Stochastic Direct Reinforcement: Application to Simple Games with Recurrence

John Moody*, Yufeng Liu*, Matthew Saffell* and Kyoungju Youn†
{moody, yufeng, saffell}@icsi.berkeley.edu, kjyoun@yahoo.com
*International Computer Science Institute, Berkeley
†OGI School of Science and Engineering, Portland

Abstract

We investigate repeated matrix games with stochastic players as a microcosm for studying dynamic, multi-agent interactions using the Stochastic Direct Reinforcement (SDR) policy gradient algorithm. SDR is a generalization of Recurrent Reinforcement Learning (RRL) that supports stochastic policies. Unlike other RL algorithms, SDR and RRL use *recurrent policy gradients* to properly address temporal credit assignment resulting from recurrent structure. Our main goals in this paper are to (1) distinguish *recurrent memory* from *standard, non-recurrent memory* for policy gradient RL, (2) compare SDR with Q-type learning methods for simple games, (3) distinguish *reactive* from *endogenous* dynamical agent behavior and (4) explore the use of recurrent learning for interacting, dynamic agents.

We find that SDR players learn much faster and hence outperform recently-proposed Q-type learners for the simple game Rock, Paper, Scissors (RPS). With more complex, dynamic SDR players and opponents, we demonstrate that recurrent representations and SDR's recurrent policy gradients yield better performance than non-recurrent players. For the Iterated Prisoners Dilemma, we show that non-recurrent SDR agents learn only to defect (Nash equilibrium), while SDR agents with recurrent gradients can learn a variety of interesting behaviors, including cooperation.

1 Introduction

We are interested in the behavior of reinforcement learning agents when they are allowed to learn dynamic policies and operate in complex, dynamic environments. A general challenge for research in multi-agent systems is for agents to learn to leverage or exploit the dynamic properties of the other agents and the overall environment. Even seemingly simple repeated matrix games can display complex behaviors when the policies are reactive, predictive or endogenously dynamic in nature.

In this paper we use repeated matrix games with stochastic players as a microcosm for studying dynamic, multi-agent interactions using the recently-proposed Stochastic Direct Reinforcement (SDR) policy gradient algorithm. Within this context, we distinguish *recurrent memory* from *standard, non-recurrent memory* for policy gradient RL (Section 2), compare policy gradient with Q-type learning methods for simple games (Section 3), contrast *reactive* with *non-reactive, endogenous* dynamical agent behavior and explore the use of recurrent learning for addressing temporal credit assignment with interacting, dynamic agents (Section 4).

Most studies of RL agents in matrix games use Q-Learning, (Littman 1994; Sandholm & Crites 1995; Claus & Boutilier 1998; Hu & Wellman 1998; Tesauro & Kephart 2002; Bowling & Veloso 2002; Tesauro 2004). These methods based on Markov Decision Processes (MDPs) learn a Q-value for each state-action pair, and represent policies only implicitly. Though theoretically appealing, Q-Learning can not easily be scaled up to the large state or action spaces which often occur in practice.

Direct reinforcement (DR) methods (*policy gradient* and *policy search*) (Williams 1992)(Moody & Wu 1997)(Moody *et al.* 1998)(Baxter & Bartlett 2001) (Ng & Jordan 2000) represent policies explicitly and do not require that a value function be learned. Policy gradient methods seek to improve the policy by using the gradient of the expected average or discounted reward with respect to the policy function parameters. More generally, they seek to maximize an agent's utility, which is a function of its rewards. We find that direct representation of policies allows us handle mixed strategies for matrix games naturally and efficiently.

Most previous work with RL agents in repeated matrix games has used static, memory-less opponents, meaning that policy inputs or state-spaces for Q-tables do not encode previous actions. (One exception is the work of (Sandholm & Crites 1995).) In order to exhibit interesting temporal behaviors, a player with a fixed policy must have memory. We refer to such players as *dynamic contestants*. We note the following properties of the repeated matrix games and players we consider:

1. Known, stationary reward structure.
2. Simultaneous play with immediate rewards.
3. Mixed policies: stochastic actions.
4. Partial-observability: players do not know each others' strategies or internal states.
5. Standard Memory: dynamic players may have policies that use memory of their opponents' actions, which is represented by non-recurrent inputs or state variables.

6. Recurrent Memory: dynamic players may have policies that depend upon inputs or state variables representing their own previous actions.

While the use of standard memory in Q-type RL agents has been well studied (Lin & Mitchell 1992; McCallum 1995) and applied in other areas of AI, the importance of distinguishing between *standard memory* and *recurrent memory* within a policy gradient context has not previously been recognized.

With dynamic contestants, the game dynamics are recurrent. Dynamic contestants can be reactive or predictive of opponents' actions and can also generate endogenous dynamic behavior by using their own past actions as inputs (recurrence). When faced with a predictive opponent, a player must have recurrent inputs in order to predict the predictive responses of its opponent. The player must know what its opponent knows in order to anticipate the response and thereby achieve best play.

The feedback due to recurrent inputs gives rise to learning challenges that are naturally solved within a non-Markovian or recurrent learning framework. The SDR algorithm described in Section 2 is well-matched for use in games with dynamic, stochastic opponents of the kinds we investigate. In particular, SDR uses *recurrent policy gradients* to properly address temporal credit assignment that results from recurrence in games.

Section 3 provides a baseline comparison to traditional value function RL methods by pitting SDR players against two recently-proposed Q-type learners for the simple game Rock, Paper, Scissors (RPS). In these competitions, SDR players learn much faster and thus out-perform their Q-type opponents. The results and analysis of this section raise questions as to whether use of a Q-table is desirable for simple games.

Section 4 considers games with more complex SDR players and opponents. We demonstrate that recurrence gives rise to predictive abilities and dynamical behaviors. Recurrent representations and SDR's recurrent policy gradients are shown to be yield better performance than non-recurrent SDR players when competing against dynamic opponents in the zero sum games. Case studies include Matching Pennies and RPS.

For the Iterated Prisoners Dilemma (a general sum game), we consider SDR self-play and play against fixed opponents. We find that non-recurrent SDR agents learn only the Nash equilibrium strategy "always defect", while SDR agents trained with recurrent gradients can learn a variety of interesting behaviors. These include the Pareto-optimal strategy "always cooperate", a time-varying strategy that exploits a generous opponent and the evolutionary stable strategy Tit-for-Tat.

2 Stochastic Direct Reinforcement

We summarize the Stochastic Direct Reinforcement (SDR) algorithm, which generalizes the recurrent reinforcement learning (RRL) algorithm of (Moody & Wu 1997; Moody *et al.* 1998; Moody & Saffell 2001) and the policy gradient work of (Baxter & Bartlett 2001). This stochastic policy gradient algorithm is formulated for probabilistic actions, partially-observed states and non-Markovian policies. With SDR, an agent represents actions explicitly and learns them directly without needing to learn a value function. Since SDR agents represent policies directly, they can naturally incorporate recurrent structure that is intrinsic to many potential applications. Via use of recurrence, the short term effects of an agent's actions on its environment can be captured, leading to the possibility of discovering more effective policies.

Here, we describe a simplified formulation of SDR appropriate for the repeated matrix games with stochastic players that we study. This formulation is sufficient to (1) compare SDR to recent work on Q-Learning in games and (2) investigate recurrent representations in dynamic settings. A more general and complete presentation of SDR will be presented elsewhere.

In the following subsections, we provide an overview of SDR for repeated, stochastic matrix games, derive the batch version of SDR with recurrent gradients, present the stochastic gradient approximation to SDR for more efficient and online learning, describe the model structures used for SDR agents in this paper and relate recurrent representations to more conventional "memory-based" approaches.

2.1 SDR Overview

Consider an agent that takes action a_t with probability $p(a_t)$ given by its stochastic policy function P

$$p(a_t) = P(a_t; \theta, \ldots) \ , \tag{1}$$

The goal of the SDR learning algorithm is to find parameters θ for the best policy according to an agent's *expected utility* U_T accumulated over T time steps. For the time being, let us assume that the agent's policy (e.g. game strategy) and the environment (e.g. game rules, game reward structure, opponent's strategy) are stationary. The expected total utility of a sequence of T actions can then be written in terms of *marginal utilities* $u_t(a_t)$ gained at each time:

$$U_T = \sum_{t=1}^{T} \sum_{a_t} u_t(a_t) p(a_t). \tag{2}$$

This simple additive utility is a special case of more general path-dependent utilities as described in (Moody *et al.* 1998; Moody & Saffell 2001), but is adequate for studying learning in repeated matrix games.

More specifically, a recurrent SDR agent has a stochastic policy function P

$$p(a_t) = P(a_t; \theta, I_{t-1}^{(n)}, A_{t-1}^{(m)}). \tag{3}$$

Here $A_{t-1}^{(m)} = \{a_{t-1}, a_{t-2} ..., a_{t-m}\}$ is a partial history of m recent actions (recurrent inputs), and the external *information set* $I_{t-1}^{(n)} = \{i_{t-1}, i_{t-2} ..., i_{t-n}\}$ represents n past observation vectors (non-recurrent, memory inputs) available to the agent at time t. Such an agent has *recurrent memory of order* m and *standard memory of length* n As described in Section 4, such an agent is said to have *dynamics of order* (m, n).

We refer to the combined action and information sets as the *observed history*. Note that the observed history is not assumed to provide full knowledge of the state S_t of the world. While probabilities for previous actions $A_{t-1}^{(m)}$ depend explicitly on θ, we assume here that components of $I_{t-1}^{(n)}$ have *no known* direct θ dependence, although there may be unknown dependencies of $I_{t-1}^{(n)}$ upon recent actions $A_{t-1}^{(m)}$ [1]. For lagged inputs, it is the explicit dependence on policy parameters θ that distinguishes *recurrence* from *standard memory*. Within a policy gradient framework, this distinction is important for properly addressing the *temporal credit assignment problem*.

The matrix games we consider have unknown opponents, and are hence partially-observable. The SDR agents in this paper have knowledge of their own policy parameters θ, the history of play, or the pay-off structure of the game u. However, they have no knowledge of the opponent's strategy or internal state. Denoting the opponent's action at time t as o_t, a partial history of n opponent plays is $I_{t-1}^{(n)} = \{o_{t-1}, o_{t-2}..., o_{t-n}\}$, which are treated as non-recurrent, memory inputs. The marginal utility or pay-off at time t depends only upon the most recent actions of the player and its opponent: $u_t(a_t) = u(a_t, o_t)$. For the game simulations in this paper, we consider SDR players and opponents with $(m, n) \in \{0, 1, 2\}$.

For many applications, the marginal utilities of all possible actions $u_t(a_t)$ are known after an action is taken. When trading financial markets for example, the profits of all possible one month trades are known at the end of the month.[2] For repeated matrix games, the complete reward structure is known to the players, and they act simultaneously at each game iteration. Once the opponent's action is observed, the SDR agent knows the marginal utility for each possible choice of its prior action. Hence, the SDR player can compute the second sum in Eq.(2). For other situations where only the utility of the action taken is known (e.g. in games where players alternate turns), then the expected total reward may be expressed as $U_T = \sum_{t=1}^{T} u_t(a_t)p(a_t)$.

2.2 SDR: Recurrent Policy Gradients

In this section, we present the batch version of SDR with recurrent policy gradients. The next section describes the stochastic gradient approximation for on-line SDR learning

U_T can be maximized by performing gradient ascent on the utility gradient

$$\Delta\theta \propto \frac{dU_T}{d\theta} = \sum_t \sum_{a_t} u_t(a_t) \frac{d}{d\theta} p(a_t). \tag{4}$$

[1] In situations where aspects of the environment can be modeled, then I_{t-1} may depend to some extent in a known way on A_{t-1}. This implicit dependence on policy parameters θ introduces additional recurrence which can be captured in the learning algorithm. In the context of games, this requires explicitly modeling the opponent's responses. Environment and opponent models are beyond the scope of this short paper.

[2] For small trades in liquid securities, this situation holds well. For very large trades, the non-trivial effects of market impact must be considered.

In this section, we derive expressions for the batch policy gradient $\frac{dp(a_t)}{d\theta}$ with a stationary policy. When the policy has m-th order recurrence, the full policy gradients are expressed as m-th order recursions. These *recurrent gradients* are required to properly account for *temporal credit assignment* when an agent's prior actions directly influence its current action. The m-th order recursions are causal (propagate information forward in time), but non-Markovian.

For the non-recurrent $m = 0$ case, the policy gradient is simply:

$$\frac{dp(a_t)}{d\theta} = \frac{dP(a_t; \theta, I_{t-1}^{(n)})}{d\theta}. \tag{5}$$

For first order recurrence $m = 1$, the probabilities of current actions depend upon the probabilities of prior actions. Note that $A_{t-1}^{(1)} = a_{t-1}$ and denote $p(a_t|a_{t-1}) = P(a_t; \theta, I_{t-1}^{(n)}, a_{t-1})$. Then we have

$$p(a_t) = \sum_{a_{t-1}} p(a_t|a_{t-1}) p(a_{t-1}) \tag{6}$$

The recursive update to the total policy gradient is

$$\begin{aligned} \frac{dp(a_t)}{d\theta} &= \sum_{a_{t-1}} \left\{ \frac{\partial p(a_t|a_{t-1})}{\partial\theta} p(a_{t-1}) \right. \\ &\quad \left. + p(a_t|a_{t-1}) \frac{dp(a_{t-1})}{d\theta} \right\} \end{aligned} \tag{7}$$

where $\frac{\partial}{\partial\theta} p(a_t|a_{t-1})$ is calculated as $\frac{\partial}{\partial\theta} P(a_t; \theta, I_{t-1}^{(n)}, a_{t-1})$ with a_{t-1} fixed.

For the case $m = 2$, $A_{t-1}^{(2)} = \{a_{t-1}, a_{t-2}\}$. Denoting $p(a_t|a_{t-1}, a_{t-2}) = P(a_t; \theta, I_{t-1}^{(n)}, a_{t-1}, a_{t-2})$, we have the second-order recursion

$$\begin{aligned} p(a_t) &= \sum_{a_{t-1}} p(a_t|a_{t-1}) p(a_{t-1}) \\ p(a_t|a_{t-1}) &= \sum_{a_{t-2}} p(a_t|a_{t-1}, a_{t-2}) p(a_{t-2}) \end{aligned} \tag{8}$$

The second-order recursive updates for the total policy gradient can be expressed with two equations. The first is

$$\begin{aligned} \frac{d}{d\theta} p(a_t) &= \sum_{a_{t-1}} \left\{ \frac{\partial}{\partial\theta} p(a_t|a_{t-1}) p(a_{t-1}) \right. \\ &\quad \left. + p(a_t|a_{t-1}) \frac{d}{d\theta} p(a_{t-1}) \right\}. \end{aligned} \tag{9}$$

In contrast to Eq.(7) for $m = 1$, however, the first gradient on the RHS of Eq.(9) must be evaluated using

$$\begin{aligned} \frac{d}{d\theta} p(a_t|a_{t-1}) &= \sum_{a_{t-2}} \left\{ \frac{\partial}{\partial\theta} p(a_t|a_{t-1}, a_{t-2}) p(a_{t-2}) \right. \\ &\quad \left. + p(a_t|a_{t-1}, a_{t-2}) \frac{d}{d\theta} p(a_{t-2}) \right\}. \end{aligned} \tag{10}$$

In the above expression, both a_{t-1} and a_{t-2} are treated as fixed and $\frac{\partial}{\partial\theta} p(a_t|a_{t-1}, a_{t-2})$ is calculated as $\frac{\partial}{\partial\theta} P(a_t; \theta, I_{t-1}^{(n)}, a_{t-1}, a_{t-2})$. These recursions can be extended in the obvious way for $m \geq 3$.

2.3 SDR with Stochastic Gradients

The formulation of SDR of Section 2.2 provides the exact recurrent policy gradient. It can be used for learning off-line in batch mode or for on-line applications by retraining as new experience is acquired. For the latter, either a complete history or a moving window of experience may be used for batch training, and policy parameters can be estimated from scratch or simply updated from previous values.

We find in practice, though, that a stochastic gradient formulation of SDR yields much more efficient learning than the exact version of the previous section. This frequently holds whether an application requires a pre-learned solution or adaptation in real-time.

To obtain SDR with stochastic gradients, an estimate $g(a_t)$ of the recurrent policy gradient for time-varying parameters θ_t is computed at each time t. The on-line SDR parameter update is then the stochastic approximation

$$\Delta\theta_t \propto \sum_{a_t} u_t(a_t) g_t(a_t). \tag{11}$$

The on-line gradients $g_t(a_t)$ are estimated via approximate recursions. For $m = 1$, we have

$$\begin{aligned} g_t(a_t) \quad &\approx \quad \sum_{a_{t-1}} \left\{ \frac{\partial p_t(a_t|a_{t-1})}{\partial \theta} p_{t-1}(a_{t-1}) \right. \\ &\left. + p_t(a_t|a_{t-1}) g_{t-1}(a_{t-1}) \right\}. \end{aligned} \tag{12}$$

Here, the subscripts indicate the times at which each quantity is evaluated. For $m = 2$, the stochastic gradient is approximated as

$$\begin{aligned} g_t(a_t) \quad &\approx \quad \sum_{a_{t-1}} \left\{ \frac{\partial p_t(a_t|a_{t-1})}{\partial \theta} p_{t-1}(a_{t-1}) \right. \\ &\left. + p_t(a_t|a_{t-1}) g_{t-1}(a_{t-1}) \right\}, \text{ with} \end{aligned} \tag{13}$$

$$\begin{aligned} \frac{dp_t(a_t|a_{t-1})}{d\theta} \quad &\approx \quad \sum_{a_{t-2}} \left\{ \frac{\partial}{\partial \theta} p_t(a_t|a_{t-1}, a_{t-2}) p_{t-2}(a_{t-2}) \right. \\ &\left. + p_t(a_t|a_{t-1}, a_{t-2}) g_{t-2}(a_{t-2}) \right\}. \end{aligned} \tag{14}$$

Stochastic gradient implementations of learning algorithms are often referred to as "on-line". Regardless of the terminology used, one should distinguish the number of learning iterations (parameter updates) from the number of events or time steps of an agent's experience. The computational issues of how parameters are optimized should not be limited by how much information or experience an agent has accumulated.

In our simulation work with RRL and SDR, we have used various types of learning protocols based on stochastic gradient parameter updates: (1) A single sequential learning pass (or *epoch*) through a history window of length τ. This results in one parameter update per observation, action or event and a total of τ parameter updates. (2) n multiple learning epochs over a fixed window. This yields a total of $n\tau$ stochastic gradient updates. (3) Hybrid on-line / batch training using history subsets (*blocks*) of duration b. With this method, a single pass through the history has τ/b parameter updates. This can reduce variance in the gradient estimates. (4) Training on N randomly resampled blocks from the a history window. (5) Variations and combinations of the above. Note that (4) can be used to implement bagging or to reduce the likelihood of being caught in history-dependent local minima.

Following work by other authors in multi-agent RL, all simulation results presented in this paper use protocol (1). Hence, the number of parameter updates equals the length of agent's history. In real-world use, however, the other learning protocols are usually preferred. RL agents with limited experience typically learn better with stochastic gradient methods when using multiple training epochs n or a large number parameter updates $N >> \tau$.

2.4 Model Structures

In this section we describe the two model structures we use in our simulations. The first model can be used for games with binary actions and is based on the $tanh$ function. The second model is appropriate for multiple-action games and uses the softmax representation. Recall that the SDR agent with memory length n and recurrence of order m takes an action with probability, $p(a_t) = P(a_t; \theta, I_{t-1}^{(n)}, A_{t-1}^{(m)})$. Let $\theta = \{\theta_0, \theta_I, \theta_A\}$ denote the bias, non-recurrent, and recurrent weight vectors respectively. The binary-action recurrent policy function has the form $P(a_t = +1; \theta, I_{t-1}^{(n)}, A_{t-1}^{(m)}) = (1 + f_t)/2$, where $f_t = \tanh(\theta_0 + \theta_I \cdot I_{t-1}^{(n)} + \theta_A \cdot A_{t-1}^{(m)})$ and $p(-1) = 1 - p(+1)$. The multiple-action recurrent policy function has the form

$$p(a_t = i) = \frac{\exp(\theta_{i0} + \theta_{iI} \cdot I_{t-1}^{(n)} + \theta_{iA} \cdot A_{t-1}^{(m)})}{\sum_j \exp(\theta_{j0} + \theta_{jI} \cdot I_{t-1}^{(n)} + \theta_{jA} \cdot A_{t-1}^{(m)})}. \tag{15}$$

2.5 Relation to "Memory-Based" Methods

For the repeated matrix games with dynamic players that we consider in this paper, the SDR agents have *both* memory of opponents' actions $I_{t-1}^{(n)}$ and recurrent dependencies on their own previous actions $A_{t-1}^{(m)}$. These correspond to *standard, non-recurrent memory* and *recurrent memory*, respectively. To our knowledge, we are the first in the RL community to make this distinction, and to propose recurrent policy gradient algorithms such as RRL and SDR to properly handle the temporal dependencies of an agent's actions.

A number of "memory-based" approaches to value function RL algorithms such as Q-Learning have been discussed (Lin & Mitchell 1992; McCallum 1995). With these methods, prior actions are often included in the observation I_{t-1} or state vector S_t. In a policy gradient framework, however, simply including prior actions in the observation vector I_{t-1} ignores their dependence upon the policy parameters θ. As mentioned previously, the probabilities of prior actions $A_{t-1}^{(m)}$ depend explicitly on θ, but components of $I_{t-1}^{(n)}$ are assumed to have *no known or direct dependence* on θ.[3]

[3]Of course, agents do influence their environments, and game players do influence their opponents. Such interaction with the en-

A standard memory-based approach of treating $A_{t-1}^{(m)}$ as part of I_{t-1} by ignoring the dependence of $A_{t-1}^{(m)}$ on θ would in effect truncate the gradient recursions above. This would replace probabilities of prior actions $p(a_{t-j})$ with Kronecker δ's (1 for the realized action and 0 for all other possible actions) and setting $\frac{d}{d\theta}p(a_{t-1})$ and $\frac{d}{d\theta}p(a_{t-2})$ to zero. When this is done, the $m = 1$ and $m = 2$ recurrent gradients of equations (7) to (10) collapse to:

$$\begin{aligned} \frac{d}{d\theta}p(a_t) &\approx \frac{\partial}{\partial\theta}p(a_t|a_{t-1}), & m = 1 \\ \frac{d}{d\theta}p(a_t) &\approx \frac{\partial}{\partial\theta}p(a_t|a_{t-1}, a_{t-2}), & m = 2. \end{aligned}$$

These equations are similar to the non-recurrent $m = 0$ case in that they have no dependence upon the policy gradients computed at times $t - 1$ or $t - 2$. When total derivative are approximated by partial derivatives, sequential dependencies that are important for solving the *temporal credit assignment problem* when estimating θ are lost.

In our empirical work with RRL and SDR, we have found that removal of the recurrent gradient terms can lead to failure of the algorithms to learn effective policies. It is beyond the scope of this paper to discuss recurrent gradients and temporal credit assignment further or to present unrelated empirical results. In the next two sections, we use repeated matrix games with stochastic players as a microcosm to (1) compare the policy gradient approach to Q-type learning algorithms and (2) explore the use of recurrent representations, and interacting, dynamic agents.

3 SDR vs. Value Function Methods for a Simple Game

The purpose of this section is to compare simple SDR agents with Q-type agents via competition in a simple game.

Nearly all investigations of reinforcement learning in multi-agent systems have utilized Q-type learning methods. Examples include bidding agents for auctions (Kephart, Hansen, & Greenwald 2000; Tesauro & Bredin 2002), autonomic computing (Boutilier *et al.* 2003), and games (Littman 1994; Hu & Wellman 1998), to name a few.

However, various authors have noted that Q-functions can be cumbersome for representing and learning good policies. Examples have been discussed in robotics (Anderson 2000), telecommunications (Brown 2000), and finance (Moody & Saffell 1999; 2001). The latter papers provide comparisons of trading agents trained with the RRL policy gradient algorithm and Q-Learning for the real-world problem of allocating assets between the S&P 500 stock index and T-Bills. A simple toy example that elucidates the representational complexities of Q-functions versus direct policy representations is the Oracle Problem described in (Moody & Saffell 2001).

To investigate the application of policy gradient methods to multi-agent systems, we consider competition between SDR and Q-type agents playing Rock-Paper-Scissors (RPS), a simple two-player repeated matrix game. We chose our opponents to be two recently proposed Q-type learners with mixed strategies, WoLF-PHC (Bowling & Veloso 2002) and Hyper-Q (Tesauro 2004). We selected RPS for the non-recurrent case studies in this section in order to build upon the empirical comparisons provided by Bowling & Veloso and Tesauro. Hence, we use memory-less Q-type players and follow key elements of these authors' experimental protocols. To enable fair competitions, the SDR players have access to only as much information as their PHC and Hyper-Q opponents. We also use the on-line learning paradigms of these authors, whereby a single parameter update (learning step) is performed after each game iteration (or "throw").

Rather than consider very long matches of millions of throws as previous authors have,[4] we have chosen to limit all simulations in this paper to games of 20,000 plays or less. We believe that $O(10^4)$ plays is more than adequate to learn a simple matrix game.The Nash equilibrium strategy for RPS is just random play, which results in an average draw for both contestants. This strategy is uninteresting for competition, and the discovery of Nash equilibrium strategies in repeated matrix games has been well-studied by others. Hence, the studies in this section emphasize the non-equilibrium, short-run behaviors that determine a player's viability as an effective competitor in plausible matches. Since real tournaments keep score from the beginning of a game, it is cumulative wins that count, not potential asymptotic performance. Transient performance matters and so does speed of learning.

3.1 Rock-Paper-Scissors & Memory-Less Players

RPS is a two-player repeated matrix game where one of three actions {(R)ock, (P)aper, (S)cissors} is chosen by each player at each iteration of the game. Rock beats Scissors which beats Paper which in turn beats Rock. Competitions are typically sequences of throws. A $+1$ is scored when your throw beats your opponent's throw, a 0 when there is a draw, and a -1 when your throw is beaten. The Nash equilibrium for non-repeated play of this game is to choose the throw randomly.

In this section we will compare SDR players to two Q-type RPS opponents: the Policy Hill Climbing (PHC) variant of Q-Learning (Bowling & Veloso 2002) and the Hyper-Q algorithm (Tesauro 2004). The PHC and Hyper-Q players are *memory-less*, by which we mean that they have no encoding of previous actions in their Q-table state spaces. Thus, there is no sequential structure in their RPS strategies. These opponents' mixed policies are simple *unconditional probabilities* for throwing rock, paper or scissors, and are *independent* of their own or their opponents' recent past actions. These Q-type agents are in essence three-armed bandit players. When learning rates are zero, their memory-less strategies are static. When learning rates are positive, competing against these opponents is similar to playing a "restless" three-armed bandit.

vironment results in *implicit* dependencies of $I_{t-1}^{(n)}$ on θ. We describe a more general formulation of SDR with environment models that capture such additional recurrence in another paper.

[4]See the discussion of RPS tournaments in Section 4.2. Computer RoShamBo and human world championship tournaments have much shorter games.

For the RPS competitions in this section, we use non-recurrent SDR agents. Since the Q-type players are memory-less and non-dynamic, the SDR agents do not require inputs corresponding to lagged actions of their own or their opponents to learn superior policies. Thus, the policy gradient players use the simple $m = 0$ case of the SDR algorithm.

The memory-less SDR and Q-type players have time-varying behavior due only to their learning. When learning rates are zero, the players policies are static unconditional probabilities of throwing {R,P,S}, and the games are stationary with no sequential patterns in play. When learning rates are non-zero, the competitions between the adaptive RL agents in this section are analogous to having two restless bandits play each other. This is also the situation for all empirical results for RL in repeated matrix games presented by (Singh, Kearns, & Mansour 2000; Bowling & Veloso 2002; Tesauro 2004).

SDR vs. PHC PHC is a simple extension of Q-learning. The algorithm, in essence, performs hill-climbing in the space of mixed policies toward the greedy policy of its current Q-function. The policy function is a table of three unconditional probabilities with no direct dependence on recent play. We use the WoLF-PHC (win or learn fast) variant of the algorithm.

For RPS competition against WoLF-PHC, an extremely simple SDR agent with no inputs and only bias weights is sufficient. The SDR agent represents mixed strategies with softmax: $p(i) = \frac{e^{w_i}}{\sum_j e^{w_j}}$, for $i, j = R, P, S$. SDR without inputs resembles Infinitesimal Gradient Ascent (Singh, Kearns, & Mansour 2000), but assumes no knowledge of the opponent's mixed strategy.

For the experiments in this section, the complexities of both players are quite simple. SDR maintains three weights. WoLF-PHC updates a Q-table with three states and has three policy parameters.

At the outset, we strove to design a fair competition, whereby neither player would have an obvious advantage. Both players have access to the same information, specifically just the current plays and reward. The learning rate for SDR is $\eta = 0.5$, and the weight decay rate is $\lambda = 0.001$. The policy update rates of WoLF-PHC for the empirical results we present are $\delta_l = 0.5$ when losing and $\delta_w = 0.125$ when winning. The ratio $\delta_l/\delta_w = 4$ is chosen to correspond to experiments done by (Bowling & Veloso 2002), who used ratios of 2 and 4. We select δ_l for PHC to equal SDR's learning rate η to achieve a balanced baseline comparison. However, since $\delta_w = \eta/4$, PHC will not deviate from winning strategies as quickly or as much as will SDR. (While learning, some policy fluctuation occurs at all times due to the randomness of plays in the game.)

Figure 1 shows a simulation result with PHC's Q-learning rate set to 0.1, a typical value used by many in the RL community. Some might find the results to be surprising. Even with well-matched policy learning parameters for SDR and PHC, and a standard Q learning rate, the Q function of PHC does not converge. Both strategies evolve continuously, and SDR dominates WoLF-PHC.

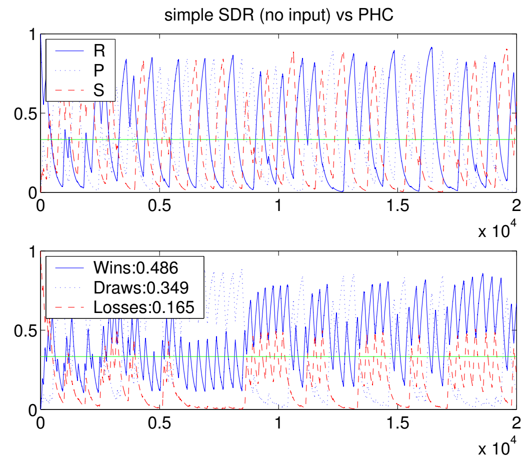

Figure 1: The dynamic behavior of the SDR player when playing Rock-Paper-Scissors against the WoLF-PHC player with Q learning rate 0.1. The top panel shows the changing probabilities of the SDR players strategy over time. The bottom panel shows the moving average of the percentage of wins, losses, and draws against WoLF-PHC. Although chaotic limit cycles in play develop, SDR achieves a consistent advantage.

Upon investigation, we find that the behavior apparent in Figure 1 is due to PHC's relative sluggishness in updating its Q-table. The role of time delays (or mis-matched time-scales) in triggering such emergent chaotic behavior is well-known in nonlinear dynamical systems. The simple version of SDR used in this experiment has learning delays, too; it essentially performs a moving average estimation of its optimal mixed strategy based upon PHC's actions. Even so, SDR does not have the encumbrance of a Q-function, so it always learns faster than PHC. This enables SDR to exploit PHC, while PHC is left to perpetually play "catch-up" to SDR. As SDR's strategy continuously evolves, PHC plays catch-up, and neither player reaches the Nash equilibrium strategy of random play with parity in performance.[5]

To confirm this insight, we ran simulated competitions for various Q learning rates while keeping the policy update rates for PHC and SDR fixed. We also wished to explore how quickly WoLF-PHC would have to learn to be competitive with SDR. Figure 2 shows performance results for a range of Q learning rates from $1/32$ to 1. WoLF-PHC is able to match SDR's performance by achieving net wins of zero (an average draw) only when its Q learning rate is set equal to 1.[6]

[5]In further experiments, we find that the slowness of Q-function updates prevents PHC from responding directly to SDR's change of strategy even with hill-climbing rates δ_l approaching 1.0.

[6]In this limit, the Q learning equations simplify, and Q-table elements for stochastic matrix games equal recently received rewards plus a stochastic constant value. Since the greedy policy depends only upon relative Q values for the possible actions, the Q-function can be replaced by the immediate rewards for each action. Hence, PHC becomes similar to a pure policy gradient algorithm, and both

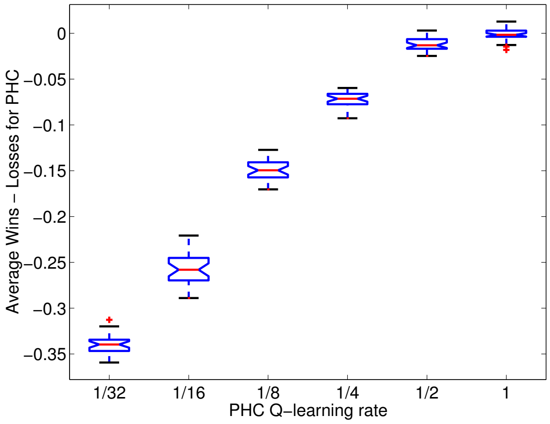

Figure 2: Average wins minus losses for PHC versus its Q learning rate. The Tukey box plots summarize the distributions of outcomes of 10 matches against SDR opponents for each case. Each match has 2.5×10^4 throws, and wins minus losses are computed for the last 2×10^4 throws. The results show that PHC loses to SDR on average for all Q learning rates less than 1, and the dominance of SDR is statistically significant. With a Q learning rate of 1, PHC considers only immediate rewards and behaves like a pure policy gradient algorithm. Only in this case is PHC able to match SDR's performance, with both players reaching Nash equilibrium and achieving an average draw.

SDR vs. Hyper-Q The standard Q-Learning process of using discrete-valued actions is not suitable for stochastic matrix games as argued in (Tesauro 2004). Hyper-Q extends the Q-function framework to *joint* mixed strategies. The hyper-Q-table is augmented by a Bayesian inference method for estimating other agent's strategies that applies a recency-weighted version of Bayes' rule to the observed action sequence. In Tesauro's simulation, Hyper-Q took 10^6 iterations to converge when playing against PHC.

As for PHC, we strove to design a fair competition between SDR and Hyper-Q, whereby neither player would have an obvious information advantage. Both players have access to the same observations, specifically the Bayesian estimate of its opponent's mixed strategy and the most recent plays and reward. Following Tesauro's RPS experiments, our simulation of the Hyper-Q player has a discount factor $\gamma = 0.9$ and a constant learning rate $\alpha = 0.01$ and uses a Bayes estimate of the opponent's strategies. Similarly, the input of SDR is a vector consisting of three probabilities estimating Hyper-Q's mixed strategy. The estimation is performed by applying the same recency-weighted Bayes' rule as used in Hyper-Q on the observed actions of the opponent. The probability $p(y|H)$ that the opponent is using mixed strategy y given the history H of observed opponent's actions can be estimated using Bayes' rule $p(y|H) \propto p(H|y)p(y)$. A recency-weighted version of Bayes' rule is obtained using $p(H|y) = \prod_{k=0}^{t} p(o_k|y(t))^{w_k}$, where $w_k = 1 - \mu(t-k)$. In this simulation, SDR maintains nine weights, whereas Hyper-Q stores a Q-table of 105625 elements.

SDR and PHC players reach Nash equilibrium.

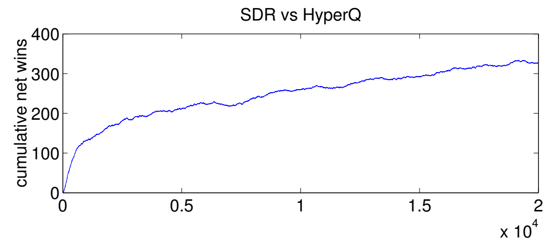

Figure 3: SDR vs Hyper-Q in a Rock-Paper-Scissors match. Following (Tesauro 2004), we plot the cumulative net wins (the sum of wins minus losses) of SDR over Hyper-Q versus number of throws. This graphical representation depicts the outcomes of matches of length up to 20,000 throws. The curve is on average increasing; SDR beats Hyper-Q more often than not as play continues. The failure of the curve to flatten out (which would corresponding to eventual net draws on average) during the game length demonstrates the relatively faster learning behavior of the SDR player. Based upon Tesauro's empirical work, we anticipate that convergence of both players to the Nash equilibrium of random play would require on the order of 10^6 throws. However, we limited all simulations in this paper to games of 20,000 throws or less.

Figure 3 shows the cumulative net wins for the SDR player over the Hyper-Q player during learning. The SDR player exploits the slow learning of Hyper-Q before the players reach the Nash equilibrium strategy of random play. During our entire simulation of 2×10^4 plays, the net wins curve is increasing and SDR dominates Hyper-Q.

3.2 Discussion

Our experimental results show that simple SDR players can adapt faster than more complex Q-type agents in RPS, thereby achieving winning performance. The results suggest that policy gradient methods like SDR may be sufficient for attaining superior performance in repeated matrix games, and also raise questions as to when use of a Q-function can provide benefit.

Regarding this latter point, we make two observations. First, repeated matrix games provide players with frequent, immediate rewards. Secondly, the implementations of PHC used by (Bowling & Veloso 2002) and of Hyper-Q used by (Tesauro 2004) are non-dynamic and memory-less[7] in the sense that their policies are just the marginal probabilities of throwing rock, paper or scissors, and current actions are not conditioned on previous actions. There are no sequential patterns in their play, so adaptive opponents do not have to

[7] One can distinguish between "long-term memory" as encoded by learned parameters and "short-term memory" as encoded by input variables. Tesauro's moving average Bayes estimate of the opponent's mixed strategy is used as an input variable, but really captures long-term behavior rather than the short term, dynamical memory associated with recent play.

solve a temporal credit assignment problem. We leave it to proponents of Q-type algorithms like WoLF-PHC or Hyper-Q to investigate more challenging multi-agent systems applications where Q-type approaches may provide significant benefits and to explore dynamical extensions to their algorithms.

In the next section, we present studies of SDR in repeated matrix games with stochastic opponents that exhibit more complex behaviors. For these cases, SDR agents must solve a temporal credit assignment problem to achieve superior play.

4 Games with Dynamic Contestants

Players using memory-less, non-dynamic mixed strategies as in (Bowling & Veloso 2002; Tesauro 2004) show none of the short-term patterns of play which naturally occur in human games. Dynamics exist in strategic games due to the fact that the contestants are trying to discover patterns in each other's play, and are modifying their own strategies to attempt to take advantage of predictability to defeat their opponents. To generate dynamic behavior, game playing agents must have memory of recent actions. We refer to players with such memory as *dynamic contestants*.

We distinguish between two types of dynamical behaviors. Purely *reactive* players observe and respond to only the past actions of their opponents, while purely *non-reactive* players remember and respond to only their own past actions. Reactive players generate dynamical behavior only through interaction with an opponent, while non-reactive players ignore their opponents entirely and generate sequential patterns in play purely autonomously. We call such autonomous pattern generation *endogenous dynamics*. Reactive players have only non-recurrent memory, while non-reactive players have only recurrent memory. A *general dynamic* player has both reactive and non-reactive capabilities and exhibits both *reactive* and endogenous dynamics. A general dynamic player with non-reactive (recurrent) memory of length m and reactive (non-recurrent) memory of length n is said to have *dynamics of order* (m, n). A player that learns to model its opponent's dynamics (whether reactive or endogenous) and to dominate that opponent becomes *predictive*.

In this section, we illustrate the use of recurrent SDR learning in competitions between dynamic, stochastic contestants. As case studies, we consider sequential structure in the repeated matrix games, Matching Pennies (MP), Rock-Paper-Scissors (RPS), and the Iterated Prisoners' Dilemma (IPD). To effectively study the learning behavior of SDR agents and the effects of recurrence, we limit the complexity of the experiments by considering only non-adaptive opponents with well-defined policies or self-play.

4.1 Matching Pennies

Matching pennies is a simple zero sum game that involves two players, A and B. Each player conceals a penny in his palm with either head or tail up. Then they reveal the pennies simultaneously. By an agreement made before the play, if both pennies match, player A wins, otherwise B wins. The player who wins receives a reward of $+1$ while the loser receives -1.

Figure 4: Matching Pennies with a *general dynamic* opponent. Boxplots summarize the average reward (wins minus losses) of an SDR learner over 30 simulated games. Each game consists of 2000 rounds. Since the opponent's play depends on both its own and its opponents play, both recurrent and non-recurrent inputs are necessary to maximize rewards.

The matching pennies SDR player uses the softmax function as described in Section 2.4. and $m = 1$ recurrence. We tested versions of the SDR player using three input sets: the first, "Non-Recurrent" with $(m, n) = (0, 1)$, uses only its opponent's previous move o_{t-1}, the second, "Only Recurrent" with $(m, n) = (1, 0)$, uses only its own previous move a_{t-1}, and the third, "Both Types" with $(m, n) = (1, 1)$, uses both its own and its opponent's previous move. Here, a_{t-1} and o_{t-1} are represented as vectors $(1, 0)^T$ for heads and $(0, 1)^T$ for tails.

To test the significance of recurrent inputs, we design two kinds of reactive opponents that will be described in the following two sections.

General Dynamic Opponent The general dynamic opponent makes its action based on both its own previous action (internal dynamics) and its opponent's previous action (reactive dynamics). In our test, we construct an opponent using the following strategy:

$$\mathbf{V} = \begin{pmatrix} 0.6 & 0.4 & 0.2 & 0.8 \\ 0.4 & 0.6 & 0.8 & 0.2 \end{pmatrix}$$
$$\mathbf{p} = \mathbf{V}(a_{t-1} \otimes o_{t-1})$$

where $a_{t-1} \otimes o_{t-1}$ is the outer product of a_{t-1} and o_{t-1}. The matrix $\mathbf{V}$ is constant. $\mathbf{p} = (p_H, p_T)^T$ gives the probability of presenting a head or tail at time t. Softmax is not needed. SDR players use the softmax representation illustrated in Eq.(15) with $I^{(1)}_{t-1} = o_{t-1}$ and $A^{(1)}_{t-1} = a_{t-1}$. The reactive and non-reactive weight matrices for the SDR player are θ_I and θ_A, respectively.

The results in Figure 4 show the importance of including both actions in the input set to maximize performance.

Purely Endogenous Opponent The *purely endogenous* opponent chooses its action based only on its own previous

Figure 5: Matching Pennies with a *purely endogenous* opponent for 30 simulations. Such a player does not react to its opponent's actions and cannot learn to directly predict its opponents responses. These features make such opponents not very realistic of challenging. Boxplots of the average reward (wins minus losses) of an SDR learner. The results show that non-recurrent, memory inputs are sufficient to provide SDR players with the predictive abilities needed to beat a purely endogenous opponent.

actions. Such a player does not react to its opponent's actions, but generates generates patterns of play autonomously. Since they cannot learn strategies to predict their opponents responses, endogenous players are neither very realistic nor very challenging to defeat.

The following is an example of the endogenous dynamic opponent's strategy:

$$\begin{aligned} \mathbf{V} &= \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix} \\ \mathbf{p} &= \mathbf{V} o_{t-1} \end{aligned} \tag{16}$$

Figure 5 shows results for 30 games of 2000 rounds of the three SDR players against the purely endogenous opponent. The results show that including only the non-recurrent inputs provides enough information for competing successfully against it.

Purely Reactive Opponent The *purely reactive* opponent chooses its action based only on its opponents' previous actions. The opponent has no endogenous dynamics of its own, but generates game dynamics through interaction. Such a player is more capable of learning to predict its opponents' behaviors and is thus a more realistic choice for competition than a purely endogenous player. The following is an example of the reactive opponent's strategy:

$$\begin{aligned} \mathbf{V} &= \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix} \\ \mathbf{p} &= \mathbf{V} a_{t-1} \end{aligned} \tag{17}$$

Figure 6 shows results for 30 games of 2000 rounds of the three SDR players against the purely reactive opponent. The results show that including only the recurrent inputs provides enough information for competing successfully against a purely reactive opponent.

Figure 6: Matching Pennies with a *purely reactive* opponent for 30 simulations. Boxplots of the average reward (wins minus losses) of an SDR learner. The results clearly show that non-recurrent, memory inputs provide no advantage, and that recurrent inputs are necessary and sufficient to provide SDR players with the predictive abilities needed to beat a purely reactive opponent.

4.2 Rock Paper Scissors

Classical game theory provides a static perspective on mixed strategies in games like RPS. For example, the Nash equilibrium strategy for RPS is to play randomly with equal probabilities of $\{R, P, S\}$, which is static and quite dull. Studying the dynamic properties of human games in the real world and of repeated matrix games with dynamic contestants is much more interesting. Dynamic games are particularly challenging when matches are short, with too few rounds for the law of large numbers to take effect. Human RPS competitions usually consist of at most a few dozens of throws. The RoShamBo (another name for RPS) programming competition extends game length to just one thousand throws. Competitors cannot win games or tournaments on average with the Nash equilibrium strategy of random play, so they must attempt to model, predict and dominate their opponents. Quickly discovering deviations from randomness and patterns in their opponents' play is crucial for competitors to win in these situations.

In this section, we demonstrate that SDR players can learn winning strategies against dynamic and predictive opponents, and that recurrent SDR players learn better strategies than non-recurrent players. For the experiments shown in this section, we are not concerned with the speed of learning in game time. Hence, we continue to use the "naive" learning protocol (1) described in Section 2.3 which takes one stochastic gradient step per RPS throw. This protocol is sufficient for our purpose of studying recurrence and recurrent gradients. In real-time tournament competition, however, we would use a learning protocol with many stochastic gradient steps per throw that would make maximally efficient use of acquired game experience and converge to best possible play much more rapidly.

Human Champions Dataset & Dynamic Opponent To simulate dynamic RPS competitions, we need a *general dynamic* opponent for SDR to compete against. We create this dynamic opponent using SDR by training on a dataset inspired by human play and then fixing its weights after convergence. The "Human Champions" dataset consists of triplets of plays called gambits often used in human world championship play (http://www.worldrps.com). (One example is the "The Crescendo" gambit P-S-R). The so-called "Great Eight" gambits were sampled at random using prior probabilities such that the resulting unconditional probabilities of each action {R,P,S} are equal. Hence there were no trivial vulnerabilities in its play. The *general dynamic* player has a softmax representation and uses its opponent's two previous moves and its own two previous moves to decide its current throw. It is identical to an SDR player, but the parameters are fixed once training is complete. Thus the dynamic player always assumes its opponents are "human champions" and attempts to predict their throws and defeat them accordingly. The dynamic player is then used as a predictive opponent for testing different versions of SDR-Learners.

Importance of recurrence in RPS Figures 7 and 8 show performance for three different types of SDR-Learners for a single match of RPS. The "Non-Recurrent" players, $(m, n) = (0, 2)$, only have knowledge of their opponents previous two moves. The "Only Recurrent" players, $(m, n) = (2, 0)$, know only about their own previous two moves. The "Both Types" players have dynamics of order $(m, n) = (2, 2)$, with both recurrent and non-recurrent memory. Each SDR-Learner plays against the same dynamic "Human Champions" opponent described previously. The SDR players adapt as the matches progress, and are able to learn to exploit weaknesses in their opponents' play. While each player is able to learn a winning strategy, the players with recurrent knowledge are able to learn to predict their opponents' reactions to previous actions. Hence, the recurrent SDR players gain a clear and decisive advantage.

4.3 Iterated Prisoners' Dilemma (IPD)

The Iterated Prisoners Dilemma IPD is a general-sum game in which groups of agents can learn either to defect or to cooperate. For IPD, we consider SDR self-play and play against fixed opponents. We find that non-recurrent SDR agents learn only the Nash equilibrium strategy "always defect", while SDR agents trained with recurrent gradients can learn a variety of interesting behaviors. These include the Pareto-optimal strategy "always cooperate", a time-varying strategy that exploits a generous opponent, and the evolutionary stable strategy Tit-for-Tat (TFT).

Learning to cooperate with TFT is a benchmark problem for game theoretic algorithms. Figure 9 shows IPD play with a dynamic TFT opponent. We find that non-recurrent SDR agents prefer the static, Nash equilibrium strategy of defection, while recurrent $m = 1$ players are able to learn to cooperate, and find the globally-optimal (Pareto) equilibrium. Without recurrent policy gradients, the recurrent SDR player also fails to cooperate with TFT, showing that recurrent learning and not just a recurrent representation is required. We find similar results for SDR self-play, where SDR agents learn to always defect, while recurrent gradient

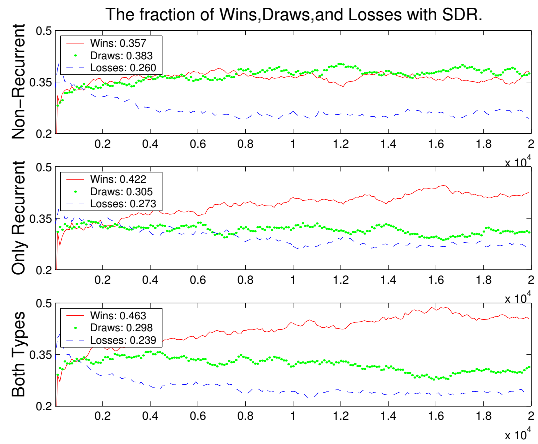

Figure 7: Learning curves showing fraction of wins, losses and draws during an RPS match between SDR players and a dynamic, predictive opponent. The top player has non-recurrent inputs, the middle player has only recurrent inputs, and the bottom player has both types of inputs. Recurrent SDR players achieve higher fractions of wins than do the non-recurrent players. The length of each match is 20000 throws with one stochastic gradient learning step per throw. Fractions of wins, draws and losses are calculated using the last 4000 throws. Note that the learning traces shown are only an illustration; much faster convergence can be obtained using multiple stochastic gradient steps per game iteration.

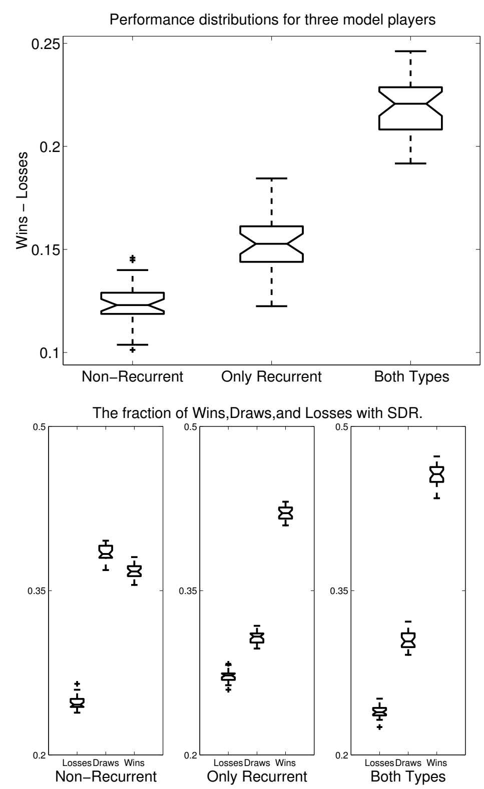

Figure 8: Performance distributions for SDR learners competing against predictive "human champions" RPS opponents. Tukey boxplots summarize the results of 30 matches such as those shown in Figure 7. The SDR players with recurrent inputs have a clear and decisive advantage.

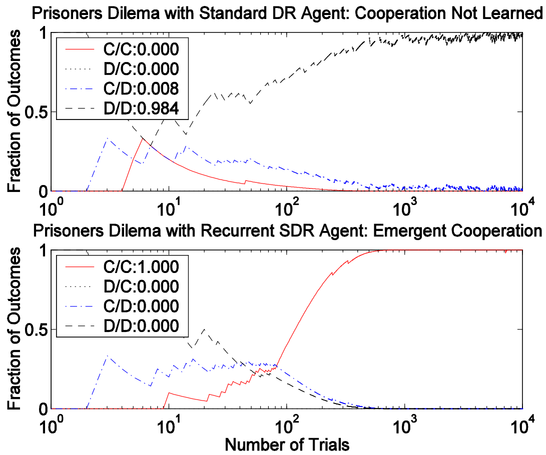

Figure 9: IPD: SDR agents versus TFT opponent. Plots of the frequencies of outcomes, for example C/D indicates TFT "cooperates" and SDR "defects". The top panel shows that a simple $m = 1$ SDR player with recurrent inputs but no recurrent gradients learns to always defect (Nash equilibrium). This player is unable to solve the temporal credit assignment problem and fails to learn to cooperate with the TFT player. The bottom panel shows that an $m = 1$ SDR player with recurrent gradients that can learn to cooperate with the TFT player, thus finding the globally-optimal Pareto equilibrium.

SDR players learn to always cooperate.

A Tit-for-Two-Tat (TF2T) player is more generous than a TFT player. TF2T considers two previous actions of its opponent and defects only after two opponent defections. When playing against TF2T, an $m = 2$ recurrent SDR agent learns the optimal dynamic strategy to exploit TF2T's generosity. This exploitive strategy alternates cooperation and defection: C-D-C-D-C-D-C-... . Without recurrent gradients, SDR can not learn this alternating behavior.

We next consider a multi-agent generalization of IPD in which an SDR learner plays a heterogeneous population of opponents. Table 1 shows the mixed strategy learned by SDR. The SDR player learns a stochastic Tit-for-Tat strategy (sometimes called "generous TFT"). Table 2 shows the Q-Table learned by a Q-Learner when playing IPD against a heterogeneous population of opponents. The table shows that the Q-Learner will only cooperate until an opponent defects, and from then on the Q-Learner will always defect regardless of the opponent's actions. The SDR player on the other hand, has a positive probability of cooperating even if the opponent has just defected, thus allowing for a switch to the more profitable cooperate/cooperate regime if the opponent is amenable.

TFT is called an "evolutionary stable strategy" (ESS), since its presence in a heterogeneous population of adaptive agents is key to the evolution of cooperation throughout the population. Recurrent SDR's ability to learn TFT may thus be significant for multi-agent learning; a recurrent SDR agent could in principle influence other learning agents

$a_t^{SDR} \setminus (a_{t-1}^{SDR}, a_{t-1}^{opp})$	CC	CD	DC	DD
C	0.99	0.3	0.97	0.1
D	0.01	0.7	0.03	0.9

Table 1: The mixed strategy learned by SDR when playing IPD against a heterogeneous population of opponents. The elements of the table show the probability of taking an action at time t given the previous action of the opponent and of the player itself. The SDR player learns a generous Tit-for-Tat strategy, wherein it will almost always cooperate after its opponent cooperates and sometimes cooperate after its opponent defects.

$a_t^{Q} \setminus \left(a_{t-1}^{Q}, a_{t-1}^{opp}\right)$	CC	CD	DC	DD
C	69	2	-14	-12
D	9.7	4	0	0

Table 2: The Q-Table learned by a Q-Learner when playing IPD against a heterogeneous population of opponents. The Q-Learner will continue to defect regardless of its opponent's actions after a single defection by its opponent. Note the preference for defection indicated by the Q-values of column 3 as compared to the SDR player's strong preference for cooperation shown in Table 1, column 3.

to discover desirable individual policies that lead to Pareto-optimal behavior of a multi-agent population.

5 Closing Remarks

Our studies of repeated matrix games with stochastic players use a pure policy gradient algorithm, SDR, to compete against dynamic and predictive opponents. These studies distinguish between *reactive* and *non-reactive, endogenous* dynamics, emphasize the naturally recurrent structure of repeated games with dynamic opponents and use recurrent learning agents in repeated matrix games with stochastic actions.

Dynamic SDR players can be reactive or predictive of opponents' actions and can also generate endogenous dynamic behavior by using their own past actions as inputs (recurrence). When faced with a reactive or predictive opponent, an SDR player must have such recurrent inputs in order to predict the predictive responses of its opponent. The player must know what its opponent knows in order to anticipate the response. Hence, games with predictive competitors have feedback cycles that induce recurrence in policies and game dynamics.

The Stochastic Direct Reinforcement (SDR) algorithm used in this paper is a policy gradient algorithm for probabilistic actions, partially-observed states and recurrent, non-Markovian policies. The RRL algorithm of (Moody & Wu 1997; Moody *et al.* 1998) and SDR are policy gradient RL algorithms that distinguish between *recurrent memory* and standard, *non-recurrent memory* via the use of recurrent gradients during learning. SDR is well-matched to learning in repeated matrix games with unknown, dynamic opponents.

As summarized in Section 2, SDR's recurrent gradients are necessary for properly training agents with recurrent inputs. Simply including an agent's past actions in an observation or state vector (a standard "memory-based" approach) amounts to a truncation that neglects the dependence of the current policy gradient on previous policy gradients. By computing the full recurrent gradient, SDR captures dynamical structure that is important for solving the temporal credit assignment problem in RL applications.

While we have focused on simple matrix games in this paper, the empirical results presented in Section 3 support the view that policy gradient methods may offer advantages in simplicity, learning efficiency, and performance over value function type RL methods for certain applications. The importance of recurrence as illustrated empirically in Section 4 may have implications for learning in multi-agent systems and other dynamic application domains. Of particular interest are our results for the Iterated Prisoners Dilemma which include discovery of the Pareto-optimal strategy "always cooperate", a time-varying strategy that exploits a generous opponent and the evolutionary stable strategy Tit-for-Tat.

Some immediate extensions to the work presented in this paper include opponent or (more generally) environment modeling that captures additional recurrence and incorporating internal states and adaptive memories in recurrent SDR agents. These issues are described in another paper. It is not yet clear whether pure policy gradient algorithms like SDR can learn effective policies for substantially more complex problems or in environments with delayed rewards. There are still many issues to pursue further in this line of research, and the work presented here is just an early glimpse.

References

Anderson, C. W. 2000. Approximating a policy can be easier than approximating a value function. Technical Report CS-00-101, Colorado State University.

Baxter, J., and Bartlett, P. L. 2001. Infinite-horizon gradient-based policy search. *Journal of Artificial Intelligence Research* 15:319–350.

Boutilier, C.; Das, R.; Kephart, J. O.; Tesauro, G.; and Walsh, W. E. 2003. Cooperative negotiation in autonomic systems using incremental utility elicitation. In *UAI 2003*.

Bowling, M., and Veloso, M. 2002. Multiagent learning using a variable learning rate. *Artificial Intelligence* 136:215–250.

Brown, T. X. 2000. 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?".

Claus, C., and Boutilier, C. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, 746–752.

Hu, J., and Wellman, M. P. 1998. Multiagent reinforcement learning: Theoretical framework and an algorithm. In *Fifteenth International Conference on Machine Learning*, 242–250.

Kephart, J. O.; Hansen, J. E.; and Greenwald, A. R. 2000. Dynamic pricing by software agents. *Computer Networks* 32(6):731–752.

Lin, L. J., and Mitchell, T. 1992. Memory approaches to reinforcement learning in non-markovian domains. Technical Report CMUCS -92-138, Carnegie Mellon University, School of Computer Science.

Littman, M. L. 1994. Markov games as a framework for multi-agent reinforcement learning. In *ICML-94*, 157–163.

McCallum, A. 1995. Instance-based utile distinctions for reinforcement learning. In *Twelfth International Conference on Machine Learning*.

Moody, J., and Saffell, M. 1999. Reinforcement learning for trading. In Michael S. Kearns, S. A. S., and Cohn, D. A., eds., *Advances in Neural Information Processing Systems*, volume 11, 917–923. MIT Press.

Moody, J., and Saffell, M. 2001. Learning to trade via direct reinforcement. *IEEE Transactions on Neural Networks* 12(4):875–889.

Moody, J., and Wu, L. 1997. Optimization of trading systems and portfolios. In Abu-Mostafa, Y.; Refenes, A. N.; and Weigend, A. S., eds., *Decision Technologies for Financial Engineering*, 23–35. London: World Scientific.

Moody, J.; Wu, L.; Liao, Y.; and Saffell, M. 1998. Performance functions and reinforcement learning for trading systems and portfolios. *Journal of Forecasting* 17:441–470.

Ng, A., and Jordan, M. 2000. Pegasus: A policy search method for large mdps and pomdps. In *Proceedings of the Sixteenth Conference on Uncertainty in Artificial Intelligence*.

Sandholm, T. W., and Crites, R. H. 1995. Multi-agent reinforcement learning in iterated prisoner's dilemma. *Biosystems* 37:147–166.

Singh, S.; Kearns, M.; and Mansour, Y. 2000. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of UAI-2000*, 541–548. Morgan Kaufman.

Tesauro, G., and Bredin, J. 2002. Strategic sequential bidding in auctions using dynamic programming. In *Proceedings of the first International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 591–598.

Tesauro, G., and Kephart, J. O. 2002. Pricing in agent economies using multi-agent qlearning. *Autonomous Agents and Multi-Agent Systems* 5(3):289–304.

Tesauro, G. 2004. Extending Q-learning to general adaptive multi-agent systems. In S. Thrun, L. S., and Schölkopf, B., eds., *Advances in Neural Information Processing Systems*, volume 16. MIT Press.

Williams, R. J. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* 8:229–256.