Learning to Trade via Direct Reinforcement

John Moody, Matthew Saffell

Abstract— **We present methods for optimizing portfolios, asset allocations and trading systems based on Direct Reinforcement. In this approach, investment decision making is viewed as a stochastic control problem, and strategies are discovered directly. We present an adaptive algorithm called Recurrent Reinforcement Learning (RRL) for discovering investment policies. The need to build forecasting models is eliminated, and better trading performance is obtained.**

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. We find that the RRL Direct Reinforcement framework enables a simpler problem representation, avoids Bellman's curse of dimensionality and offers compelling advantages in efficiency.

We demonstrate how Direct Reinforcement can be used to optimize risk-adjusted investment returns (including the differential Sharpe ratio), while accounting for the effects of transaction costs. In extensive simulation work using real financial data, we find that our approach based on RRL produces better trading strategies than systems utilizing Q-Learning (a value function method). Real world applications include an intra-daily currency trader and a monthly asset allocation system for the S&P 500 Stock Index and T-Bills.

Keywords— **Recurrent Reinforcement Learning, Direct Reinforcement, policy gradient, value function, trading, Differential Sharpe Ratio, Downside Deviation, risk, Q-Learning, TD-Learning.**

I. Introduction

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. In this paper, we describe *Direct Reinforcement* (DR) methods to optimize investment performance criteria. Investment decision making is viewed as a stochastic control problem, and strategies are discovered directly. We present an adaptive algorithm called Recurrent Reinforcement Learning (RRL). The need to build forecasting models is eliminated, and better trading performance is obtained. This methodology can be applied to optimizing systems designed to trade a single security, allocate assets or manage a portfolio.

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,[1] 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.

In Moody et al [1], [2], we proposed the RRL algorithm for Direct Reinforcement. RRL is an adaptive *policy search* algorithm that can learn an investment strategy on-line. We demonstrated in those papers that Direct Reinforcement provides a more elegant and effective means for training trading systems and portfolio managers when market frictions are considered than do more standard supervised approaches.

In this paper, we contrast our *Direct Reinforcement* (or "policy search") approach with commonly used value function based approaches. We use the term "Direct Reinforcement" 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 [3], [4], but have received little attention from the reinforcement learning community during the past two decades. Notable exceptions are Williams' REINFORCE algorithm [5], [6] and Baxter & Bartlett's recent work [7].[2]

Methods such as dynamic programming[8], TD-Learning[9] or Q-Learning[10], [11] 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 [12], [13] 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".[3]

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 [17], [18], an elevator scheduler [19] and a space-shuttle payload scheduler [20]. 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. One class of performance criteria frequently used in the financial community

The authors are with the Computational Finance Program, Oregon Graduate Institute of Science and Technology, 20000 NW Walker Rd., Beaverton, OR 97006. E-mail: {moody, saffell}@cse.ogi.edu. The authors are also with Nonlinear Prediction Systems.

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

[2]Baxter & Bartlett have independently proposed the term "Direct Reinforcement" for *policy gradient* algorithms in a Markov Decision Process framework. We 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.

[3]For reviews and in-depth presentations of value function and actor-critic methods, see [14], [15], [16].

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. We describe commonly used measures of risk, and review how *differential* forms of the Sharpe Ratio and Downside Deviation Ratio can be formulated to enable efficient online learning with RRL.

We present empirical results for discovering tradeable structure in the US Dollar/British Pound foreign exchange market via Direct Reinforcement. In addition, we compare performance for an RRL-Trader and Q-Trader that learn switching strategies between the S&P 500 Stock Index and Treasury Bills. For both traders, the results demonstrate the presence of predictable structure in US stock prices over a 25-year test period. However, we find that the RRL-Trader performs substantially better than the Q-Trader. Relative to Q-Learning, we observe that RRL enables a simpler problem representation, avoids Bellman's curse of dimensionality and offers compelling advantages in efficiency. The S&P 500 and foreign exchange results were previously presented in [2], [21], [22].

We discuss the relative merits of Direct Reinforcement and Value Function learning, and provide arguments and examples for why value function based methods may result in unnatural problem representations. Our results suggest that Direct Reinforcement offers a powerful alternative to reinforcement algorithms that learn a value function, for problem domains where immediate estimates of incremental performance can be obtained.

To conclude the introduction, we would like to note that computational finance offers many interesting and challenging potential applications of reinforcement learning methods. While our work emphasizes Direct Reinforcement, most applications in finance to date have been based upon dynamic programming type methods. Elton & Gruber [23] 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 [24], [25], [26] 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 [27]. Duffie [28], [29] describes stochastic control and dynamic programming methods in finance in depth. Dynamic programming provides the basis of the Cox, Ross, Rubinstein [30] and other widely used binomial option pricing methods. See also the strategic asset allocation work of Brennan et al. [31]. Due to the curse of dimensionality, approximate dynamic programming is often required to solve practical problems, as in the work by Longstaff and Schwartz [32] 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 [33] uses a TD(λ) approach for valuing options and performing portfolio optimization. Neuneier [34] uses a Q-Learning approach to make asset allocation decisions, and Neuneier & Mihatsch [35] incorporate a notion of risk sensitivity into the construction of the Q-Function. Derivatives pricing applications have been studied by Tsitsiklis and Van Roy [36], [37]. Moody and Saffell compare Direct Reinforcement to Q-Learning for asset allocation in [21], and explore the minimization of downside risk using Direct Reinforcement in [22].

II. Trading Systems and Performance Criteria

A. Structure of Trading Systems

In this paper, we consider 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 [2] for a discussion of multiple asset portfolios.

Here, our 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.[4]

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 decision function:

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

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:

$$F_t = \text{sign}(\, uF_{t-1} + v_0 r_t + v_1 r_{t-1} + \ldots + v_m r_{t-m} + w \,) \,, \tag{2}$$

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 IV-A.

The above formulation describes a discrete-action, deterministic trader, but can be easily generalized. One simple

[4]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.

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 $F()$:

$$F_t = F(\theta_t; F_{t-1}, I_t; \epsilon_t) \quad \text{with} \quad \epsilon_t \sim p_\epsilon(\epsilon) \quad . \tag{3}$$

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

$$p(F_t; \theta_t; F_{t-1}, I_t) \quad . \tag{4}$$

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.

B. 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. We 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:

$$\begin{aligned} P_T &= \sum_{t=1}^{T} R_t \quad \text{where} \\ R_t &\equiv \mu \left\{ r_t^f + F_{t-1}(r_t - r_t^f) - \delta |F_t - F_{t-1}| \right\} \end{aligned} \tag{5}$$

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:

$$R_t = \mu \left\{ F_{t-1} r_t - \delta |F_t - F_{t-1}| \right\} \quad . \tag{6}$$

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:

$$\begin{aligned} W_T &= W_0 \prod_{t=1}^{T} \{1 + R_t\} \quad \text{where} \\ \{1 + R_t\} &\equiv \left\{ 1 + (1 - F_{t-1}) r_t^f + F_{t-1} r_t \right\} \times \\ &\qquad \{1 - \delta |F_t - F_{t-1}|\} \quad . \end{aligned} \tag{7}$$

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

$$\{1 + R_t\} = \{1 + F_{t-1} r_t\} \{1 - \delta |F_t - F_{t-1}|\} \quad . \tag{8}$$

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 [2].

C. Performance Criteria

In general, the performance criteria that we consider 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

$$U(W_T, \ldots, W_t, \ldots, W_1, W_0) \quad . \tag{9}$$

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 intertemporal 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

$$U(R_T, \ldots, R_t, \ldots, R_2, R_1; W_0) \quad . \tag{10}$$

For brevity, we denote this general form by U_T.

For optimizing our traders, we will be interested in the marginal increase in performance due to return R_t at each time step:

$$D_t \propto \Delta U_t = U_t - U_{t-1} \; . \tag{11}$$

Note that U_t depends upon the current trading return R_t, but that U_{t-1} does not. Our strategy will be 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.[5]

D. 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 [38]. Denoting as before the trading system returns for period t (including transactions costs) as R_t, the Sharpe ratio is defined to be:

$$S_T = \frac{\text{Average}(R_t)}{\text{Standard Deviation}(R_t)} \tag{12}$$

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

where the average and standard deviation are estimated for periods $t = \{1, \ldots, T\}$. Note that for ease of exposition and analysis, we 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, we use Equation (12) for discussion purposes without loss of mathematical generality.[6]

Proper on-line learning requires that we compute the influence on the Sharpe ratio (marginal utility D_t) of the trading return R_t at time t. To accomplish this, we have derived a new objective function called the *differential Sharpe ratio* for on-line optimization of trading system performance [1], [2]. It is obtained by considering exponential moving averages of the returns and standard deviation of returns in (12), and expanding to first order in the adaptation rate η:

$$\begin{aligned} S_t|_{\eta>0} &\approx S_t|_{\eta=0} + \eta \frac{dS_t}{d\eta}|_{\eta=0} + O(\eta^2) \\ &= S_{t-1} + \eta \frac{dS_t}{d\eta}|_{\eta=0} + O(\eta^2) \ . \end{aligned} \tag{13}$$

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, we define the *differential Sharpe ratio* as:

$$D_t \equiv \frac{dS_t}{d\eta} = \frac{B_{t-1}\Delta A_t - \frac{1}{2}A_{t-1}\Delta B_t}{(B_{t-1} - A_{t-1}^2)^{3/2}} \ . \tag{14}$$

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

$$\begin{aligned} A_t &= A_{t-1} + \eta \Delta A_t = A_{t-1} + \eta(R_t - A_{t-1}) \\ B_t &= B_{t-1} + \eta \Delta B_t = B_{t-1} + \eta(R_t^2 - B_{t-1}) \ . \end{aligned} \tag{15}$$

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 (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 (31) in Section III) as the current contribution to the performance function U_t. Since S_{t-1} in Equation (13) does not depend on R_t, we have $dU_t/dR_t = dS_t/dR_t \approx \eta dD_t/dR_t$. When optimizing the trading system using Equation (14), the relevant derivatives have the simple form:

$$\frac{dD_t}{dR_t} = \frac{B_{t-1} - A_{t-1}R_t}{(B_{t-1} - A_{t-1}^2)^{3/2}} \ . \tag{16}$$

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

$$R_t^* = B_{t-1}/A_{t-1} \ . \tag{17}$$

Thus, the Sharpe ratio actually penalizes gains larger than R_t^*, which is counter-intuitive relative to most investors' notions of risk and reward.

E. 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 focussed on the mean-variance framework for portfolio optimization, he proposed the semi-variance as a means for dealing with downside returns [39]. 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 [40], [41], [42], [43], [44].

[6]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.

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:

$$\text{Sterling Ratio} = \frac{\text{Annualized Average Return}}{\text{Maximum Drawn-Down}} . \quad (18)$$

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 we focus on the Downside Deviation as a measure of downside risk in this paper.[7]

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

$$\text{DD}_T = \left(\frac{1}{T} \sum_{t=1}^{T} \min\{R_t, 0\}^2 \right)^{\frac{1}{2}} . \quad (19)$$

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

$$\text{DDR}_T = \frac{\text{Average}(R_t)}{\text{DD}_T} . \quad (20)$$

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 our recurrent reinforcement learning algorithm (Section III), we need to compute the influence of the return at time t on the DDR. In a similar manner to the development of the differential Sharpe ratio in [2], we define exponential moving averages of returns and of the squared Downside Deviation:

$$\begin{aligned} A_t &= A_{t-1} + \eta(R_t - A_{t-1}) \quad (21) \\ \text{DD}_t^2 &= \text{DD}_{t-1}^2 + \eta(\min\{R_t, 0\}^2 - \text{DD}_{t-1}^2) , \end{aligned}$$

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

$$\text{DDR}_t \approx \text{DDR}_{t-1} + \eta \frac{d\text{DDR}_t}{d\eta}|_{\eta=0} + O(\eta^2) . \quad (22)$$

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

$$\begin{aligned} D_t &\equiv \frac{d\text{DDR}_t}{d\eta} \\ &= \frac{R_t - \frac{1}{2}A_{t-1}}{\text{DD}_{t-1}} , R_t > 0 \quad (23) \\ &= \frac{\text{DD}_{t-1}^2 \cdot (R_t - \frac{1}{2}A_{t-1}) - \frac{1}{2}A_{t-1}R_t^2}{\text{DD}_{t-1}^3} , R_t \leq 0 \quad (24) \end{aligned}$$

[7]White has found that the Downside Deviation tracks the Sterling ratio effectively [45].

From Equation (24) 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 [22] for detailed experimental results on the use of the Downside Deviation Ratio to build RRL trading systems.

III. 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.

In [1], [2], we compared supervised learning to our 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 [46]. 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 search"), *Value Function* or *Actor-Critic* methods. The choice of the best method depends upon the nature of the problem domain. We will discuss this issue in greater detail in Section V. In this section, we present the Recurrent Reinforcement Learning algorithm for Direct Reinforcement and review value function based methods, specifically Q-Learning [10] and a refinement of Q-Learning called Advantage Updating [47]. In Section IV-C, we compare the RRL and value function methods for systems that learn to allocate assets between the S&P 500 stock index and T-Bills.

A. Recurrent Reinforcement Learning

In this section, we describe the Recurrent Reinforcement Learning algorithm for Direct Reinforcement. This algorithm was originally presented in [1] and [2].

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 (1) and trading returns of form (6) or (8), the gradient of U_T with respect to the parameters θ of the system after

a sequence of T periods is

$$\frac{dU_T(\theta)}{d\theta} = \sum_{t=1}^{T} \frac{dU_T}{dR_t} \left\{ \frac{dR_t}{dF_t}\frac{dF_t}{d\theta} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta} \right\} \quad (25)$$

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)

$$\Delta\theta = \rho \frac{dU_T(\theta)}{d\theta} \quad (26)$$

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) [48], [49]. The temporal dependencies in a sequence of decisions are accounted for through a recursive update equation for the parameter gradients:

$$\frac{dF_t}{d\theta} = \frac{\partial F_t}{\partial \theta} + \frac{\partial F_t}{\partial F_{t-1}}\frac{dF_{t-1}}{d\theta} \quad . \quad (27)$$

The above expressions (25) and (27) assume differentiability of F_t. For the long/short traders with thresholds described in Section II-A, the reinforcement signal can be backpropagated through the pre-thresholded outputs in a manner similar to the Adaline learning rule [50]. Equations (25), (26) and (27) 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). 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 (25) that depends on the most recently realized return R_t during a forward pass through the data:

$$\frac{dU_t(\theta)}{d\theta_t} \approx \frac{dU_t}{dR_t} \left\{ \frac{dR_t}{dF_t}\frac{dF_t}{d\theta_t} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta_{t-1}} \right\} \quad . \quad (28)$$

The parameters are then updated on-line using:

$$\Delta\theta_t = \rho \frac{dU_t(\theta_t)}{d\theta_t} \; . \quad (29)$$

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 (27) is:

$$\frac{dF_t}{d\theta_t} \approx \frac{\partial F_t}{\partial \theta_t} + \frac{\partial F_t}{\partial F_{t-1}}\frac{dF_{t-1}}{d\theta_{t-1}} \quad . \quad (30)$$

Equations (28), (29) and (30) 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) [51] and Dynamic Backpropagation [52]. See also the discussion of backpropagating utility in Werbos [53].

For differential performance criteria D_t described in Equation (11) of Section II-C (such as the differential Sharpe ratio (14) and differential Downside Deviation ratio (24)), the stochastic update equations (28) and (29) become:

$$\begin{aligned} \Delta\theta_t &= \rho \frac{dD_t(\theta_t)}{d\theta_t} \\ &\approx \rho \frac{dD_t}{dR_t} \left\{ \frac{dR_t}{dF_t}\frac{dF_t}{d\theta_t} + \frac{dR_t}{dF_{t-1}}\frac{dF_{t-1}}{d\theta_{t-1}} \right\} \quad . \end{aligned} \quad (31)$$

We use on-line algorithms of this recurrent reinforcement learning type in the simulations presented in Section IV. Note that we find that use of a noise variable ϵ_t provides little advantage for the real financial applications that we consider, since the data series contain significant intrinsic noise. Hence, we find that a simple "greedy" update is adequate.[8]

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 our formulation can be obtained for traders that maximize discounted future rewards. We have experimented with this approach, but found little advantage for the problems we consider. A second extension to the formulation is to consider a stochastic trader (Equation (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 (4). While the expected reward framework is appealing from a theoretical perspective, Equations (28), (29) and (30) presented above provide the practical basis for simulations.

Although we have focussed our discussion on traders of a single risky asset with scalar F_t, the algorithms described in this section can be trivially generalized to the vector case for portfolios. Optimization of portfolios is described in [1], [2].

B. Value Functions and Q-Learning

Besides explicitly training a trader to take actions, we can also implicitly learn correct actions through the technique of *value iteration*. Value iteration uses a *value function* to evaluate and improve policies (see [14] for a tutorial introduction and [16] 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.

[8]Tesauro finds a similar result for TD-Gammon [17], [18]. A "greedy" update works well, because the dice rolls in the game provided enough uncertainty to induce extensive strategy exploration.

The value function satisfies *Bellman's equation*

$$V^{\pi}(x) = \sum_{a} \pi(x,a) \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{\pi}(y) \right\} \quad , \quad (32)$$

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 (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:

$$V^{*}(x) = \max_{\pi} V^{\pi}(x) \quad , \quad (33)$$

and satisfies Bellman's optimality equation

$$V^{*}(x) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{*}(y) \right\} \quad . \quad (34)$$

The value iteration update:

$$V_{t+1}(x) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V_{t}(y) \right\} \quad , \quad (35)$$

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:

$$a^{*} = \arg\max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma V^{*}(y) \right\} \quad . \quad (36)$$

B.1 Q-Learning

The technique named Q-Learning [10] uses a value function which estimates future rewards based on both the current state and the current action taken. We can write the Q-function version of Bellman's optimality equation as

$$Q^{*}(x,a) = \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma \max_{b} Q^{*}(y,b) \right\} \quad . \quad (37)$$

Similarly to Equation (35), the Q-function can be learned using a value iteration approach:

$$Q_{t+1}(x,a) = \max_{a} \sum_{y} p_{xy}(a) \left\{ D(x,y,a) + \gamma Q_{t}(y) \right\} \quad . \quad (38)$$

This iteration has been shown [10] 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 (36) in order to choose the best action. One calculates the best action as

$$a^{*} = \arg\max_{a}(Q^{*}(x,a)) \quad . \quad (39)$$

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

$$\frac{1}{2}(D(x,y,a) + \gamma \max_{b} Q(y,b) - Q(x,a))^{2} \quad . \quad (40)$$

B.2 Advantage Updating

A refinement of the Q-Learning algorithm is provided by Advantage Updating [47]. 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:

$$Q^{*}(x,a) = V^{*}(x) + A^{*}(x,a) \quad . \quad (41)$$

Similarly to Q-Learning, the optimal action to take in state x is found by $a^{*} = \arg\max_{a}(A^{*}(x,a))$. See Baird [47] for a description of the learning algorithms.

IV. Empirical Results

This section presents empirical results for three problems. First, controlled experiments using artificial price series are done to test the RRL algorithm's ability to learn profitable trading strategies, to maximize risk adjusted return (as measured by the Sharpe 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. Finally, we 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, we 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 V-D.

A. Trader Simulation

In this section we demonstrate the use of the RRL algorithm to optimize trading behavior using the differential Sharpe Ratio (Equation (14)) in the presence of transaction costs. More extensive results are presented in [2]. There, we 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 II-A. 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.

A.1 Data

We generate log price series as random walks with autoregressive trend processes. The two parameter model is thus:

$$p(t) = p(t-1) + \beta(t-1) + k\,\epsilon(t) \quad (42)$$

$$\beta(t) = \alpha\,\beta(t-1) + \nu(t)\ , \quad (43)$$

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

$$z(t) = \exp\left(\frac{p(t)}{R}\right) \quad (44)$$

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.[9]

For the results we present here, we 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 2.

A.2 Simulated Trading Results

Figures 2, 3 and 4 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 (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 2 and in the distribution of differential Sharpe ratios in the lower left panel of Figure 4.

Figure 5 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. We 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.

[9]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.

B. US Dollar/British Pound Foreign Exchange Trading System

A {long, short, neutral} trading system is trained on half-hourly US Dollar / British Pound foreign exchange (FX) rate data. The experiments described in this section were first reported in [22]. 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.[10] 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 (24), a measure of risk-adjusted return.

The top panel in Figure 6 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 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 6 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 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.

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

C. S&P 500 / T-Bill Asset Allocation

In this section we 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 [21]. 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. We use the Advantage Updating refinement instead of the standard Q-Learning algorithm, because we found it to yield better trading results. See Section III-B.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 7 along with the 3-month Treasury Bill and S&P 500 dividend yields. The 84 monthly input series used in the trading systems include both financial and macroeconomic data. All data are obtained from Citibase,[11] 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.

C.1 Simulation Details

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 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 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. The networks are trained initially with the γ discounting factor set to 0. Then γ is set to 0.75. We find decreasing performance when the value of γ is adjusted to higher values.

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

To investigate the bias / variance tradeoff for the Q-Traders, we 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).

C.2 S&P Experimental Results

Figure 8 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[12]. 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. 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.[13] Remarkably, the superior results for the RRL-Trader are based on networks with a single thresholded *tanh* unit, while those for the Q-Trader required networks with 30 hidden *tanh* units.[14]

Figure 9 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. We can see that 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 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 we find that both trading systems outperform the buy and hold strategy, as measured by both accumulated wealth and Sharpe ratio. These differences are

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

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

[14] As discussed in the Section IV-C.1, care was taken to avoid both underfitting and overfitting in the Q-Trader case, and smaller nets performed substantially worse.

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 [2]. Further discussion of the Q-Trader vs. RRL-Trader performance is presented in Section V-D.

C.3 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 10 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:

$$S_i = \frac{\left|\frac{dF}{dx_i}\right|}{\max_j \left|\frac{dF}{dx_j}\right|} \quad , \tag{45}$$

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

The time-varying sensitivities in Figure 10 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.

V. Learn the Policy or Learn the Value?

As mentioned in Section III, 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.

A. Immediate vs. Future Rewards

Reinforcement signals received from the environment can be *immediate* or *delayed*. In some problems, such as checkers [54], [55], backgammon [17], [18], navigating a maze[56], or maneuvering around obstacles[57], reinforcement from the environment occurs only at the end of the game or task. The final rewards received are {success, failure} or {win, lose}. For such tasks, 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 [8], TD-Learning [9] and Q-Learning [10], [11].

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 [12]. 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.

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.[15]

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 [58], [59] and Baxter & Bartlett [7][16] show convergence to locally optimal policies by using simulation based methodologies to approximate expected rewards. Sutton et al [60] and Konda & Tsitsiklis [61] obtain similar results when estimating expected rewards from a value function implemented using a function approximator. An application to robot navigation is provided by Grudic and Ungar [62]. 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.

B. 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.

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

[16] 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. Our intended usage of the term 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.

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 [63]. 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.[17]

We find our 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.

C. An Example

We present an example of how an increase in complexity occurs when a policy is represented implicitly through the use of a value function. We 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, we 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. We 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 we decide to learn the value function before taking actions, we find in this case that we have to learn the XOR function. As shown in Figure 11, 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, we can solve this problem 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 we make the problem 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, if we wish to approximate the value function to an arbitrarily small precision, we must use an arbitrarily large model.

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

For the S&P 500 / T-Bill asset allocation problem described in Section IV-C, we 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, we 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 IV-C.3 was easy to formulate and implement. It enables us to identify 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 we 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.[18] Portfolio management requires a continuous weight for each of N

[17] Brown [64] provides a nice example that demonstrates the brittleness of Q-Learners in noisy environments.

[18] We have encountered this obstacle in preliminary, unpublished experiments.

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, we can only consider discrete action sets. The number of discrete actions that must be considered is exponential in N. As another issue, we must also consider 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.

VI. Conclusions

In this paper, we have demonstrated how to train trading systems via Direct Reinforcement. We have described the Recurrent Reinforcement Learning (RRL) algorithm, and used it to optimize financial performance criteria such as the *differential Sharpe ratio* and *differential Downside Deviation ratio*. We have also provided empirical results that demonstrate the presence of predictability as discovered by RRL in intradaily US Dollar/British Pound exchange rates and in the monthly S&P 500 Stock Index for the 25 year test period 1970 through 1994.

In previous work [1], [2], we showed that trading systems trained via RRL significantly outperform systems trained using supervised methods. In this paper, we have compared the Direct Reinforcement approach using RRL to the Q-Learning Value Function method. We find that an RRL-Trader achieves better performance than a Q-Trader for the S&P 500 / T-Bill asset allocation problem. We observe that relative to Q-Learning, RRL enables a simpler problem representation, avoids Bellman's curse of dimensionality and offers compelling advantages in efficiency.

We have also discussed the relative merits of Direct Reinforcement and Value Function learning, and provided arguments and examples for why value function based methods may result in unnatural problem representations. For problem domains where immediate estimates of incremental performance can be obtained, our results suggest that Direct Reinforcement offers a powerful alternative.

Acknowledgments

The authors wish to thank Lizhong Wu and Yuansong Liao for their contributions to our early work on Direct Reinforcement, and Amir Atiya, Gerry Tesauro and the reviewers for their helpful comments on this manuscript. We gratefully acknowledge support for this work from Nonlinear Prediction Systems and from DARPA under contract DAAH01-96-C-R026 and AASERT grant DAAH04-95-1-0485.

References

[1] John Moody and Lizhong Wu, "Optimization of trading systems and portfolios," in *Decision Technologies for Financial Engineering*, Y. Abu-Mostafa, A. N. Refenes, and A. S. Weigend, Eds., London, 1997, pp. 23–35, World Scientific, This is a slightly revised version of the original paper that appeared in the NNCM*96 Conference Record, published by Caltech, Pasadena, 1996.

[2] John Moody, Lizhong Wu, Yuansong Liao, and Matthew Saffell, "Performance functions and reinforcement learning for trading systems and portfolios," *Journal of Forecasting*, vol. 17, pp. 441–470, 1998.

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

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

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

[6] Ronald J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Machine Learning*, vol. 8, pp. 229–256, 1992.

[7] Jonathan Baxter and Peter L. Bartlett, "Direct gradient-based reinforcement learning: I. Gradient estimation algorithms," Tech. Rep., Computer Sciences Laboratory, Australian National University, 1999.

[8] Richard E. Bellman, *Dynamic Programming*, Princeton University Press, Princeton, NJ, 1957.

[9] Richard S. Sutton, "Learning to predict by the method of temporal differences," *Machine Learning*, vol. 3, no. 1, pp. 9–44, 1988.

[10] C. J. C. H. Watkins, *Learning with Delayed Rewards*, Ph.D. thesis, Cambridge University, Psychology Department, 1989.

[11] C. J. Watkins and P. Dayan, "Technical note: Q-Learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[12] Andrew G. Barto, Richard S. Sutton, and Charles W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 13, no. 5, pp. 835–846, September 1983.

[13] Andrew G. Barto, *Handbook of Intelligent Control*, chapter 12, pp. 469–492, Van Nostrand Reinhold, New York, 1992.

[14] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, 1996.

[15] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, 1996.

[16] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning: An Introduction*, MIT Press, Cambridge, MA, 1997.

[17] G. Tesauro, "TD-Gammon, a self-teaching backgammon program, achieves master-level play," *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994.

[18] G. Tesauro, "Temporal difference learning and TD-Gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995.

[19] Robert H. Crites and Andrew G. Barto, "Improving elevator performance using reinforcement learning," in *Advances in Neural Information Processing Systems*, David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, Eds., 1996, vol. 8, pp. 1017–1023.

[20] Wei Zhang and Thomas G. Dietterich, "High-performance job-shop scheduling with a time-delay TD(λ) network," in *Advances in Neural Information Processing Systems*, David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, Eds., 1996, vol. 8, pp. 1024–1030.

[21] John Moody and Matthew Saffell, "Reinforcement learning for trading," in *Advances in Neural Information Processing Systems*, Sara A. Solla Michael S. Kearns and David A. Cohn, Eds. 1999, vol. 11, pp. 917–923, MIT Press.

[22] John Moody and Matthew Saffell, "Minimizing downside risk via stochastic dynamic programming," in *Computational Finance 1999*, Andrew W. Lo Yaser S. Abu-Mostafa, Blake LeBaron and Andreas S. Weigend, Eds. 2000, pp. 403–415, MIT Press.

[23] E. J. Elton and M. J. Gruber, "Dynamic programming applications in finance," *Journal of Finance*, vol. 26, no. 2, 1971.

[24] R. C. Merton, "Lifetime portfolio selection under uncertainty: The continuous-time case," *Review of Economics and Statistics*, vol. 51, pp. 247–257, August 1969.

[25] R. C. Merton, "Optimum consumption and portfolio rules in a continuous-time model," *Journal of Economic Theory*, vol. 3, pp. 373–413, December 1971.

[26] Robert C. Merton, *Continuous-Time Finance*, Blackwell Publisher Inc, 1990.

[27] Douglas T. Breeden, "Intertemporal portfolio theory and asset pricing," in *Finance*, John Eatwell, Murray Milgate, and Peter Newman, Eds., pp. 180–193. The New Palgrave, Macmillan Press, New York, 1987.

[28] Darrell Duffie, *Security Markets: Stochastic Models*, Academic Press, 1988.

[29] Darrell Duffie, *Dynamic Asset Pricing Theory*, Princeton University Press, 2 edition, 1996.

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

[31] M. J. Brennan, E. S. Schwartz, and R. Lagnado, "Strategic asset allocation," *Journal of Economic Dynamics and Control*, vol. 21, pp. 1377–1403, 1997.

[32] F. Longstaff and E. Schwartz, "Valuing American options by simulation: A simple least squares approach," *Review of Financial Studies*, 2001, To appear.

[33] Benjamin Van Roy, "Temporal-difference learning and applications in finance," in *Computational Finance 1999*, Yaser S. Abu-Mostafa, Blake LeBaron, Andrew W. Lo, and Andreas S. Weigend, Eds. 2001, pp. 447–461, MIT Press.

[34] Ralph Neuneier, "Optimal asset allocation using adaptive dynamic programming," in *Advances in Neural Information Processing Systems*, David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, Eds. 1996, vol. 8, pp. 952–958, MIT Press.

[35] Ralph Neuneier and Oliver Mihatsch, "Risk sensitive reinforcement learning," in *Advances in Neural Information Processing Systems*, Michael S. Kearns, Sara A. Solla, and David A. Cohn, Eds. 1999, vol. 11, pp. 1031–1037, MIT Press.

[36] J. N. Tsitsiklis and B. Van Roy, "Optimal stopping of Markov processes: Hilbert space theory, approximation algorithms, and an application to pricing high-dimensional financial derivatives," *IEEE Transactions on Automatic Control*, vol. 44, no. 10, pp. 1840–1851, October 1999.

[37] J. N. Tsitsiklis and B. Van Roy, "Regression methods for pricing complex American-style options," *IEEE Transactions on Neural Networks*, vol. 12, no. 4, July 2001, This issue.

[38] William F. Sharpe, "Mutual fund performance," *Journal of Business*, pp. 119–138, Jan 1966.

[39] H.M. Markowitz, *Portfolio Selection: Efficient Diversification of Investments*, New York: Wiley, 1959.

[40] F.A. Sortino and R. van der Meer, "Downside risk – capturing what's at stake in investment situations," *The Journal of Portfolio Management*, vol. 17, pp. 27–31, 1991.

[41] D. Nawrocki, "Optimal algorithms and lower partial moment: Ex post results," *Applied Economics*, vol. 23, pp. 465–470, 1991.

[42] D. Nawrocki, "The characteristics of portfolios selected by n-degree lower partial moment," *International Review of Financial Analysis*, vol. 1, pp. 195–209, 1992.

[43] F.A. Sortino and H.J. Forsey, "On the use and misuse of downside risk," *The Journal of Portfolio Management*, vol. 22, pp. 35–42, 1996.

[44] D. Nawrocki, "A brief history of downside risk measures," *Journal of Investing*, pp. 9–26, Fall 1999.

[45] Halbert White, "Personal communication.," Unpublished., 1996.

[46] Richard S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. thesis, University of Massachusetts, Amherst, 1984.

[47] Leemon C. Baird, "Advantage updating," Tech. Rep. WL-TR-93-1146, Wright Laboratory, Wright-Patterson Air Force Base, OH 45433-7301, 1993.

[48] D.E. Rumelhart, G.E. Hinton, and R.J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing: Exploration in the microstructure of cognition*, D.E. Rumelhart and J.L. McClelland, Eds., chapter 8, pp. 319–362. MIT Press, Cambridge, MA, 1986.

[49] P.J. Werbos, "Back-propagation through time: What it does and how to do it," *IEEE Proceedings*, vol. 78, no. 10, pp. 1550–1560, Oct. 1990.

[50] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *IRE WESCON Convention Record*, 1960, pp. 96–104.

[51] Ronald J. Williams and David Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural Computation*, vol. 1, pp. 270–280, 1989.

[52] Kumpati S. Narendra and Kannan Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Transactions on Neural Networks*, vol. 1, no. 1, pp. 4–27, 1990.

[53] P.J. Werbos, "Neurocontrol and supervised learning: An overview and evaluation," in *Handbook of Intelligent Control*, David A. White and Donald A. Sofge, Eds., chapter 3, pp. 65–90. Van Nostrand Reinhold, New York, 1992.

[54] A. L. Samuel, "Some studies in machine learning using the game of checkers," *IBM Journal on Research and Development*, vol. 3, pp. 211–229, 1959.

[55] A. L. Samuel, "Some studies in machine learning using the game of checkers. II – Recent progress," *IBM Journal on Research and Development*, vol. 11, pp. 601–617, 1967.

[56] J. Peng and R. J. Williams, "Efficient learning and planning within the Dyna framework," *Adaptive Behavior*, vol. 1, no. 4, pp. 437–454, 1993.

[57] A. W. Moore and C. G. Atkeson, "Prioritized sweeping: Reinforcement learning with less data and less real time," *Machine Learning*, vol. 13, pp. 103–130, 1993.

[58] Peter Marbach and John N. Tsitsiklis, "Simulation-based optimization of Markov reward processes," in *IEEE Conference on Decision and Control*, 1998.

[59] Peter Marbach and John N. Tsitsiklis, "Simulation-based optimization of Markov reward processes," *IEEE Transactions on Automatic Control*, 2001, To appear.

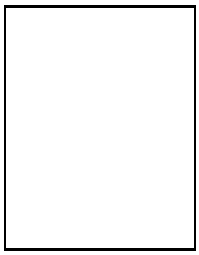
[60] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour, "Policy gradient methods for reinforcement learning with function approximation," in *Advances in Neural Information Processing Systems*, Todd K. Leen Sara A. Solla and Klaus-Robert Muller, Eds. 2000, vol. 12, pp. 1057–1063, MIT Press.

[61] Vijay R. Konda and John N. Tsitsiklis, "Actor-critic algorithms," in *Advances in Neural Information Processing Systems*, Sara A. Solla, Todd K. Leen, and Klaus-Robert Muller, Eds. 2000, vol. 12, pp. 1008–1014, MIT Press.

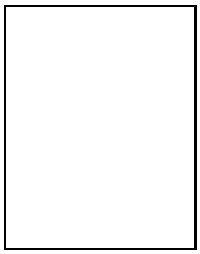
[62] Gregory Z. Grudic and Lyle H. Ungar, "Localizing policy gradient estimates to action transitions," in *Seventeenth International Conference on Machine Learning*, 2000.

[63] Leemon Baird and Andrew Moore, "Gradient descent for general reinforcement learning," in *Advances in Neural Information Processing Systems*, Sara A. Solla Michael S. Kearns and David A. Cohn, Eds. 1999, vol. 11, pp. 968–974, MIT Press.

[64] Timothy X. Brown, "Policy vs. value function learning with variable discount factors," Talk presented at the NIPS 2000 Workshop entitled "Reinforcement Learning: Learn the Policy or Learn the Value Function?", December 2000.



John Moody is the Director of the Computational Finance Program and a Professor of Computer Science and Electrical Engineering at Oregon Graduate Institute of Science and Technology. He is also the Founder and President of Nonlinear Prediction Systems, a company specializing in the development of forecasting and trading systems. His research interests include computational finance, time series analysis and machine learning. Moody recently served as Program Co-Chair for Computational Finance 2000 in London, and is a past General Chair and Program Chair of the Neural Information Processing Systems (NIPS) conference. He received his B.A in Physics from the University of Chicago (1979) and his M.A. and Ph.D. in Theoretical Physics from Princeton University (1981 and 1984).



Matthew Saffell is a Ph.D. candidate in the Computer Science and Engineering Department at the Oregon Graduate Institute and a Consulting Scientist at Nonlinear Prediction Systems. He received his B.Sc. in Computer Science and Engineering with a minor in Mathematics from LeTourneau University in 1992, and his M.Sc. in Computer Science and Engineering from the University of Tennessee in 1994.

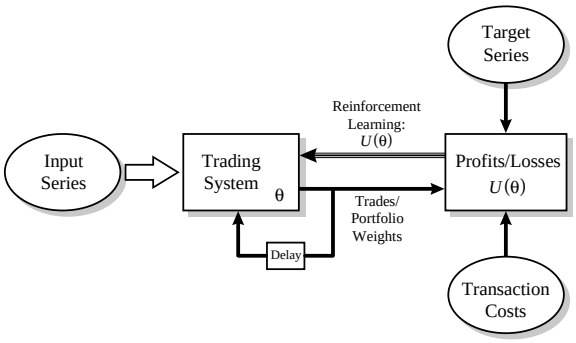


Fig. 1. A trading system based on Direct Reinforcement, the approach taken in this paper. The system trades the target series, making trading decisions based upon a set of input variables and the current positions held. No intermediate steps such as making forecasts or labelling desired trades are required, and it is not necessary to learn a value function. The trader learns a strategy via *trial and error* exploration, taking actions and receiving positive or negative reinforcement based on the results. A trading performance function $U(\theta)$, such as profit, utility or risk-adjusted return, is hence used to *directly optimize* the trading system parameters θ. 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. For the traders considered in this paper, the *Direct Reinforcement* (policy search) method *Recurrent Reinforcement Learning* is used to optimize the trader.

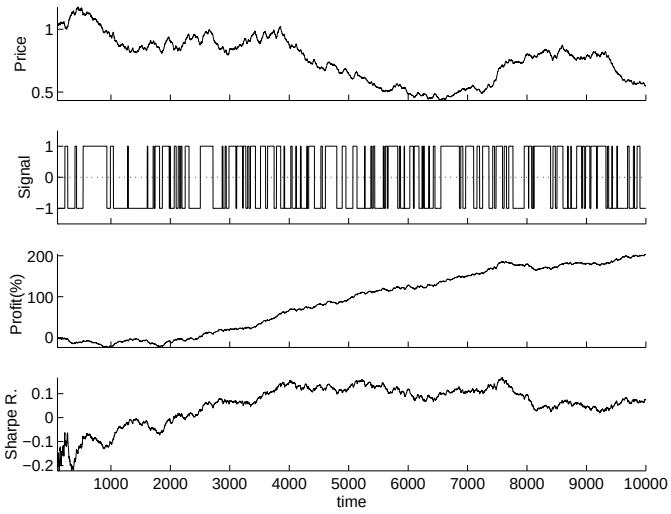


Fig. 2. Artificial prices (top panel), trading signals (second panel), cumulative sums of profits (third panel) and the moving average Sharpe ratio with $\eta = 0.01$ (bottom panel). The system performs poorly while learning from scratch during the first 2000 time periods, but its performance remains good thereafter.

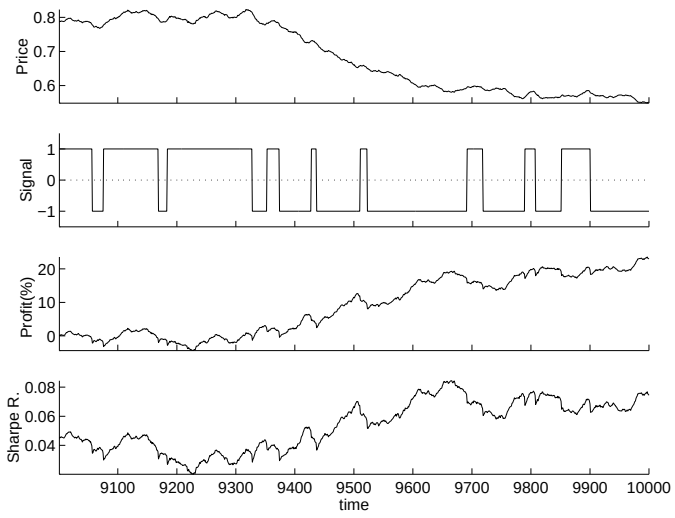


Fig. 3. An expanded view of the last thousand time periods of Figure 2. The exponential moving Sharpe ratio has a forgetting time scale of $1/\eta = 100$ periods. A smaller η would smooth the fluctuations out.

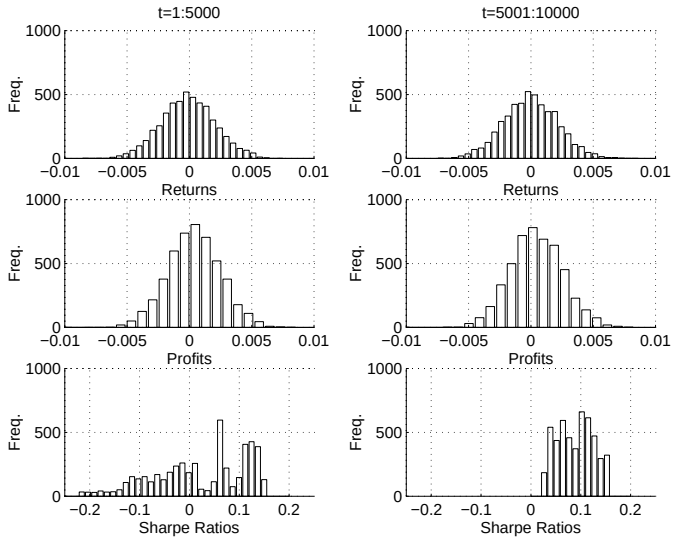


Fig. 4. Histograms of the price changes (top), trading profits per time period (middle) and Sharpe ratios (bottom) for the simulation shown in Figure 2. The left column is for the first 5,000 time periods, and the right column is for the last 5,000 time periods. The transient effects during the first 2000 time periods for the real-time recurrent learning are evident in the lower left graph.

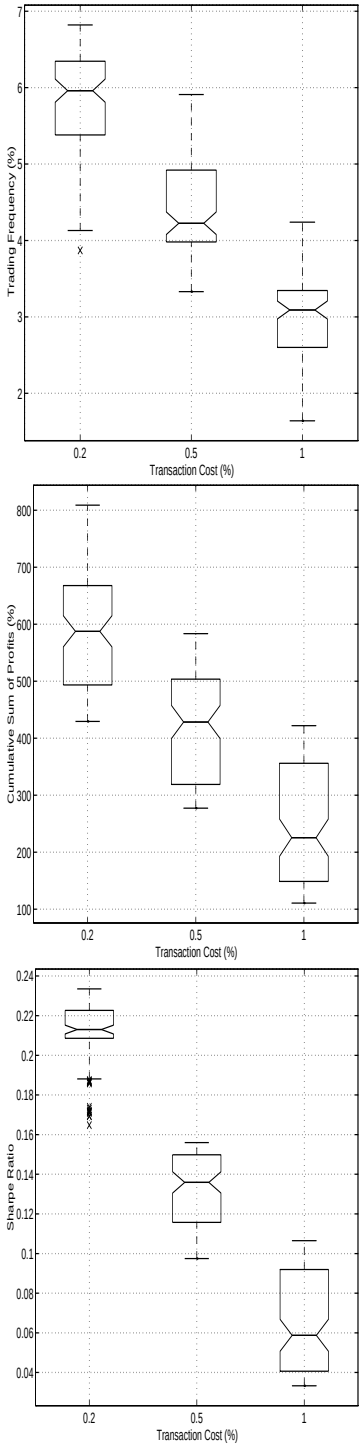


Fig. 5. Boxplots of trading frequency, cumulative sums of profits and Sharpe ratios vs. transaction costs. The results are obtained over 100 trials with various realizations of artificial data and initial system parameters. Increased transaction costs reduce trading frequency, profits and Sharpe ratio, 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.

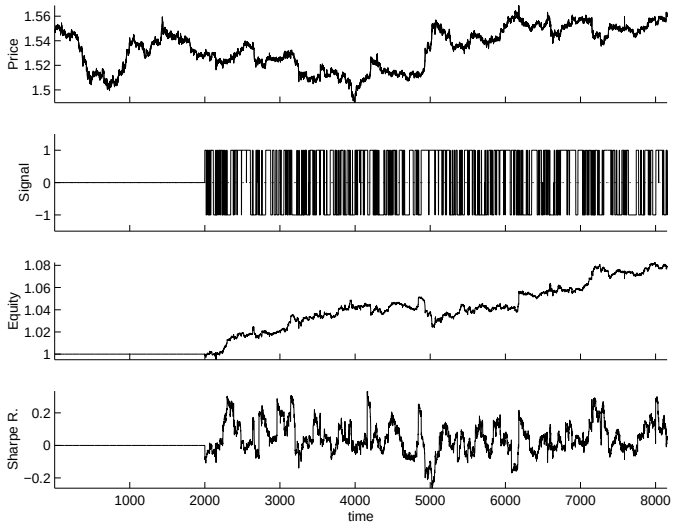


Fig. 6. {Long, short, neutral} trading system of the US Dollar/British Pound that uses the bid/ask spread as transaction costs. The data consists of half-hourly quotes for the 5 day per week, 24 hour interbank FX market. The time period shown is the first 8 months of 1996. The trader is optimized via 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.

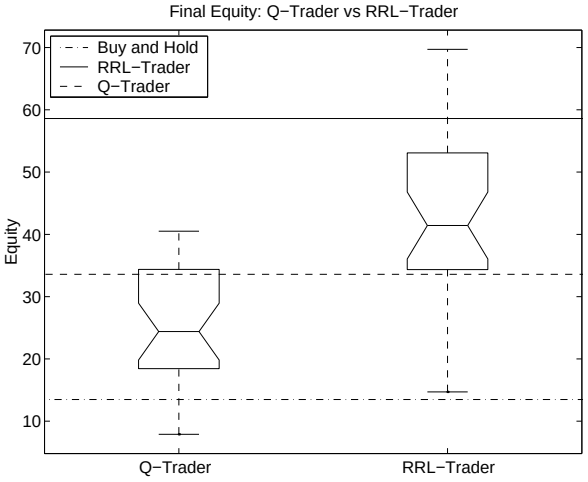


Fig. 8. Test results for ensembles of simulations using the S&P 500 stock index and 3-month Treasury Bill data over the 1970-1994 time period. The boxplots show the performance for the ensembles of RRL-Trader and Q-Trader trading systems. The horizontal lines indicate the performance of the systems and 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.

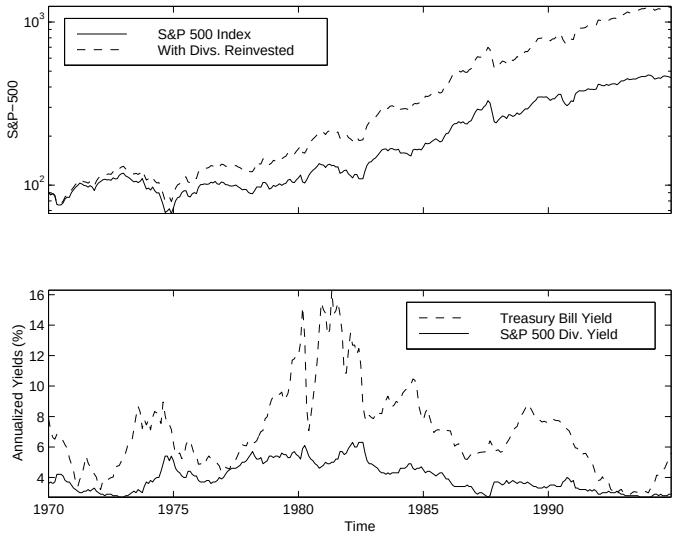


Fig. 7. Time series that influence the return attainable by the S&P 500 / TBill asset allocation system. The top panel shows the S&P 500 series with and without dividends reinvested. The bottom panel shows the annualized monthly Treasury Bill and S&P 500 dividend yields.

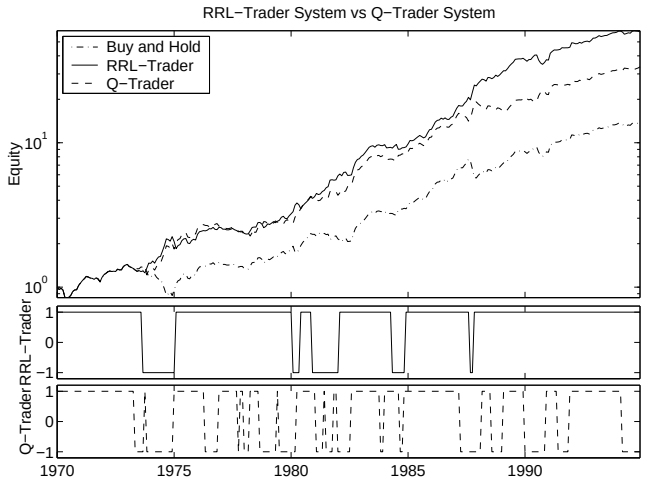


Fig. 9. Test results for ensembles of simulations using the S&P 500 stock index and 3-month Treasury Bill data over the 1970-1994 time period. Shown are the equity curves associated with the systems and the buy and hold strategy, as well as the trading signals produced by the systems. The solid curves 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.

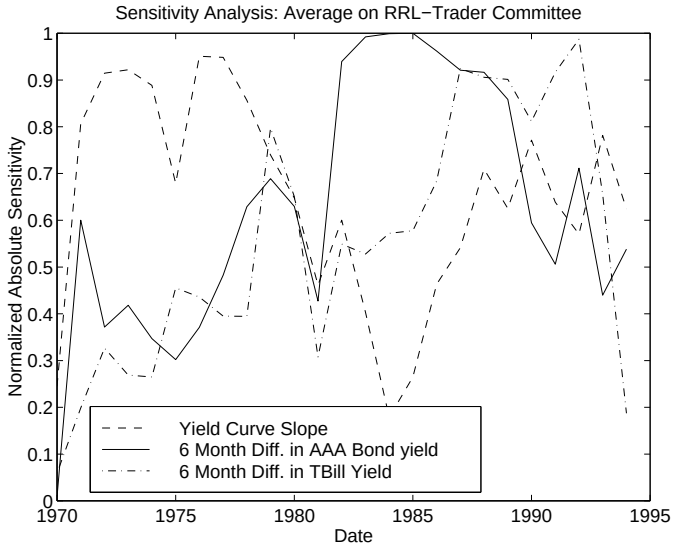


Fig. 10. Sensitivity traces for three of the inputs to the RRL-Trader trading system averaged over the ensemble of traders. The non-stationary relationships typical among economic variables is evident from the time-varying sensitivities.

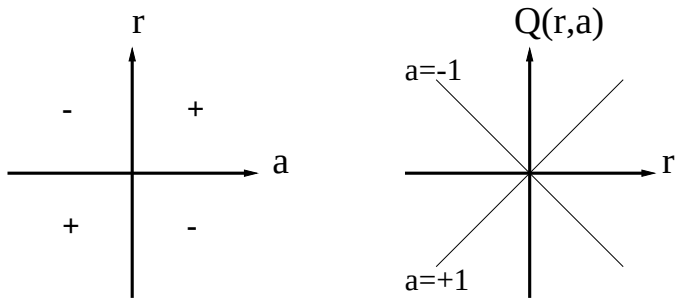


Fig. 11. A representation of the value function to be learned by the Q-Learning algorithm for the example given in the text (Section V). The function represents the Q-value, $Q(r, a)$, which is the value from taking action "a" in state "r". The figure on the left shows the value function for the case 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, $a = \text{sign}(r)$.