54 19 19 26	52 19 21 26	57 15 16 30	47 14 26 31	57 13 16 32	60 12 13 33
Misclass. Rate	0.381	0.322	0.339	0.263	0.339	0.246	0.212

Table A.2: DLEAD.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.817*	2.375	2.108	1.425*	1.750*	1.258*	1.250*
RMSE	1.348*	1.541	1.452	1.194*	1.323*	1.122*	1.118*
MAE	1.117	1.142	1.125	0.892*	0.950*	0.858*	0.833*
Confusion Matrix	0 0 0 0	22 11 0 0	17 8 12 26	32 5 7 30	24 5 13 31	31 2 7 33	33 1 6 34
Misclass. Rate	—	0.333	0.317	0.162	0.247	0.123	0.095

Table A.3: DLEAD.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.817*	2.375	1.667*	1.667*	1.700*	1.308*	1.408*	1.250*
RMSE	1.348*	1.541	1.291*	1.291*	1.304*	1.144*	1.187*	1.118*
MAE	1.117	1.142	0.917*	0.917*	0.967	0.792*	0.842*	0.800*
Confusion Matrix	0 0 0 0	22 11 0 0	38 5 5 23	38 5 5 23	37 6 4 21	33 3 5 26	35 6 5 27	35 4 3 26
Misclass. Rate	—	0.333	0.141	0.141	0.147	0.119	0.151	0.103

Table A.4: DLEAD.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.031	2.380	1.137	0.497*	1.414	0.341*	0.334*
NRMSE	1.015	1.543	1.066	0.705*	1.189	0.584*	0.578*
NMAE	1.026	1.542	1.000	0.720	1.036	0.555*	0.577*
Confusion	73 47	43 29	59 35	55 4	52 10	60 3	63 5
Matrix	0 0	30 18	14 12	18 43	21 37	13 44	10 42
Misclass. Rate	0.392	0.492	0.408	0.183	0.258	0.133	0.125

Table A.5: DLEAD.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.700	3.783	2.267	0.792*	1.767	0.625*	0.650*
RMSE	1.304	1.945	1.506	0.890*	1.329	0.791*	0.806*
MAE	1.100	1.500	1.150	0.625*	1.017	0.508*	0.567*
Confusion	0 0	15 18	3 7	38 1	18 10	33 0	34 0
Matrix	0 0	0 0	14 35	0 50	8 50	2 50	1 50
Misclass. Rate	—	0.545	0.356	0.011	0.209	0.024	0.012

Table A.6: DLEAD.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.700	3.783	0.908	1.300	0.833*	0.867*	0.817*	0.792*
RMSE	1.304	1.945	0.953	1.140	0.913*	0.931*	0.904*	0.890*
MAE	1.100	1.500	0.675*	0.850	0.650*	0.617*	0.633*	0.592*
Confusion	0 0	15 18	35 4	37 6	35 7	38 5	37 5	39 5
Matrix	0 0	0 0	0 41	2 43	0 39	0 53	0 49	0 47
Misclass. Rate	—	0.545	0.050	0.091	0.086	0.052	0.055	0.055

Table A.7: DRM.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.005*	2.615	1.114*	1.519	2.403	0.870*	0.870*
NRMSE	1.003*	1.617	1.055*	1.232	1.550	0.933*	0.933*
NMAE	0.995*	1.615	1.000*	1.272	0.994	0.972*	0.958*
Confusion	0 0	18 37	28 28	23 29	34 38	28 22	25 18
Matrix	55 65	37 28	27 37	32 36	21 27	27 43	30 47
Misclass. Rate	0.458	0.617	0.458	0.508	0.492	0.408	0.400

Table A.8: DRM.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.383	5.592	3.033	4.592	4.600	3.167*	2.575*
RMSE	1.544	2.365	1.742	2.143	2.145	1.780*	1.605*
MAE	1.350*	1.942	1.417*	1.692	1.650	1.333*	1.242*
Confusion	0 0	17 27	21 16	21 20	33 23	23 13	22 11
Matrix	0 0	0 0	6 8	22 17	16 19	16 22	14 23
Misclass. Rate	—	0.614	0.431	0.525	0.429	0.392	0.357

Table A.9: DRM.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.383	5.592	4.567	2.325*	4.192	3.208*	2.400*	3.400
RMSE	1.544	2.365	2.137	1.525*	2.047	1.791*	1.549*	1.844
MAE	1.350*	1.942	1.617	1.325*	1.542	1.375	1.317*	1.400
Confusion	0 0	17 27	20 10	0 0	17 8	22 12	6 4	24 11
Matrix	0 0	0 0	26 28	0 0	21 26	17 23	9 14	18 25
Misclass. Rate	—	0.614	0.429	0.000	0.403	0.392	0.394	0.372

Table A.10: DRM.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.054*	2.893	0.922*	1.382	3.232	0.728*	0.787*
NRMSE	1.027*	1.701	0.960*	1.175	1.798	0.853*	0.887*
NMAE	1.023*	1.627	0.991*	1.311	1.049	0.897*	0.909*
Confusion	0 0	32 34	20 30	29 35	35 40	33 29	34 29
Matrix	55 65	23 31	35 35	26 30	20 25	22 36	21 36
Misclass. Rate	0.458	0.475	0.542	0.508	0.500	0.425	0.417

Table A.11: DRM.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.967*	5.125	1.825*	3.708	3.933	1.983*	2.092*
RMSE	1.402*	2.264	1.351*	1.926	1.983	1.408*	1.446*
MAE	1.167*	1.808	1.142*	1.525	1.567	1.033*	1.025*
Confusion	0 0	29 27	25 13	27 15	39 20	33 6	32 8
Matrix	0 0	0 0	2 11	14 18	13 9	10 23	13 25
Misclass. Rate	—	0.482	0.294	0.392	0.407	0.222	0.269

Table A.12: DRM.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.967*	5.125	3.325	3.292*	3.500	2.917*	1.883*	2.325*
RMSE	1.402*	2.264	1.823	1.814*	1.871	1.708*	1.372*	1.525*
MAE	1.167*	1.808	1.358	1.325	1.433	1.300	0.983*	1.092
Confusion	0 0	29 27	33 7	33 8	31 11	31 9	30 5	34 8
Matrix	0 0	0 0	24 29	24 29	21 25	20 25	18 25	19 25
Misclass. Rate	—	0.482	0.333	0.340	0.364	0.341	0.295	0.314

Table A.13: FM2DQ.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.061	1.199	1.021	0.486*	1.493	0.571*	0.612*
NRMSE	1.030	1.095	1.010	0.697*	1.222	0.756*	0.782*
NMAE	1.025*	1.149	1.004	0.747*	1.021	0.825*	0.842*
Confusion Matrix	86 34 0 0	68 15 18 19	76 25 10 9	78 15 8 19	46 14 40 20	76 18 10 16	76 14 10 20
Misclass. Rate	0.283	0.275	0.292	0.192	0.450	0.233	0.200

Table A.14: FM2DQ.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.983*	2.775	2.625	1.575*	3.492	1.858*	1.933*
RMSE	1.408*	1.666	1.620	1.255*	1.869	1.363*	1.390*
MAE	1.167*	1.258	1.292	0.842*	1.375	0.975*	1.000*
Confusion Matrix	0 0 0 0	16 11 0 0	13 9 14 30	27 7 4 36	11 9 22 44	23 8 11 36	23 10 8 36
Misclass. Rate	—	0.407	0.348	0.149	0.360	0.244	0.234

Table A.15: FM2DQ.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.983*	2.775	3.825	3.625	3.617	2.483	2.408	2.250*
RMSE	1.408*	1.666	1.956	1.904	1.902	1.576	1.552	1.500*
MAE	1.167*	1.258	1.408	1.375	1.383	1.133	1.125	1.033*
Confusion Matrix	0 0 0 0	16 11 0 0	25 18 13 39	25 15 13 42	25 16 13 41	24 12 8 39	24 15 10 34	27 13 7 38
Misclass. Rate	—	0.407	0.326	0.295	0.305	0.241	0.301	0.235

Table A.16: FM2DQ.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.143*	1.019*	1.031*	0.670*	4.453	0.720*	0.676*
NRMSE	1.069*	1.010*	1.016*	0.818*	2.110	0.849*	0.822*
NMAE	1.045*	1.045*	1.023*	0.805*	1.017	0.875*	0.858*
Confusion Matrix	89 31 0 0	74 10 15 21	79 31 10 0	79 11 10 20	39 9 50 22	88 28 1 3	82 26 7 5
Misclass. Rate	0.258	0.208	0.342	0.175	0.492	0.242	0.275

Table A.17: FM2DQ.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.967*	2.192*	2.525	1.508*	3.550*	1.633	1.350*
RMSE	1.402*	1.480*	1.589	1.228*	1.884*	1.278	1.162*
MAE	1.183*	1.125*	1.358	0.808*	1.367	1.017*	0.917*
Confusion Matrix	0 0 0 0	13 9 0 0	15 13 13 31	25 9 7 40	14 5 23 51	12 6 10 36	18 6 9 39
Misclass. Rate	—	0.409	0.361	0.198	0.301	0.250	0.208

Table A.18: FM2DQ.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.967*	2.192*	2.975	3.533	3.533	2.192*	2.208	2.467
RMSE	1.402*	1.480*	1.725	1.880	1.880	1.480*	1.486	1.571
MAE	1.183*	1.125*	1.242	1.350	1.350	1.058	1.075	1.217
Confusion Matrix	0 0 0 0	13 9 0 0	26 14 12 39	26 16 12 43	26 16 12 43	23 15 10 39	26 13 9 38	26 13 12 37
Misclass. Rate	—	0.409	0.286	0.289	0.289	0.287	0.256	0.284

Table A.19: FSPCOM.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.019*	2.056	1.094*	1.333	2.292	0.927*	0.924*
NRMSE	1.009*	1.434	1.046*	1.154	1.514	0.963*	0.961*
NMAE	1.020*	1.356	1.027*	1.222	1.022	0.970*	0.976*
Confusion Matrix	81 39 0 0	57 23 24 16	73 33 8 6	52 21 29 18	38 24 43 15	70 25 11 14	66 20 15 19
Misclass. Rate	0.325	0.392	0.342	0.417	0.558	0.300	0.292

Table A.20: FSPCOM.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.258*	3.875	2.417*	3.692	5.375	2.267*	2.300*
RMSE	1.503*	1.968	1.555*	1.921	2.318	1.506*	1.517*
MAE	1.308*	1.525	1.300*	1.492	1.908	1.167*	1.217*
Confusion Matrix	0 0 0 0	30 17 0 0	3 0 8 7	23 15 21 20	22 20 32 15	20 9 14 19	20 7 18 21
Misclass. Rate	—	0.362	0.444	0.456	0.584	0.371	0.379

Table A.21: FSPCOM.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.258*	3.875	4.467	3.758	4.342	2.525*	2.242*	2.908*
RMSE	1.503*	1.968	2.113	1.939	2.084	1.589*	1.497*	1.705*
MAE	1.308*	1.525	1.617	1.458	1.592	1.208*	1.208*	1.325
Confusion Matrix	0 0 0 0	30 17 0 0	30 13 25 23	28 13 13 13	27 12 23 17	22 7 20 24	22 10 14 20	18 7 19 24
Misclass. Rate	—	0.362	0.418	0.388	0.443	0.370	0.364	0.382

Table A.22: FSPCOM.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.130*	2.995	1.222*	1.502*	3.098	0.996*	1.088*
NRMSE	1.063*	1.731	1.105*	1.226*	1.760	0.998*	1.043*
NMAE	1.126	1.721*	1.162	1.236	1.013*	1.059	1.107
Confusion	90 30	60 30	90 30	66 16	58 18	67 28	66 23
Matrix	0 0	30 0	0 0	24 14	32 12	23 2	24 7
Misclass. Rate	0.250	0.500	0.250	0.333	0.417	0.425	0.392

Table A.23: FSPCOM.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.617*	6.775	2.633*	3.242*	5.250	2.600*	2.692*
RMSE	1.618*	2.603	1.623*	1.800*	2.291	1.612*	1.641*
MAE	1.483*	2.208	1.467*	1.525*	1.767	1.383	1.408
Confusion	0 0	27 33	0 0	26 4	30 10	17 0	13 0
Matrix	0 0	0 0	4 5	22 23	33 14	20 19	22 22
Misclass. Rate	—	0.550	0.444	0.347	0.494	0.357	0.386

Table A.24: FSPCOM.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.617*	6.775	3.783	3.950	3.425*	3.100*	2.392*	2.592*
RMSE	1.618*	2.603	1.945	1.987	1.851*	1.761*	1.546*	1.610*
MAE	1.483*	2.208	1.600	1.583	1.342*	1.417*	1.275*	1.358*
Confusion	0 0	27 33	28 6	28 4	42 2	27 1	30 0	22 0
Matrix	0 0	0 0	28 21	26 28	24 28	23 32	26 29	23 22
Misclass. Rate	—	0.550	0.410	0.349	0.271	0.289	0.306	0.343

Table A.25: FYAAAC.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.044*	2.051	1.086*	1.135*	2.450	1.033*	1.043*
NRMSE	1.022*	1.432	1.042*	1.065*	1.565	1.016*	1.021*
NMAE	1.021	1.389*	1.016	1.103	0.984*	1.014	1.017
Confusion Matrix	56 63 0 0	29 30 27 33	34 45 22 18	37 39 19 24	24 33 32 30	55 54 1 9	56 58 0 5
Misclass. Rate	0.529	0.479	0.563	0.487	0.546	0.462	0.487

Table A.26: FYAAAC.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.875*	5.183	3.917	4.608	6.308	3.392*	3.167*
RMSE	1.696*	2.277	1.979	2.147	2.512	1.842*	1.780*
MAE	1.575*	1.767*	1.650*	1.625*	1.975	1.608*	1.567*
Confusion Matrix	0 0 0 0	23 24 0 0	21 18 17 18	28 26 17 28	19 30 27 30	19 27 3 10	15 17 2 6
Misclass. Rate	—	0.511	0.473	0.434	0.538	0.508	0.475

Table A.27: FYAAAC.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.875*	5.183	4.742	5.492	6.025	4.283	3.075*	3.750
RMSE	1.696*	2.277	2.178	2.343	2.455	2.070	1.754*	1.936
MAE	1.575*	1.767*	1.692*	1.825	1.942	1.700*	1.525*	1.667
Confusion Matrix	0 0 0 0	23 24 0 0	19 28 12 19	18 24 20 21	20 34 15 16	19 28 15 14	22 21 2 11	15 26 6 10
Misclass. Rate	—	0.511	0.513	0.530	0.576	0.566	0.411	0.561

Table A.28: FYAAAC.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.096*	1.792	1.107*	1.145*	4.304	1.091	1.053*
NRMSE	1.047*	1.339	1.052*	1.070*	2.075	1.044	1.026*
NMAE	1.020*	1.312*	1.018	1.047*	1.003	1.023	0.999*
Confusion Matrix	64 56 0 0	40 33 24 23	56 53 8 3	49 37 15 19	42 19 22 37	64 56 0 0	64 56 0 0
Misclass. Rate	0.467	0.475	0.508	0.433	0.342	0.467	0.467

Table A.29: FYAAAC.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.850*	4.417*	3.617*	3.725*	3.692*	3.058*	2.850*
RMSE	1.688*	2.102*	1.902*	1.930*	1.921*	1.749*	1.688*
MAE	1.517*	1.583*	1.733*	1.525*	1.242*	1.575*	1.517*
Confusion Matrix	0 0 0 0	21 19 0 0	7 15 11 7	21 25 11 25	32 19 12 41	0 0 5 1	0 0 0 0
Misclass. Rate	—	0.475	0.650	0.439	0.298	0.833	0.000

Table A.30: FYAAAC.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.850*	4.417*	4.417*	4.983	4.683*	4.375*	4.942*	4.842*
RMSE	1.688*	2.102*	2.102*	2.232	2.164*	2.092*	2.223*	2.200*
MAE	1.517*	1.583*	1.583*	1.700*	1.600*	1.575*	1.742*	1.692*
Confusion Matrix	0 0 0 0	21 19 0 0	18 18 25 32	16 25 19 29	18 21 24 33	3 3 42 56	0 0 47 58	0 0 43 56
Misclass. Rate	—	0.475	0.462	0.494	0.469	0.433	0.448	0.434

Table A.31: HSBP.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.000	2.011	1.266	1.184	1.402	0.647*	0.664*
NRMSE	1.000	1.418	1.125	1.088	1.184	0.804*	0.815*
NMAE	0.999	1.371	0.999	1.115	1.056	0.793*	0.823*
Confusion	56 57	31 27	31 25	45 28	42 28	46 20	46 19
Matrix	0 0	25 30	25 32	11 29	14 29	10 37	10 38
Misclass. Rate	0.504	0.460	0.442	0.345	0.372	0.265	0.257

Table A.32: HSBP.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.233	4.167	3.483	2.883	3.183	1.592*	1.642*
RMSE	1.494	2.041	1.866	1.698	1.784	1.262*	1.281*
MAE	1.283	1.550	1.467	1.233	1.417	0.958*	0.975*
Confusion	0 0	20 16	7 14	33 20	27 20	33 9	35 9
Matrix	0 0	0 0	15 20	6 26	9 27	5 29	6 28
Misclass. Rate	—	0.444	0.518	0.306	0.349	0.184	0.192

Table A.33: HSBP.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.233	4.167	3.242	3.867	3.733	2.525	2.033	2.508
RMSE	1.494	2.041	1.800	1.966	1.932	1.589	1.426	1.584
MAE	1.283	1.550	1.308	1.450	1.417	1.208	1.133	1.158
Confusion	0 0	20 16	31 14	36 20	33 17	37 14	29 14	37 20
Matrix	0 0	0 0	13 35	11 33	14 36	5 26	9 26	5 23
Misclass. Rate	—	0.444	0.290	0.310	0.310	0.232	0.295	0.294

Table A.34: HSBP.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.026*	2.244	1.302	0.926	2.283	0.813*	0.591*
NRMSE	1.013*	1.498	1.141	0.962	1.511	0.902*	0.769*
NMAE	1.044*	1.555	1.011	1.009	0.988	0.903*	0.751*
Confusion Matrix	46 71 0 0	15 31 31 40	24 40 22 31	40 30 6 41	27 20 19 51	30 21 16 50	40 26 6 45
Misclass. Rate	0.607	0.530	0.530	0.308	0.333	0.316	0.274

Table A.35: HSBP.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.783*	4.075	2.775	2.175	2.625	1.575*	1.492*
RMSE	1.335*	2.019	1.666	1.475	1.620	1.255*	1.221*
MAE	1.100*	1.625	1.342	1.092	1.192	0.992*	0.758*
Confusion Matrix	0 0 0 0	11 17 0 0	10 11 15 20	30 20 2 22	23 16 11 50	18 6 3 13	33 8 4 29
Misclass. Rate	—	0.607	0.464	0.297	0.270	0.225	0.162

Table A.36: HSBP.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.783*	4.075	2.733	2.883	3.550	2.033*	1.967*	2.242*
RMSE	1.335*	2.019	1.653	1.698	1.884	1.426*	1.402*	1.497*
MAE	1.100*	1.625	1.167	1.217	1.400	0.983*	0.917*	1.225
Confusion Matrix	0 0 0 0	11 17 0 0	28 22 7 43	27 23 8 45	27 27 6 38	29 23 3 32	29 24 1 37	27 24 2 12
Misclass. Rate	—	0.607	0.290	0.301	0.337	0.299	0.275	0.400

Table A.37: IP.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.270	1.430	0.936*	1.011	1.473	0.748	0.649*
NRMSE	1.127	1.196	0.967*	1.005	1.214	0.865	0.806*
NMAE	1.134	1.132	1.005*	1.043	1.148	0.920	0.847*
Confusion Matrix	80 39 0 0	62 21 18 18	80 37 0 2	65 21 15 18	48 4 32 35	74 29 6 10	74 28 6 11
Misclass. Rate	0.328	0.328	0.311	0.303	0.303	0.294	0.286

Table A.38: IP.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.700	1.750	1.358*	1.692	1.400*	1.125*	1.150*
RMSE	1.304	1.323	1.165*	1.301	1.183*	1.061*	1.072*
MAE	1.083	0.967	0.892*	0.925*	0.850*	0.775*	0.800*
Confusion Matrix	0 0 0 0	12 7 0 0	5 2 3 34	16 14 4 43	11 3 8 58	17 7 1 43	13 8 1 41
Misclass. Rate	—	0.368	0.114	0.234	0.138	0.118	0.143

Table A.39: IP.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.700	1.750	1.950	1.167*	1.458*	1.458	1.333*	1.300*
RMSE	1.304	1.323	1.396	1.080*	1.208*	1.208	1.155*	1.140*
MAE	1.083	0.967	1.033	0.767*	0.842*	0.892*	0.850*	0.867*
Confusion Matrix	0 0 0 0	12 7 0 0	16 20 1 36	12 8 3 45	19 15 3 46	15 13 2 43	19 11 1 35	16 13 1 32
Misclass. Rate	—	0.368	0.288	0.162	0.217	0.205	0.182	0.226

Table A.40: IP.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.561	2.447	1.115*	1.206	2.422	0.724	0.578*
NRMSE	1.249	1.564	1.056*	1.098	1.556	0.851	0.760*
NMAE	1.321	1.563	1.062*	1.158	0.974	0.868*	0.766*
Confusion Matrix	86 31 0 0	66 21 20 10	86 31 0 0	74 15 12 16	56 17 30 14	77 11 9 20	80 9 6 22
Misclass. Rate	0.265	0.350	0.265	0.231	0.402	0.171	0.128

Table A.41: IP.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.600*	3.017	1.275*	1.633	1.608	0.992*	0.967*
RMSE	1.265*	1.737	1.129*	1.278	1.268	0.996*	0.983*
MAE	1.050*	1.283*	0.792*	0.933	0.975*	0.675*	0.667*
Confusion Matrix	0 0 0 0	1 16 0 0	0 0 2 45	13 17 0 47	9 5 6 63	15 10 1 58	15 11 0 57
Misclass. Rate	—	0.941	0.043	0.221	0.133	0.131	0.133

Table A.42: IP.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.600*	3.017	1.833	1.492*	1.750	1.433*	0.958*	1.158
RMSE	1.265*	1.737	1.354	1.221*	1.323	1.197*	0.979*	1.076
MAE	1.050*	1.283*	1.033	0.908*	0.983	0.800*	0.742*	0.842
Confusion Matrix	0 0 0 0	1 16 0 0	16 20 0 51	16 15 0 54	16 19 0 52	15 17 0 59	15 12 0 53	15 17 0 55
Misclass. Rate	—	0.941	0.230	0.176	0.218	0.187	0.150	0.195

Table A.43: LHUR.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.001	0.978	0.811	0.670*	1.414	0.514*	0.531*
NRMSE	1.001	0.989	0.900	0.819*	1.189	0.717*	0.729*
NMAE	1.000	0.965	1.006	0.846*	1.086	0.736*	0.759*
Confusion	38 60	27 23	29 25	26 11	28 36	30 15	34 17
Matrix	0 0	11 37	9 35	12 49	10 24	8 45	4 43
Misclass. Rate	0.612	0.347	0.347	0.235	0.469	0.235	0.214

Table A.44: LHUR.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.900	2.067	2.092	1.692*	3.092	1.825	1.800
RMSE	1.378	1.438	1.446	1.301*	1.758	1.351	1.342
MAE	1.217	1.000	1.058	0.875*	1.358	0.975	1.000
Confusion	0 0	36 17	42 15	41 6	44 28	39 15	43 17
Matrix	0 0	0 0	13 22	11 33	13 8	7 34	5 30
Misclass. Rate	—	0.321	0.304	0.187	0.441	0.232	0.232

Table A.45: LHUR.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.900	2.067	1.933	2.292	1.525*	1.600*	1.233*	1.233*
RMSE	1.378	1.438	1.390	1.514	1.235*	1.265*	1.111*	1.111*
MAE	1.217	1.000	0.867*	1.008	0.775*	0.983	0.867*	0.800*
Confusion	0 0	36 17	50 13	59 33	58 17	37 5	43 9	43 7
Matrix	0 0	0 0	10 32	1 12	2 28	11 30	2 18	10 30
Misclass. Rate	—	0.321	0.219	0.324	0.181	0.193	0.153	0.189

Table A.46: LHUR.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.013	1.858	1.000	0.608	2.218	0.326*	0.342*
NRMSE	1.006	1.363	1.000	0.780	1.489	0.571*	0.585*
NMAE	0.979*	1.346	1.052	0.851	0.971	0.630*	0.635*
Confusion	0 0	27 19	21 45	36 21	27 35	40 26	39 23
Matrix	40 75	13 56	19 30	4 54	13 40	0 49	1 52
Misclass. Rate	0.348	0.278	0.557	0.217	0.417	0.226	0.209

Table A.47: LHUR.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	2.142	3.208	2.300	1.458	4.042	1.058*	1.067*
RMSE	1.463	1.791	1.517	1.208	2.010	1.029*	1.033*
MAE	1.208*	1.458	1.267	0.875	1.608	0.725*	0.700*
Confusion	0 0	28 14	26 17	38 7	28 21	42 13	41 11
Matrix	0 0	0 0	4 11	4 30	13 16	1 33	2 37
Misclass. Rate	—	0.333	0.362	0.139	0.436	0.157	0.143

Table A.48: LHUR.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	2.142	3.208	1.708	1.600	1.292*	1.025*	2.142	0.808*
RMSE	1.463	1.791	1.307	1.265	1.137*	1.012*	1.463	0.899*
MAE	1.208*	1.458	0.942	0.817*	0.742*	0.658*	1.008*	0.675*
Confusion	0 0	28 14	29 4	34 3	39 4	38 2	21 1	39 3
Matrix	0 0	0 0	5 33	5 34	2 32	5 36	16 38	4 37
Misclass. Rate	—	0.333	0.127	0.105	0.078	0.086	0.224	0.084

Table A.49: PUNEW.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.174	0.712	0.552*	0.447*	1.661	0.444*	0.518
NRMSE	1.084	0.844	0.743*	0.669*	1.289	0.666*	0.719
NMAE	0.982*	0.865	1.000*	0.687*	1.129	0.708*	0.776*
Confusion Matrix	114 4 0 0	110 4 4 0	112 4 2 0	113 4 1 0	110 4 4 0	114 4 0 0	114 4 0 0
Misclass. Rate	0.034	0.068	0.051	0.042	0.068	0.034	0.034

Table A.50: PUNEW.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.542	1.508	1.133*	1.117*	1.950	1.192	1.525
RMSE	1.242	1.228	1.065*	1.057*	1.396	1.092	1.235
MAE	0.958*	0.842	0.717*	0.733*	1.050	0.758*	0.858
Confusion Matrix	0 0 0 0	34 8 0 0	60 6 6 0	38 3 4 12	56 6 7 0	43 9 3 5	46 12 6 5
Misclass. Rate	—	0.190	0.167	0.123	0.188	0.200	0.261

Table A.51: PUNEW.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.542	1.508	1.400	1.358	1.267	1.050*	0.917*	0.992*
RMSE	1.242	1.228	1.183	1.165	1.125	1.025*	0.957*	0.996*
MAE	0.958*	0.842	0.883	0.875	0.833	0.733*	0.650*	0.642*
Confusion Matrix	0 0 0 0	34 8 0 0	63 3 8 6	61 3 8 4	61 3 8 4	60 4 6 4	59 3 3 2	61 3 3 2
Misclass. Rate	—	0.190	0.138	0.145	0.145	0.135	0.090	0.087

Table A.52: PUNEW.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.432*	0.502*	0.423*	0.361*	2.862	0.429*	0.598*
NRMSE	1.196*	0.709*	0.650*	0.601*	1.692	0.655*	0.773*
NMAE	0.977*	0.727*	1.023*	0.628*	1.052	0.683*	0.817*
Confusion Matrix	120 0 0 0	120 0 0 0	119 0 1 0	120 0 0 0	108 0 12 0	120 0 0 0	120 0 0 0
Misclass. Rate	0.000	0.000	0.008	0.000	0.100	0.000	0.000

Table A.53: PUNEW.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.367*	1.000*	0.975*	1.000*	2.592	1.000*	1.225*
RMSE	1.169*	1.000*	0.987*	1.000*	1.610	1.000*	1.107*
MAE	0.867*	0.750*	0.608*	0.683*	1.142	0.683*	0.792*
Confusion Matrix	0 0 0 0	37 0 0 0	71 7 11 0	48 3 3 0	69 10 16 0	50 11 1 0	47 12 2 0
Misclass. Rate	—	0.000	0.202	0.111	0.274	0.194	0.230

Table A.54: PUNEW.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.367*	1.000*	1.533	1.692	1.633	2.092	1.350*	1.475
RMSE	1.169*	1.000*	1.238	1.301	1.278	1.446	1.162*	1.214
MAE	0.867*	0.750*	0.967	1.025	0.967	1.058	0.833	0.892
Confusion Matrix	0 0 0 0	37 0 0 0	72 0 14 0	72 5 10 0	75 4 10 0	66 0 18 0	73 8 12 0	70 8 14 0
Misclass. Rate	—	0.000	0.163	0.172	0.157	0.214	0.215	0.239

Table A.55: YCS.L.FD3 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.037*	2.318	1.075*	1.297*	3.567	1.238	1.086*
NRMSE	1.018*	1.522	1.037*	1.139*	1.889	1.112	1.042*
NMAE	0.987*	1.491	0.993*	1.296	0.987	1.175	1.103
Confusion	0 0	19 33	22 35	25 33	25 32	17 28	17 30
Matrix	53 67	34 34	31 32	28 34	28 35	36 39	36 37
Misclass. Rate	0.442	0.558	0.550	0.508	0.500	0.533	0.550

Table A.56: YCS.L.FD3 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.892*	4.225	2.692	3.975	4.658	3.175	2.825
RMSE	1.375*	2.055	1.641	1.994	2.158	1.782	1.681
MAE	1.158*	1.608	1.275*	1.542	1.708	1.458	1.392
Confusion	0 0	19 22	21 19	23 22	25 19	15 14	17 14
Matrix	0 0	0 0	12 15	22 18	22 21	21 13	15 9
Misclass. Rate	—	0.537	0.463	0.518	0.471	0.556	0.527

Table A.57: YCS.L.FD3 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.892*	4.225	4.100	4.292	5.075	3.575	3.358	3.075
RMSE	1.375*	2.055	2.025	2.072	2.253	1.891	1.833	1.754
MAE	1.158*	1.608	1.600	1.642	1.808	1.508	1.442	1.442
Confusion	0 0	19 22	18 19	24 31	21 23	25 25	36 26	18 16
Matrix	0 0	0 0	19 16	13 8	32 17	18 9	9 4	20 9
Misclass. Rate	—	0.537	0.528	0.579	0.591	0.558	0.467	0.571

Table A.58: YCS.L.FD12 Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
NMSE	1.287*	2.556	1.004*	1.597*	9.050	1.055*	1.078*
NRMSE	1.134*	1.599	1.002*	1.264*	3.008	1.027*	1.038*
NMAE	1.155	1.594	1.040	1.254	1.011*	1.089	1.100
Confusion	0 0	37 32	30 22	36 27	31 20	34 15	33 13
Matrix	69 51	32 19	39 29	33 24	38 31	35 36	36 38
Misclass. Rate	0.575	0.533	0.508	0.500	0.483	0.417	0.408

Table A.59: YCS.L.FD12 Quantized Regression Results

	Trivial Predictors		Iterated Predictions		Direct Predictions		
Quantized Regression:	Median Return	No Change Return	Linear AR	Bayesian VAR	Linear Regression	Linear Network	Neural3 Network
MSE	1.683*	3.717	1.892*	2.833*	5.508	1.925*	2.100*
RMSE	1.297*	1.928	1.375*	1.683*	2.347	1.387*	1.449*
MAE	1.100*	1.483*	1.058*	1.283*	1.958	1.058*	1.133*
Confusion	0 0	37 18	44 19	38 19	32 14	40 17	36 13
Matrix	0 0	0 0	4 3	17 5	37 12	6 0	8 0
Misclass. Rate	—	0.327	0.329	0.456	0.537	0.365	0.368

Table A.60: YCS.L.FD12 Classification Results

	Trivial Classifiers		Softmax Representation			Thermometer Representation		
Classification:	Median Return	No Change Return	Linear	Neural3	Neural10	Linear	Neural3	Neural10
MSE	1.683*	3.717	3.417	3.433	3.500	2.808*	2.708*	2.858*
RMSE	1.297*	1.928	1.848	1.853	1.871	1.676*	1.646*	1.691*
MAE	1.100*	1.483*	1.450*	1.417*	1.433*	1.308*	1.258*	1.325*
Confusion	0 0	37 18	40 15	46 19	50 21	43 12	43 13	37 12
Matrix	0 0	0 0	23 11	22 6	18 2	16 8	17 6	18 5
Misclass. Rate	—	0.327	0.427	0.441	0.429	0.354	0.380	0.417

Biographical Note

Matthew John Saffell was born August 14, 1970 in Youngstown, Ohio. He received a B.S. in Computer Science and Engineering with a minor in Mathematics from LeTourneau University in 1992. In 1994 he received an M.S. in Computer Science from the University of Tennessee. His areas of interest include machine learning, time series, and finance. Matthew served on the Student Council at OGI for a total of four years, the last year of which he was Student Body President. He also received the Distinguished Student Award in 1998.