Applying reinforcement learning to economic problems

Neal Hughes

Australian National University

November 14, 2014

Abstract

Reinforcement learning provides algorithms for solving Markov Decision Processes (MDPs), which do not require prior knowledge of the payoff and or transition functions. Rather agents 'learn' optimal polices by observing the outcomes (e.g., the payoffs and state transitions) of their actions.

In this paper, we introduce methods based around fitted Q iteration (Ernst et al. 2005): a batch version of Q-learning (Watkins 1992). Fitted Q iteration is proven to converge (in the single agent case) for certain types of function approximators - here we focus on the popular approximation method tile coding.

We consider how reinforcement learning can be applied to complex multi-agent problems: stochastic games, where each agent faces a MDP with transition and payoff functions that are dependent on the actions of the other players. We demonstrate these methods in the context of single and multi-agent water storage problems.

1 Introduction

Reinforcement learning provides a wide range of algorithms for solving Markov Decision Processes (MDPs), which do not require 'models' of the 'environment' (the payoff and transition functions). Rather agents 'learn' optimal polices by observing the outcomes of their actions: optimisation by simulation. Similar to dynamic programming, reinforcement learning methods work by exploiting the Bellman principle.

While used extensively in artificial intelligence and operations research, reinforcement learning has received limited attention in economics. One reason, is that for all their simplicity and intuitive appeal, many practical challenges are faced in adapting these methods to economic problems.

We focus on the method of fitted Q iteration (Ernst et al. 2005); a batch version of standard Q-learning (Watkins and Dayan 1992). In fitted Q iteration, a large number of action, payoff and state transition samples are simulated, to which an action-value or Q function is then fit. Similar to fitted value iteration, the method is proven to converge subject to assumptions on function approximation.

Our goal is to develop solution methods for complex multi-agent problems, specifically stochastic games, where each agent faces a MDP with state transition and payoff functions dependent on the behaviour of other agents. Here we develop an approach in the spirit of 'multi-agent learning' (Fudenberg and Levine 2007), which combines reinforcement learning with learning concepts from game theory.

Our approach provides a middle ground between the dynamic programming methods used in heterogeneous agent macro models and the simulation and search methods commonly used in agent based computational economics — both in terms of the size and complexity of the models it can be applied to and the degree of rationality or 'intelligence' assumed for the agents.

We begin by defining our problem space: the single agent MDP and the stochastic game. We then provide an introduction to reinforcement learning for single agent problems before detailing the function approximation techniques we employ, particularly tile coding. We then demonstrate the single agent method with an application to a water storage problem.

We then move on to multiple agent problems. Here we summarise the literature on solution concepts for stochastic games including the Markov Perfect Equilibrium and Oblivious Equilibrium, before discussing the overlapping computer science and economic literature on multi-agent learning. We then introduce our multiple agent version of fitted Q iteration, and detail its application to a water storage problems.

2 The problems

2.1 Markov decision process

A MDP represents the problem of an *agent* selecting some *action* in an *environment* in order to maximise a *reward*. A MDP operates in discrete time: each period given current state s_t the agent takes an action a_t, the environment then produces a reward r_t and a state transition s_{t+1}.

Formally, a Markov decision process is a tuple (S, A, T, R, β). $S \subset \mathbb{R}^{D_S}$ is the state space, where $D_S \in \{1, 2, ...\}$ is the dimensionality of the state space. $A \subset \mathbb{R}^{D_A}$ is the action space. $T : S \times A \times S \to [0, 1]$ is the transition function, a probability density function such that

$$\int_{S'} T(s, a, s')\ ds' = \text{Prob}(s_{t+1} \in S' | s_t = s, a_t = a)$$

$R : S \times A \to \mathbb{R}$ is the reward function. Finally $\beta \in (0, 1)$ is the discount rate. The users problem is to choose a_t to maximise the expected discounted reward

$$\max_{\{a_t\}_{t=0}^{\infty}} E \left\{ \sum_{t=0}^{t=\infty} \beta^t R(a_t, s_t) \right\}$$

given T, R, s_0 and $a_t \in \Gamma(s_t) \subset A$, where Γ is the feasibility correspondence.

A (Markovian) *policy function* for the MDP is a mapping from states to actions $f : S \to A$. The discounted expected reward of following policy f is defined

$$V^f(s) = E \left\{ \sum_{t=0}^{\infty} \beta^t R(f(s_t), s_t) | s_0 = s \right\}$$

where $s_{t+1} \sim T(s_t, f(s_t))$.

The *value function* associated with the optimal policy is defined as

$$V^*(s) = \sup\{V^f(s)\}$$

Typically the value function also satisfies the Bellman principle

$$V^*(s) = \max_a \left\{ R(s, a) + \beta \int_S T(s, a, s') V^*(s')\ ds' \right\}$$

2.2 Stochastic game

A stochastic game is essentially a multiple agent MDP.

There is a finite set of players $I = \{0, 1, ..., N\}$. The agents take actions $a^i_t \in A^i \subset \mathbb{R}^{D_A}$. We define the action profile as $a_t = (a^i_t)_{i \in I}$ and the action space $A = A^0 \times A^1 \times ... \times A^N$. The state of the game s_t, can include both agent specific states $s^i_t \in S^i \subset \mathbb{R}^{D_S}$ and a global state $s^g_t \in S^g \subset \mathbb{R}^{D_G}$, the state space is $S = S^0 \times S^1 \times ... \times S^N \times S^g$.

Each agent has reward function $R_i : S \times A \rightarrow \mathbb{R}$ and as before we have a transition function $T : S \times A \times S \rightarrow [0, 1]$ and discount rate β. The agents problem is to choose a^i_t to maximise their expected discounted reward

$$\max_{\{a^i_t\}^{\infty}_{t=0}} E \left\{ \sum_{t=0}^{t=\infty} \beta^t R_i(a_t, s_t) \right\}$$

given s_0, R_i, T, a^{-i}_t and $a^i_t \in \Gamma_i(s_t) \subset A^i$.

We define player policy functions $f_i : S \rightarrow A_i$ and the policy profile function $f : S \rightarrow A$. For any policy profile each player has a value function $V^f_i : S \rightarrow \mathbb{R}$ defined by

$$V^f_i(s) = E \left\{ \sum_{t=0}^{\infty} \beta^t R((f(s_t), s_t) | s_0 = s \right\}$$

Stochastic games were first introduced by Shapley (1953), who showed that a two player zero-sum stochastic game could be solved by value iteration. Some of the main economic applications of stochastic games have been in industrial organisation, particularly models of oligopoly with investment and firm entry and exit (Ericson and Pakes 1995).

Stochastic games have also been applied to common pool resource extraction problems (see for example Levhari and Mirman 1980). Recent applications have included fisheries (Kennedy 1987), groundwater (Negri 1989, Rubio and Casino 2001, Burt and Provencher 1993) and even surface water reservoirs (Ganji et al. 2007). Stochastic games have also been applied to commodity storage problems (Murphy et al. 1987, Rui and Miranda 1996).

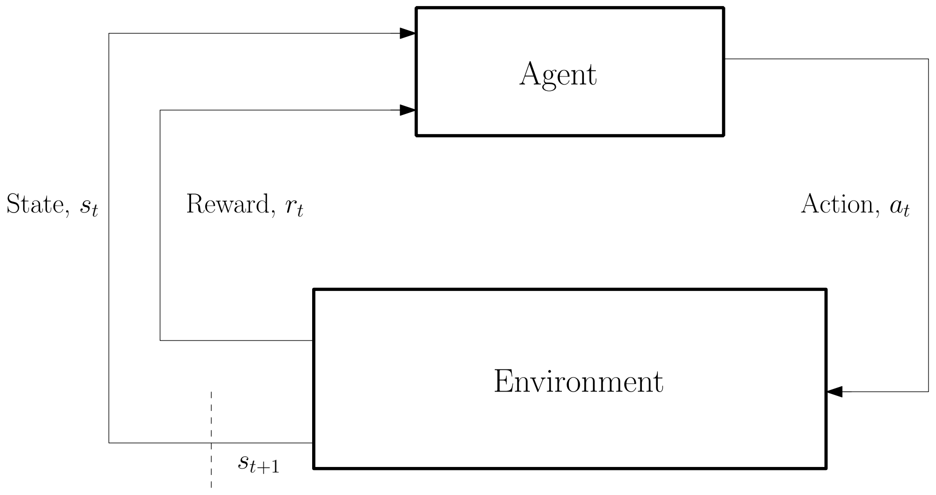
3 Single agent reinforcement learning

Reinforcement Learning is a sub-field of machine learning concerned with solving MDPs. For a detailed introduction see Sutton and Barto (1998), Bertsekas and

Tsitsiklis (1995) or Weiring and Otterlo (2012).

Central to reinforcement learning are so called 'model free' approaches: where the transition and payoff functions are assumed unknown. Here the agent must learn only by observing the outcomes — the reward and state transition — of its interactions with the environment (figure 1). Learning a good policy requires some degree of 'exploration': that is, testing a range of actions in each state.

Figure 1: Reinforcement learning



While historically related, the reinforcement learning methods sometimes applied in repeated games (see for example Erev and Roth 1998) are distinct from the methods we refer to here, as they involve estimating value functions via the Bellman principle.

Reinforcement learning has some computational advantages over dynamic programming, particularly in larger problems. As a simulation method, attention is limited to realised state combinations (see Judd et al. 2010), rather than a regular (tensor product) grid over the state space. Since, state variables are often correlated this can greatly reduce the complexity of the problem (Judd et al. 2010).

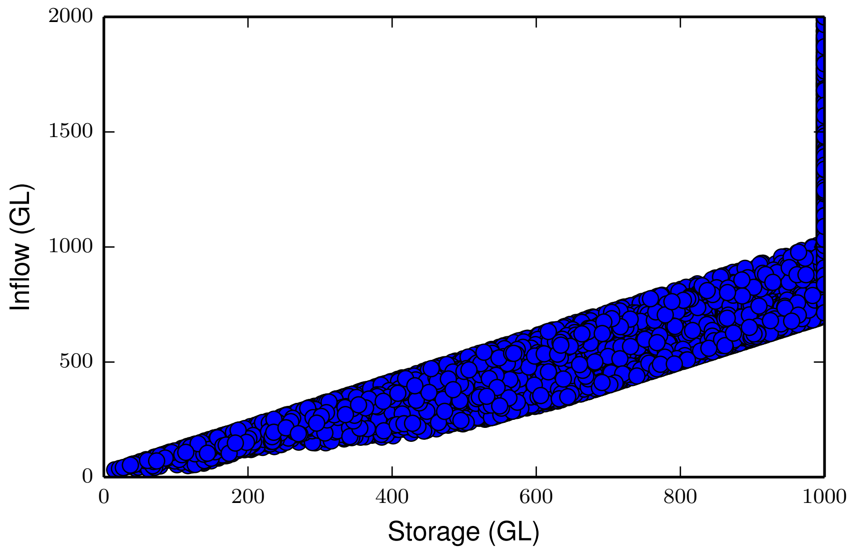
For example, figure 2 shows 10,000 simulated state points (storage S_t by inflow I_t) for a water storage problem (see section 5). Note that all inflows above the storage capacity of 1000 GL are associated with storage of 1000.

Reinforcement learning methods can also be easy to parallelize and generally provide greater flexibility to trade-off computation time and accuracy.

3.1 Q-learning

Q-learning (Watkins and Dayan 1992) is the canonical 'model free' reinforcement learning method. Q-learning works on the 'state-action' value function $Q : S \times$

Figure 2: Planner's storage problem, sample state points



$A \to \mathbb{R}$, defined as the present value payoff from taking action a in state s and following an optimal policy thereafter

$$Q^*(a,s) = R(s,a) + \beta \int_S T(s,a,s') \max_a Q^*(a,s')\ ds'$$

Once in possession of Q^*, we can compute an optimal (aka greedy) policy without the payoff and transition functions

$$f^*(s) = \arg\max_a Q^*(a,s)$$
$$\max_a Q(a,s) = V^*(s)$$

In standard Q-learning we incrementally update the Q function by observing state-action transitions $\{s_t, a_t, r_t, s_{t+1}\}$. For a discrete state and action space, the algorithm operates as follows:

Algorithm 1: Q-learning with discrete state and actions

```
1 Initialise $Q$, $s_0$
2 for $t = 0$ to $T$ do
3     select $a_t \in A$ ;                    // from some exploration policy
4     simulate $(a_t, s_t)$ and observe $r_t$ and $s_{t+1}$
5     set $Q(a_t, s_t) = (1 - \alpha_t)Q(a_t, s_t) + \alpha_n\{r_t + \beta \max_a Q(a, s_{t+1})\}$
6 end
```

The actions a_t must be selected according to an 'exploration' (partially randomised) policy. A simple option is an ϵ-greedy policy: a random policy with probability ϵ and an optimal policy otherwise. The choice of exploration policy involves an exploration-exploitation trade-off: a highly random policy will provide good coverage of the state-action space, but risks spending too much time at irrelevant points.

$\alpha_t \in (0, 1)$ is known as the learning rate. α may be constant, but more commonly follows a decreasing schedule. Watkins and Dayan (1992) shows (for discrete state and actions) that Q-learning converges as $t \to \infty$, subject to conditions over the exploration policy and learning rate α_t.

Q-learning can be extended to the continuous state and action case through function approximation. However, this typically voids convergence guarantees. Further, Q-learning is known to be unreliable (prone to spectacular divergence) in the continuous case (Weiring and Otterlo 2012).

3.2 Fitted Q iteration

Fitted Q iteration (Riedmiller 2005, Ernst et al. 2005) is a batch algorithm. First, a simulation is run for T periods (again using an exploration policy). Then a series of Q function updates is applied to the accumulated set of state-action samples (see Algorithm 13).

The approach has two main advantages: *data efficiency* since all samples are stored and reused and *stability* since Q functions can be fit to large samples. Further, it has some attractive properties in multiple agent problems (Weiring and Otterlo 2012).

Algorithm 2: Fitted Q Iteration (continuous state and action)

```
initialise $s_0$
for $t = 0$ to $T$ do                    // Simulate the system for $T$ periods
    select $a_t \in A$ ;                 // from some exploration policy
    simulate $(a_t, s_t)$
    store the sample $(s_t, a_t, r_t, s_{t+1})$
end
initialise $Q(a_t, s_t)$
repeat                                   // Iterate until convergence
    for $t = 0$ to $T$ do
        set $\hat{Q}_t = r_t + \beta. \max_a .Q(a, s_{t+1})$
    end
    estimate $Q$ by regressing $\hat{Q}_t$ against $(a_t, s_t)$
until a stopping rule is satisfied;
```

The separation of simulation and the fitting stages provides also additional flexibility. Firstly, it allows any type of parametric or non-parametric regression (supervised learning) model to be applied. Secondly, it facilitates parallel computing, since the simulation stage is so called embarrassingly parallel.

The success of fitted Q iteration depends crucially on the type of function approximation schemes employed. A variety of schemes have been used in the literature, including random forests (Ernst et al. 2005), neural networks (Riedmiller 2005) and tile coding (Timmer and Riedmiller 2007). Similar to continuous dynamic programming, the algorithm is guaranteed to converge for certain classes (i.e., non-expansive) function approximators (Ernst et al. 2005) (see Section **??**).

3.3 Fitted Q-V iteration

In noisy economic problems large samples T may be required. In this case optimising Q for each state point s_t may be an unnecessary burden (especially in the multi-agent case). One option is to optimise over a representative subset of state points, then estimate a continuous state value function V (Algorithm 18).

Algorithm 3: Fitted Q-V iteration

```
1  initialise $s_0$
2  for $t = 0$ to $T$ do                          // Simulate the system for $T$ periods
3      select $a_t \in A$ ;                          // from some exploration policy
4      simulate $(a_t, s_t)$
5      store the sample $(s_t, a_t, r_t, s_{t+1})$
6  end
7  initialise $Q(a_t, s_t)$
8  repeat                                          // Iterate until convergence
9      select a subset $\{s_k\}_{k=1}^{K} \subset \{s_t\}_{t=0}^{t=T}$
10     for $k = 0$ to $K$ do
11         set $\hat{V}_k = \max_{a_k} .Q(a_k, s_k)$
12     end
13     estimate $V(s_t)$ by regressing $\hat{V}_k$ on $s_k$
14     for $t = 0$ to $T$ do
15         set $\hat{Q}_t = r_t + \beta.V(s_{t+1})$
16     end
17     estimate $Q$ by regressing $\hat{Q}_t$ against $(a_t, s_t)$
18 until a stopping rule is satisfied;
```

3.4 Sample grids

A natural choice for our subset of state points $\{s_k\}_{k=1}^{K}$ is a sample of approximately equidistant points (i.e., a sample grid). Our starting point here is a simple distance based method (Algorithm 4), which provides a subset of points at least $\underline{r}$ distance apart. [1]. This method is very similar to the approach of Judd et al. (2010) [2].

[1]Note typically we scale all input data to the range [0, 1]

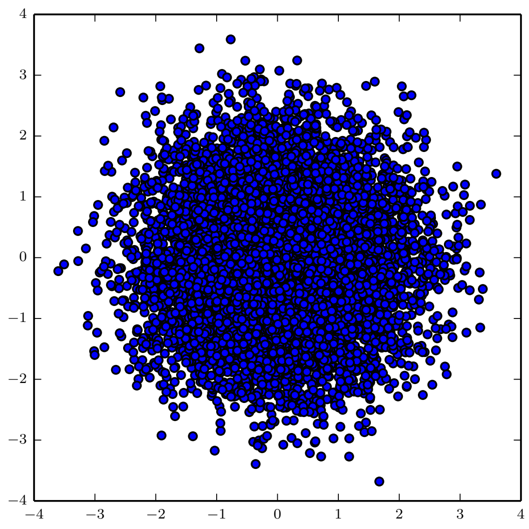
[2]However, our method also counts the sample points within the radius $\underline{r}$ of each point: in order to identify outliers. Judd et al. (2010) remove outliers by separately estimating a density function.

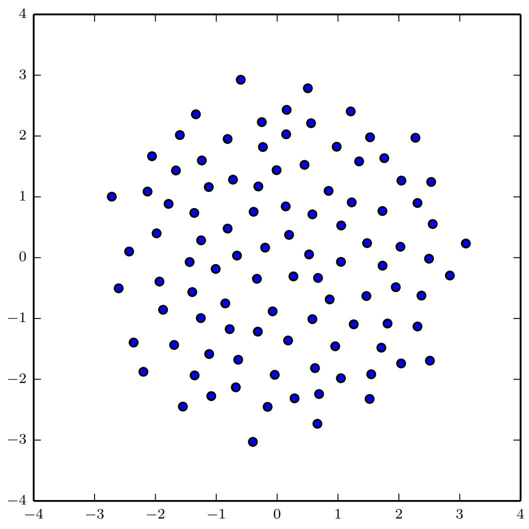
Figure 3 provides a demonstration in two dimensions. This method is sufficient for moderate sample sizes but can become inefficient for large dense data sets. One option is to add an early stopping condition, another is to employ some form of function approximation (i.e., tilecoding).

Algorithm 4: Selecting an approximately equidistant grid

```
set $J = 0$, $\mathbf{c}_0 = \mathbf{X}_0$, $n_0 = 0$
for $t = 0$ to $T$ do
    set $r_{min} = \infty$
    for $j = 0$ to $J$ do                    // Find the nearest center $\mathbf{c}_{j^*}$
        set $r = ||\mathbf{X}_t - \mathbf{c}_j||$
        if $r < r_{min}$ then
            set $r_{min} = r$
            set $j^* = j$
        end
    end
    if $r_{min} > \underline{r}$ then           // Add $\mathbf{X}_t$ as the next center $\mathbf{c}_j$
        set $j = j + 1$
        set $\mathbf{c}_j = \mathbf{X}_t$
        set $n_j = 0$
    else                                     // Increment counter for center $\mathbf{c}_{j^*}$ by 1
        set $n_{j^*} = n_{j*} + 1$
    end
end
```

Figure 3: An approximately equidistant grid in two dimensions


(a) 10000 iid standard normal points


(b) 100 points at least 0.4 apart

4 Function approximation

The success of batch reinforcement learning depends crucially on function approximation. In machine learning this is known as 'supervised learning': to 'learn' (estimate) a model for a 'target' (dependent) variable Y conditional a vector of 'input' (explanatory) variables $\mathbf{X}$, given only a set of 'training data' $\{Y_t, \mathbf{X}_t\}_{t=0}^{T}$. A demonstration of the problem is provided in figure 4.

The goal with function approximation is prediction: we want a model that can accurately predict Y given realisations of $\mathbf{X}$ outside our training sample. In attempting to minimise prediction error we face a bias-variance trade-off. A highly flexible model is at risk of 'overfitting' noisy data (figure 4a) while an inflexible model may lead to systematically biased predictions (figures 4b, 4e).

For our purposes, computation time is also important: both fitting (estimation) time and prediction (function call) time. In practice, subtle tradeoffs are faced between predictive power, fitting time and prediction time. Unfortunately, there is no general purpose method that achieves the optimal balance of all factors in all applications: there is no free lunch (Wolpert 1996).

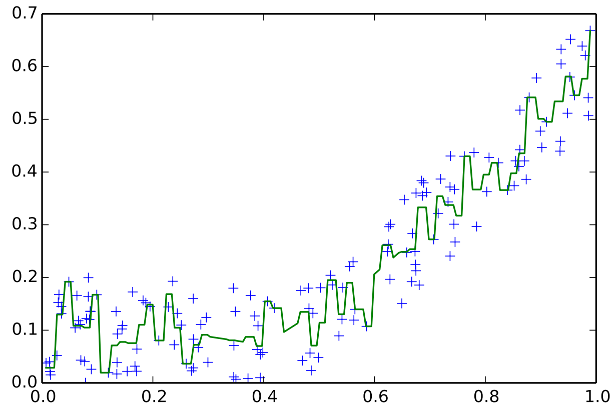
Our Fitted $Q - V$ iteration approach, poses two distinct approximation problems: A *big problem* (i.e., the Q function) and a *small problem* (i.e., the policy and value functions f, V). The features of these problems are summarised in table 1.

Table 1: Two approximation problems

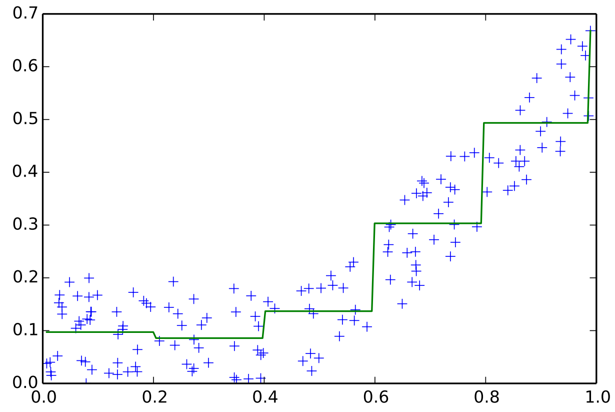
	Big Problem	Small problem
Sample size	Large (0.5-1 $\times$ 10^6)	Small (500-2000)
Input dimensions	Small (5)	Small (4)
Data structure	None	Gridded
Target noise	High	Low
Time constraint	Fitting	Prediction
Extrapolation important	No	Yes
Example	$Q(a_t, s_t)$	$f(s_t)$

For various reasons, standard methods employed in economics — such as orthogonal polynomials — are not ideal for these problems. Below we introduce the method of tile coding, which we use to approximate Q, V and f in all our water storage problems.

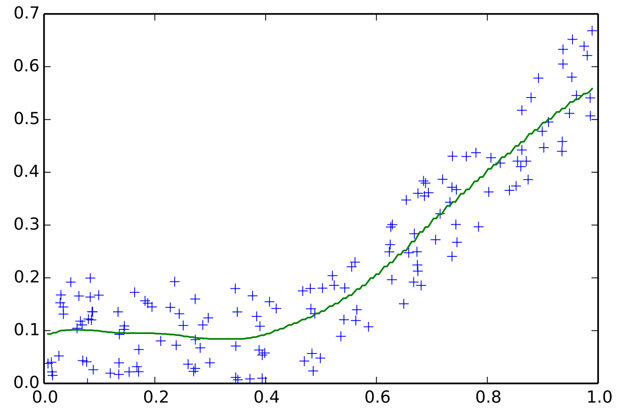
Figure 4: A comparison of function approximation methods in one dimension



(a) Tilecoding (TC-A) $N_L = 1, N_T = 60$

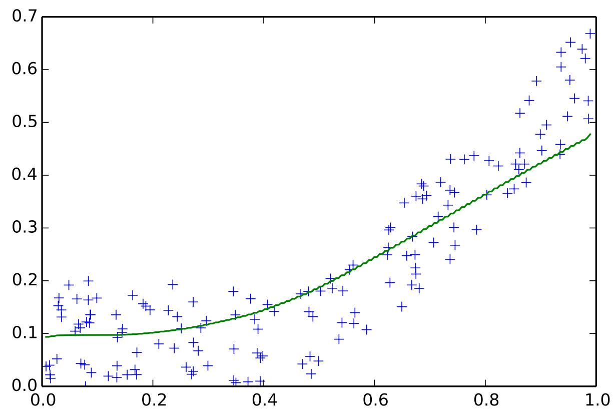


(b) Tilecoding (TC-A) $N_L = 1, N_T = 6$

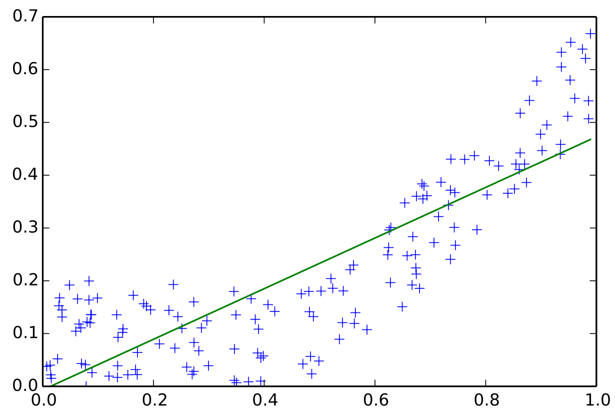


(c) Tilecoding (TC-SGD) $N_T = 4, N_L = 40$

a



(d) Tilecoding (TC-A) $N_T = 3, N_L = 40$



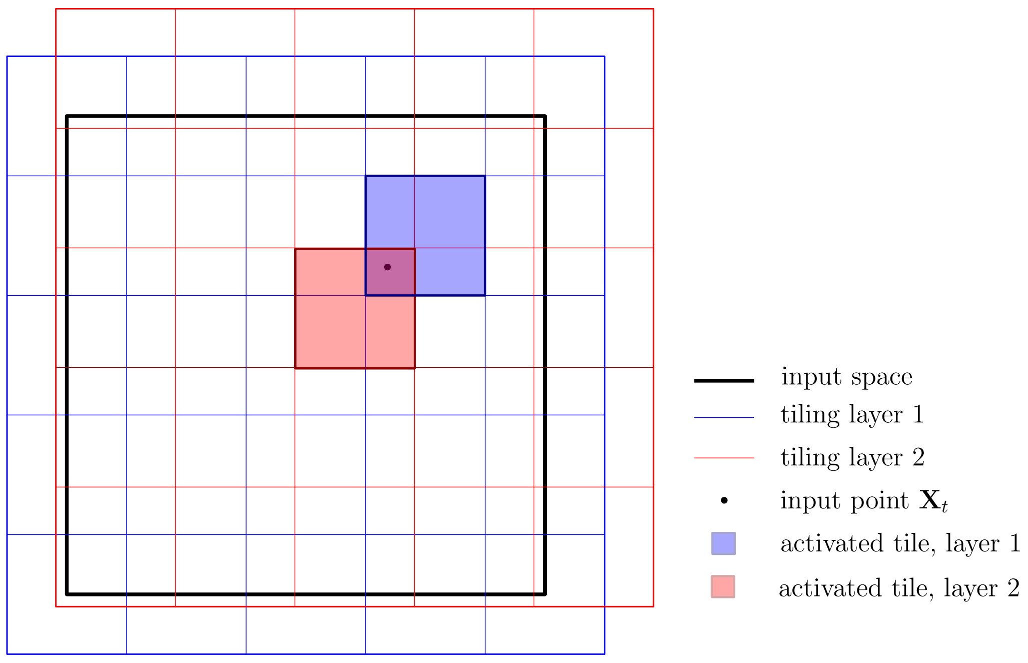
(e) Linear model

For various reasons, standard methods employed in economics — such as orthogonal polynomials — are not ideal for these problems. Below we introduce a method known as tile coding, which is particularly well suited to noisy problems in low (i.e., less than 10) dimensions.

4.1 Tile coding

Tile coding (Albus 1975, Sutton and Barto 1998) is a function approximation scheme popular in reinforcement learning. With tile coding, the input space is partitioned into *tiles*. A whole partition is referred to as a *tilling* or a *layer*. A tile coding scheme then involves multiple overlapping layers, each offset from each other according to some *displacement vector*.

Figure 5: Tile coding



Tile coding is best understood visually. In the simplest approach the tilings are just regular grids (i.e. rectangular tiles) and each grid is offset uniformly (i.e. diagonally) as in figure 5.

More formally, a tile coding scheme involves $i = 1$ to N_L layers. Each layer contains $j = 1$ to N_T binary basis functions

$$\phi_{ij}(\mathbf{X}) = \begin{cases} 1 & \text{if } \mathbf{X} \in \mathscr{X}_{ij} \\ 0 & \text{if } \textit{otherwise} \end{cases}$$

For each point $\mathbf{X}_t$ one tile j in each layer i is *activated* and our predicted value is the mean of the weights w_{ij} attached to the active tiles

$$Y = \frac{1}{N_L} \sum_{i=1}^{N_L} \sum_{j=1}^{N_T} w_{ij} \phi_{ij}(\mathbf{X})$$

Tile coding is a constant piecewise approximation scheme: the 'resolution' of the approximation depends on the number of layers and the number of tiles per layer. For example, figure 4a shows a tile coding scheme with $N_L = 1$ and $N_T = 6$. Just increasing the number of tiles gives a finer resolution but provides less *generalisation* leading to over fitting (see figure 4b). Figure 4c, shows a model with $N_L = 40$ and $N_T = 4$ which provides both high resolution and good generalisation.

Tile coding has some computational advantages schemes with basis functions of global support. Tile weights are stored in arrays and accessed directly (computing array indexes just involves integer conversion of **X**). Each function call then involves only the N_L active weights. As such, prediction time is low and grows linearly in the number of layers rather than exponential in the number of dimensions.

Function predict(**X**)

```
1 set $\hat{Y} = 0$
2 for $i = 0$ to $N_L$ do
3     $j = index(\mathbf{X}, i)$ ;            // returns index of active tile
4     $\hat{Y} = \hat{Y} + \frac{1}{N_L} w_{ij}$
5 end
6 return $\hat{Y}$
```

This speed gain comes at the cost of higher memory usage. However, since many reinforcement learning applications are CPU bound, this is an efficient use of resources. Memory limits are not an issue in any of our problems. In higher dimensions, weight arrays can be compressed through 'hashing'.

4.1.1 Fitting

The standard method of training the weights is SGD. An alternative, is to define each weight as a simple average, as in Algorithm 5.

Timmer and Riedmiller (2007) considers the use of tile coding in fitted-Q-iteration. When tile weights are simple averages, fitted-Q-iteration is guaranteed to converge on a unique fixed point (Timmer and Riedmiller 2007). A convergence result is possible, because this form of tile coding is a non-expansive approximator: that is a *smoother* or *averager* (see Stachurski 2008, Gordon 1995)[3].

[3]This form of tile coding is in fact closely related to other more common averaging methods such as k-nearest neighbours and random forests.

Algorithm 5: Fitting tile weights by averaging

```
for $t = 0$ to $T$ do
    for $j = 0$ to $N_L$ do
        $i = index(\mathbf{X}_t, j)$ ;          // returns index $i$ of active tile
        set $n_{ij} = n_{ij} + 1$ ;             // count tile sample size
        set $w_{ij} = w_{ij} + Y_{ij}$ ;        // calculate sum
    end
end
for $j = 0$ to $N_l$ do
    for $i = 0$ to $N_T$ do
        set $w_{ij} = w_{ij} / n_{ij}$ ;        // calculate mean
    end
end
```

While fitting by averaging provides a convergence guarantee it is unlikely to provide ideal performance: it will suffer badly from bias if the tiles are too wide (see figure 4d) and variance if the tiles are too small. An alternative approach is 'Averaged SGD' (ASGD) (Bottou 2010), where the weights are defined as the average of a single SGD pass over the data (Algorithm 6). .

Algorithm 6: Averaged Stochastic Gradient Descent (ASGD) - Tile coding

```
Initialise $w_{ij}$ by averaging (Algorithm 5)
set $\bar{w}_{ij} = 0$ for all $i, j$
for $t = 0$ to $T$ do                           // A single SGD pass
    set $\delta_t =$ predict $(\mathbf{X})$ $-Y_t$
    for $j = 0$ to $J$ do
        $i = index(\mathbf{X}_t, j)$
        set $w_{ij} = w_{ij} - \alpha_t \delta_t$
        set $\bar{w}_{ij} = \bar{w}_{ij} + w_{ij}$ ;    // sum the weight updates
    end
end
for $j = 0$ to $N_L$ do
    for $i = 0$ to $N_L$ do
        set $w_{ij} = \bar{w}_{ij} / n_{ij}$ ;          // calculate mean
    end
end
```

Bottou (2010) demonstrates the superiority of ASGD over SGD from problems with large samples. In Section 5 we show that fitting the Q function by ASGD achieves a performance gain over averaging with no loss of stability.

4.1.2 Big problems

For big problems (i.e., the Q function) we use tile coding with regularly spaced grids. We use the 'optimal' displacement vectors of (Brown and Harris 1994). Tile weights are fit by ASGD.

Tile coding can suffer from noise in regions where training data are sparse, so for input variables with tails, we limit the tiling to a percentile range of the training data (e.g., the 1st to 99th). We then pass on the job of extrapolating into the unrepresented parts of the input space to our policy and value functions. [4].

4.1.3 Small problems

For small problems we again use regular grids and optimal displacement vectors. The tile weights are fit either by averaging — for value functions — or standard SGD using averaging for starting values.

For extrapolation, we combine our tile coding scheme with a sparse linear spline model. The combined scheme replaces the tile code weight w_{ij} with the linear spline predicted value if $n_{ij} = 0$.

5 The planner's water storage problem

Here we apply fitted Q-V iteration to the planners storage problem and compare it to the benchmark of dynamic programming.

5.1 The problem

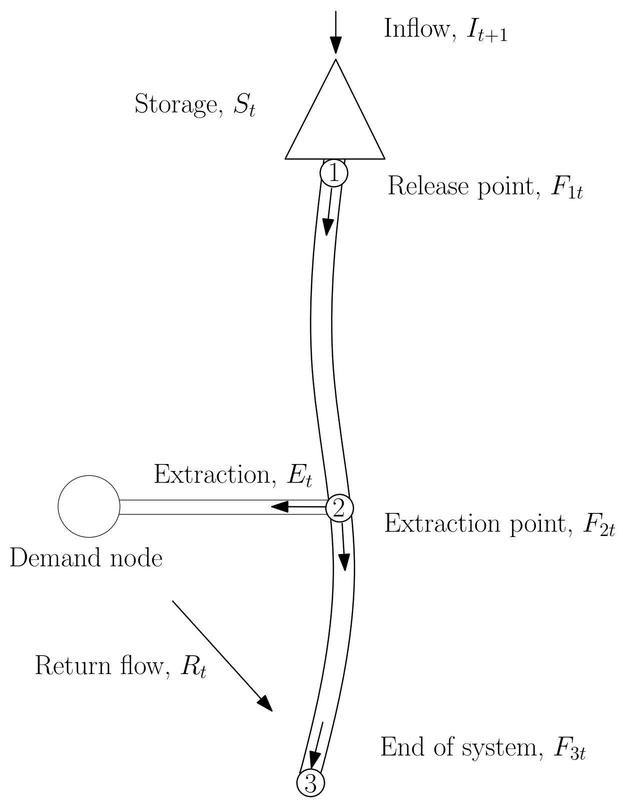
The problem is that of a planner managing a reservoir on a river. The reservoir receives stochastic water inflows makes storage releases to satisfy multiple water users (i.e., irrigation farmers) located at a single demand node.

The planners problems is to maximise the benefits from water use, subject to the hydrological (water supply) constraints:

$$\max_{\{W_t\}_{t=0}^{t=\infty}} E\left\{\sum_{t=0}^{\infty} \beta^t \Pi(Q_t, I_t)\right\}$$

[4]There are a number of other more complex options here. One is the idea of 'adaptive tile coding' where the tile sizes are endogenous and may for example be larger in regions with less data (see Whiteson et al. 2007). Another is to apply a non-linear scaling to the input data to make it more uniformly distributed.

Figure 6: A simple river system



Subject to:

$$S_{t+1} = \min\{S_t - W_t - \delta_0 \alpha S_t^{2/3} + I_{t+1},\, K\}$$

$$0 \leq W_t \leq S_t$$

$$Q_t \leq \max\{(1-\delta_{1b})W_t - \delta_{1a}, 0\}$$

where Π is a concave payoff function (i.e., irrigation profit function), Q_t is water use, I_{t+1} is the stochastic storage inflow (following an AR-1 process), S_t the storage volume, W_t the storage withdrawals, K is the fixed storage capacity, $\delta_0, \delta_{1a}, \delta_{1b}, \alpha$ are all loss (i.e., evaporation) parameters and β is the discount rate. Further detail on parameter assumptions is contained in my thesis.

An important feature of the problem are storage 'spills': where the storage reaches capacity and further inflows are 'lost' (i.e. flow uncontrolled downstream) we define spills Z_t as

$$Z_t = \max\{I_{t+1} - (K - S_t - W_t - \delta_0 \alpha S_t^{2/3}), 0\}$$

5.2 The solution

For fitted *Q*-*V* iteration we adopt a two stage 'growing batch' approach. We simulate an initial batch of samples assuming uniform (i.e., uninformed) exploration:

$$W_t = \epsilon_t . S_t$$
$$\epsilon_t \sim U[0,1]$$

After running fitted *Q*-*V* on the first first batch we add a second batch of samples, this time with Gaussian exploration:

$$W_t = \min\{\max\{\hat{f}(S_t, \tilde{I}_t) + \epsilon_t S_t, 0\}, S_t\}$$
$$\epsilon_t \sim N(0, \delta)$$
$$0 < \delta < 1$$

where $\hat{f}$ is the policy function obtained from the first batch.

Algorithm 4 is used to build a grid of state points with $\underline{r} = 0.02$. Tilecoding is used to approximate Q, V and f. We test fitting the Q function both by averaging (TC-A) with ASGD (TC-ASGD).

For dynamic programming we employ fitted policy iteration and use tile coding to approximate V over a 35×35 grid of the state space. We begin both methods with the initial guess $V(\mathbf{X}) = 0$.

Both methods are coded predominantly in cython. Both make user of parallelization and run on a standard 4-core i7 desktop. Mean welfare, storage, and solution time are shown in tables 2, 3, 4 (each the average of 10 runs).

Fitted *Q*-*V* iteration obtains a policy comparable with SDP in less time. The fact that fitted *Q*-*V* iteration compares well for trivial single agent problems, suggests significant gains (in computation time) may be achievable in larger problems.

Table 2: Mean social welfare

	5000	10000	20000	50000	80000
Myopic	181.4	181.4	181.4	181.4	181.4
SDP	186.6	186.6	186.6	186.6	186.6
TC-A	185.4	185.5	185.8	185.9	186.0
TC-ASGD	185.7	185.9	185.9	186.2	186.3

Reinforcement learning achieves welfare levels up to 99.8 per cent of the SDP solution and results in similar mean storage levels.

Figure 7: Performance of fitted Q-V iteration, planner's storage problem

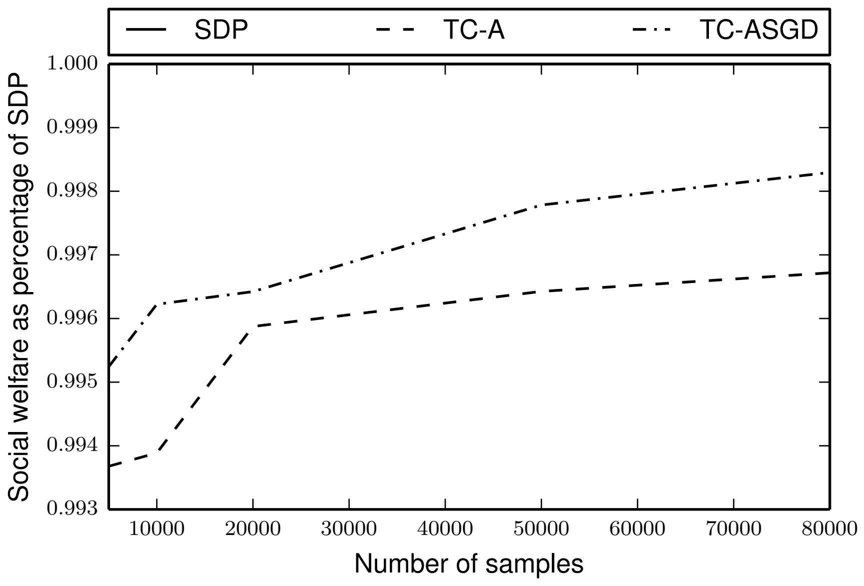


Table 3: Mean storage

	5000	10000	20000	50000	80000
Myopic	577.4	577.4	577.4	577.4	577.4
SDP	697.8	697.8	697.8	697.8	697.8
TC-A	691.5	686.2	686.9	685.8	690.5
TC-ASGD	696.6	704.7	700.8	710.8	699.6

Table 4: Computation time

	5000	10000	20000	50000	80000
SDP	6.6	7.2	7.5	7.4	7.4
TC-A	0.4	0.4	0.5	0.6	0.8
TC-ASGD	0.4	0.6	0.9	1.3	1.9

For a given sample size ASGD outperforms averaging. Note that in this trivial example, TC-A still performs well on a computation time basis, because fitting time is longer with ASGD. However in more complex problems — where simulation is more time intensive — the gains from ASGD become more important.

6 Equilibrium in stochastic games

6.1 Markov Perfect Equilibrium

A natural equilibrium concept for stochastic games is a 'Markov perfect equilibrium' (MPE) (Maskin and Tirole 1988). An MPE is defined by a set of markovian policies functions $\{f_0, f_1, ..., f_N\}$ which simultaneously solve each agents' problem, forming a sub-game perfect Nash equilibrium.

MPE existence results have been established for specific classes of stochastic games as early as Shapley (1953). These early results typically assume finite state space or time horizons and mixed strategies. A general (pure strategy, infinite horizon and state space) existence result has remained elusive despite much recent attention (Duggan 2012, Escobar 2011, Balbus et al. 2011, Horst 2005, Amir 2005).

Broadly, MPE existence results involve two steps: one, show that for any feasible set of opponent policies f^{-i} the agents' problems have unique solutions $V_i^*(s)$; two, show that the static 'stage game', with payoff functions $\pi_i(a, s, V_i(s))$

$$\pi_i(a, s, V_i(s)) = R(s, a) + \beta \int_S T(s, a, s') V_i(s') \, ds'$$

has a Nash equilibrium for any $s \in S$ and any feasible set of V_i.

Recent existence results all rely on particular regularizing assumptions. For example Escobar (2011) adopts an assumption of concave reduced payoffs: concavity of π_i with respect to a. Horst (2005) relies on a weak interaction condition: that players utility is affected more their own actions than by all other players actions. Amir (2005) applies the lattice theory concepts of supermodularity and increasing differences. Recently Duggan (2012) proves existence of MPE where the transition functions are subject to a particular form of noise.

In general, uniqueness of equilibra in stochastic games is not guaranteed. As demonstrated by Doraszelski and Satterthwaite (2010) multiple equilibira are commonly observed in the Ericson and Pakes (1995) style models. The uniqueness of equilibria is usually considered numerically, by testing invariance to starting values. Although standard algorithms are not guaranteed to locate all possible equilibria (Borkovsky et al. 2008).

6.2 Oblivious Equilibrium

A general problem with stochastic games is that the dimensionality of the state space, $D_S^n \times D_G$, can be too large, particularly when n is large and $D_S > 0$. Further the assumption that the agents have information on all opponent state variables becomes unrealistic.

Oblivious Equilibrium (OE) is an alternative to MPE in the case were n is large (Weintraub et al. 2008). Here opponent state variables are replaced with relevant summary statistics, such that the state space for the agents' problems is condensed to $S^i \times S^G$.

Weintraub et al. (2008) shows that under certain conditions (a large number of similarly sized firms) OE approximates MPE for oligopoly type models. This result is generalised to a broader class of dynamic stochastic games by Abhishek et al. (2007). In context of oligopoly models opponent state variables are replaced with their long-run average means. Weintraub et al. (2010) extend OE to models with aggregate shocks, here opponent states are replaced with their mean conditional on the aggregate shock.

While uniqueness of OE is not guaranteed Weintraub et al. (2008) find no examples of multiple equilibria in applied problems, and argue that in general OE is likely to involve fewer equilibria than MPE.

7 Learning in games

7.1 Learning in repeated games

The theory of learning in games describes how less than fully rational agents adapt in response to observed past play. There is much economic literature on learning in repeated games: testing how closely learning models reflect human behaviour in experiments (for example Erev and Roth 1998) and establishing if and when they converge on equilibrium in models (see Fudenberg and Levine 1998).

Learning models are particularly relevant for games with large numbers of agents and 'aggregate statistics' where "players are only trying to learn their optimal strategy, and not to influence the future course of the overall system (Fudenberg and Levine 2007; pp. 3). In the most general learning models, the population of agents can have 'heterogeneous beliefs', so that identical agents may play differing policies.

The oldest learning model is *fictitious play*: where each agent plays a best response to the empirical distribution of past play. A related model is the *partial best response* dynamic (Fudenberg and Levine 1998): where a sample of users play a best response to the previous periods play. Fudenberg and Levine (1998) show that for repeated games, these two models have the identical asymptotic properties.

Another model popular in economics is 'reinforcement learning' — here we refer to the 'foresight-free' methods, not the machine learning methods which are the focus of this paper. Here agents maintain a probability distribution over actions, with actions that result in higher payoffs gradually receiving higher probabilities. Erev and Roth (1998) show that such simple rules closely match the behaviour of humans in experiments.

7.2 Multiple agent learning

Unfortunately, the economic literature on learning in stochastic games is surprisingly scarce (see Fudenberg and Levine 1998). Here we turn to multiple agent learning: a relatively young but rapidly expanding field at the intersection of game theory and machine learning / artificial intelligence (Shoham et al. 2007, Fudenberg and Levine 2007, Busoniu et al. 2008). Here concepts of equilibrium and learning in repeated games meet reinforcement learning algorithms for single agent MDPs.

While reinforcement learning methods are designed for artificial 'software agents', they have a foundation in human and animal behavioural psychology and neuroscience (Weiring and Otterlo 2012). Putting aside this scientific 'inspiration', reinforcement learning provides a set of mature algorithms for representing agents who optimise subject to limited information and computational resources.

An obvious starting point for stochastic games, is to allow each agent to follow a single agent algorithm. In this case the behaviour of the other agents becomes part of the environment that needs to be learned[5].

This type of multi-agent Q-learning has been applied widely in computer science domains, with some success (Busoniu et al. 2008). In the multi-agent context their are no convergence guarantees (as the environment is no longer stationary) and the convergence properties have been subject to limited study (Busoniu et al. 2008).

To date reinforcement learning has received little attention from economists:

> From the perspective of economists, Q-learning and other procedures that use generalizations of reinforcement learning to estimate value functions in environments with a state variable have not been well-studied....It may be that considering Q-learning in the multiple-agent

[5]In stochastic games incremental Q-learning or 'actor-critic' methods with 'soft-max' exploration would be a natural analog to the Erev and Roth (1998) type methods used in repeated games

> case where players simultaneously try to calculate value function will lead to important new insights (Fudenberg and Levine 2007; pp. 6)
>
> Surprisingly, the predictions of Q-learning models have yet to be compared with data from controlled laboratory experiments with human subjects - a good topic for future research (Tesfatsion and Judd 2006; pp. 979).

Economic applications of Q-learning are rare and the few examples (Tesauro and Kephart 2002, Kutschinski et al. 2003) apply Q-learning to repeated games (oligopoly price / quantity competition) rather than stochastic games.

8 Multiple agent fitted Q-V iteration

8.1 The algorithm

Here we present a multi-agent version of fitted Q-V iteration (Algorithm 24). In essence, the method combines our single agent algorithm with two smoothing dynamics (i.e., learning models). Similar to repeated games, a non-smoothed application of batch reinforcement learning (i.e., computing optimal policies for all agents then repeating) will be unstable and prone to cycles.

One option is a partial best response dynamic. Within each simulation stage the environment is stationary (the users' policy functions are fixed) so we can compute optimal (best response) policies using fitted Q-V iteration, assign these to a random sample of the population and repeat. The other option is some form of fictitious play. Here all users would take the optimal polices each stage, but new sample batches would be combined with the existing batch of samples (similar to the growing batch approach in section 5)

Our general multi-agent algorithm (algorithm 24) combines both of these types of smoothing. The method permits much flexibility. With high K and $\lambda = 1$ we have a partial best response dynamic, with low λ, low K it approaches an 'on-line' reinforcement learning method. Our preferred approach (detailed in the next section) is a comprise between these extremes.

8.2 Interpreting the method

This method is to be interpreted firstly as a learning algorithm. Within the computer science literature, this represents 'rational' agent learning (Bowling and

Algorithm 7: Multiple agent fitted Q-V iteration

```
simulate an initial batch of samples $\{s_t, a_t, r_t, s_{t+1}\}_{t=0}^{T}$
store the samples in arrays $\mathbf{s}, \mathbf{s}_{+1}, \mathbf{a}$ and $\mathbf{r}$
initialise $f_i$, $V_i$, $s_0$, $Q_i$
for $J$ iterations do
    for $t = 0$ to $\lambda T$ do                    // Simulate for $\lambda T$ periods
        select $a_t$ using policies $f_i$ with exploration
        simulate $(a_t, s_t)$
        store the new samples $\{s_t, a_t, r_t, s_{t+1}\}$
    end
    replace $\lambda T$ samples of $\mathbf{s}, \mathbf{s}_{+1}, \mathbf{a}$ and $\mathbf{r}$ with new samples
    for $K$ iterations do
        select a subset of state points $\tilde{\mathbf{s}} \subset \mathbf{s}$
        for $i \in I$ do
            set $\hat{\mathbf{Q}}^i = \mathbf{r}^i + \beta.V^i(\mathbf{s}_{+1})$
            update $Q_i$, by regressing $\hat{\mathbf{Q}}^i$ on $(\mathbf{a}^i, \mathbf{s})$
            set $\hat{\mathbf{V}}^i = \max_a Q^i(a, \tilde{\mathbf{s}})$
            update $V_i$ by regressing $\hat{\mathbf{V}}^i$ on $\tilde{\mathbf{s}}$
        end
    end
    for $i \in I^u$ do                    // update policy functions for $I^u \subset I$
        set $\hat{\mathbf{a}}^i = \arg\max_a Q^i(a, \tilde{\mathbf{s}})$
        update $f_i$ by regressing $\hat{\mathbf{a}}^i$ on $\tilde{\mathbf{s}}$
    end
end
```

Veloso 2001): learning that converges on best response policies given stationary opponent policies. Within the economic literature, the approach might be described as an 'optimisation-based' learning method (Crawford 2013).

At the same time, the approach is only a small departure from algorithms used to compute rational expectations equilibria. For example, the approach is similar to methods used to solve stochastic games for MPE, including the value iteration method of Shapley (1953) and the simulation based approach of Pakes and McGuire (2001). It is also related to the Krusell and Smith (1998) style algorithms used to solve macro heterogeneous agent models, where user problems are solved by dynamic programming and user interactions are estimated through simulation.

While our approach is clearly related to the field of Agent Based Computational Economics (ACE), there are some important differences. In practice, agent based models tend to rely on genetic algorithms / replicator dynamics and almost never make use of reinforcement learning methods (although they are considered in Tesfatsion and Judd 2006).

The differences between ACE and multi-agent learning reflect their respective origins. Agent based modeling comes from the physical sciences, with a focus on the aggregate dynamics resulting from very large numbers of simple agents. A typical example being animal herds. Multi agent learning comes from an engineering / computer science background. Here the focus is on how a moderate number of intelligent agents can develop good (optimal) policy rules. With typical examples being the interaction of robots / vehicles in logistics or defence.

9 The decentralised storage problem

Below we detail the application of fitted Q-V iteration to a decentralised version of the water storage problem.

9.1 The problem

The decentralised problem is defined in detail my thesis, below is a brief outline.

In the decentralised model their are a large number of water users $i = 1$ to n. Each user holds a property right to the reservoir enabling them to make their own storage / withdrawal decisions. These rights operate as 'water accounts'. Each period these accounts are credited with a share λ_i of inflow and debited for user withdrawals w_{it}. The evolution of user account balances s_{it} follows the general form

$$s_{it+1} = \min\{\max\{s_{it} - w_{it} - l_{it} + \lambda_i I_{t+1} + x_{it+1}, 0\}, k_{it}\}$$

$$w_{it} \leq s_{it}$$

$$\sum_{i=1}^{n} \lambda_i = 1, \quad \sum_{i=1}^{n} s_{it} = S_t, \quad \sum_{i=1}^{n} l_{it} = \delta_0 \alpha S_t^{2/3}$$

where l_{it} are user storage loss deductions, k_{it} are user account limits and x_{it} are the 'storage externalities'. Intuitively x_{it} are account reconciliations, which ensure the total account balance $\sum_{i=1}^{n} s_{it}$ matches the physical storage volume S_t.

Here just note that x_{it} can be a rather complicated function of the storage balances and withdrawals of all users $\boldsymbol{s}_t = (s_{1t}, s_{2t}, ..., s_{nt})$ and $\boldsymbol{w}_t = (w_{1t}, w_{2t}, ..., w_{nt})$ as well as quantities S_t, W_t, L_t and I_t.

In addition to making withdrawal decisions users can also engage in a water spot market, trading their water 'allocations': withdrawals adjusted for delivery losses. The users problem is to choose withdrawals w_{it} and water use q_{it} to maximise private welfare, which is the sum of profits from water use $\pi_i(q_{it}, I_t, e_{it})$ and net proceeds from water trading $R(w_{it}, q_{it}, P_t, \tau)$

$$\max_{\{w_{it}\}_{t=0}^{t=\infty}} E\left\{\sum_{t=0}^{\infty} \beta^t \left(\pi_i(q_{it}, I_t, e_{it}) + R(w_{it}, q_{it}, P_t, \tau)\right)\right\}$$

subject to the water accounting constraints, the behaviour of the other agents and the physical constraints. Here e_{it} is a user specific productivity shock, P_t is the market price and τ the transaction cost.

The version referred to below involves 100 agents, divided into two groups of 50 the 'low reliability' users (with inelastic water demands) and the 'high reliability' users (with elastic demands).

The purpose of the model is to test different property rights schemes. Here a property rights system is defined by the specification of l_{it}, k_{it} and x_{it}. For example, we compare the case of storage property rights, $k_{it} = \lambda_i K$, with an open access outcome, $k_{it} = K$, in which users are not accountable for the effect they have on spills.

9.2 The solution

We begin the process by solving the planner's problem by SDP, and using the planners solution to derive guesses for the user policy functions. We then solve the high and low reliability users problems (holding opponent policies fixed) by single agent fitted Q-V iteration. This yields an initial batch of T samples and estimates $\hat{f}_i, \hat{v}_i$ of the policy and value functions. We then proceed to the full multi-agent algorithm as outlined above.

We use $J = 25$ major iterations and $K = 1$ value iterations and $\lambda = 0.10$. After each major iteration we update policies for a 20 per cent random sample of agents. From this sample of agents we select a subsample to become 'explorers'. We adopt Gaussian exploration:

$$w_{it} = \min\{\max\{\hat{f}_i(s_{it}, S_t, e_{it}, \tilde{I}_t) + N(0, \delta_t.s_{it}), 0\}, s_{it}\}$$
$$0 < \delta_t < 1$$

The number of explorers and range of exploration declines over the 25 iterations, from 10 explorers (5 per user group) and $\delta = 0.25$ to 4 explorers and $\delta = 0.085$.

We pool the samples from the respective user groups (the low and high reliability users). We begin with T of 100,000, which gives 500,000 samples for each of our user groups. We optimise the Q function for both groups over a sample grid of state points with radius 0.045. Tile coding is used to approximate the policy and value functions and the Q function (fit by ASGD).

9.3 Some results

Below are some sample results just to demonstrate the performance of the method. Again, there's not enough space here to throughly explain these results.

Firstly, the method achieves a degree of convergence. While subject to some noise, changes in value and policy functions tend to diminish, rather than cycle. Importantly key model aggregate variables (prices, storage volumes, payoffs) show a high degree of stability. Note that, cycles are found to emerge in these problems if the degree of smoothing is insufficient.

Scenarios, that involve very few externalities (storage capacity rights CS) and which are expected (from theoretical results) to achieve close to optimal outcomes, closely match the planner's SDP solution. Scenarios with large externalities (such as open access - OA) result in expected changes in behaviour (above optimal storage) and welfare losses.

If anything the multi-agent solution tends to be more robust (i.e., achieves the same result when solved multiple times) than the single agent. While individual policy functions tend to display some noise, this averages out across large numbers of agents.

Figure 8: Mean storage by iteration

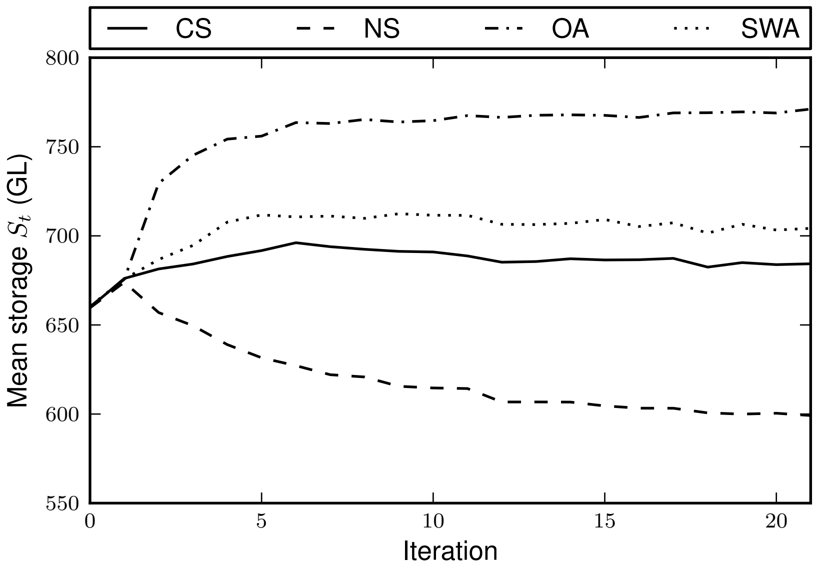
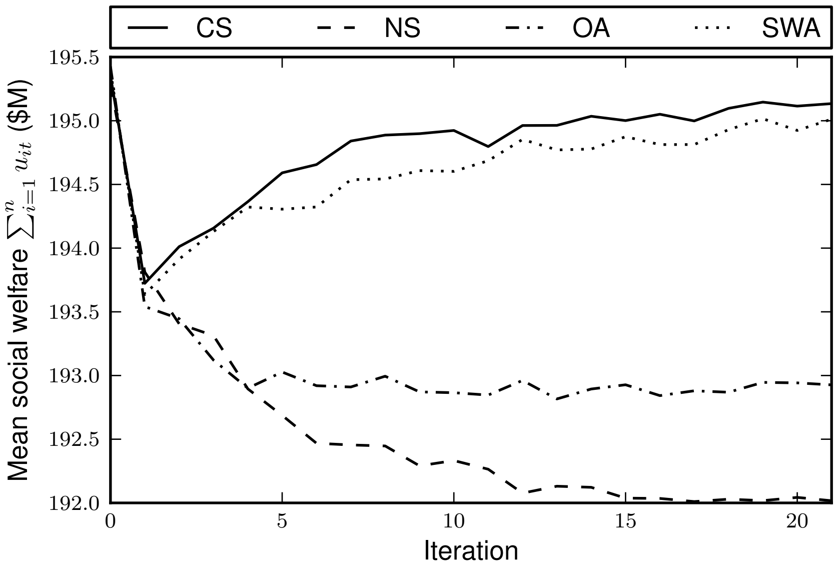


Figure 9: Mean social welfare by iteration



10 Conclusions

Reinforcement learning provides a mature set of algorithms for solving MDPs. While some practical problems are faced in adapting these methods to economic problems, they are not insurmountable.

Economic problems tend to be noisy, relatively 'smooth' (the value and policy functions tend to be smooth after averaging out the noise) and involve relatively small state spaces. The batch method of fitted Q-V iteration is well suited to this context. While large samples may be required, the method can be faster than dynamic programming, when combined with appropriate function approximation.

Here we find the method of tile coding ideal. Tile coding can process large data sets much faster than alternatives relying on global basis functions. Tile coding may also be useful in other applications in economics including dynamic programming. In any more than around 10 dimensions the memory requirements of tile coding can become restrictive[6].

Our multi-agent reinforcement learning method provides a middle ground between Krusell and Smith (1998) style methods and agent based methods — both in terms of the size and complexity of the models it can be applied to and the degree of rationality or 'intelligence' assumed for the agents.

Reinforcement learning allows us to consider larger more complex multi-agent problems — including those with externalities – than can be attempted with traditional techniques. While, we may need to relax our notion of equilibrium (i.e., move away from rational expectations), we can hold tightly to the idea of individually maximising agents.

Clearly, the simulation and search methods of agent based economics provide maximum flexibility in terms of the size and complexity of models that can be considered. However, agent based methods often rely on simple behavioural rules (which may then replicate based on success). With reinforcement learning, we can have more realistic 'intelligent' agent behaviour.

The field of multi-agent learning is still relatively young. There is much debate within and between disciplines, on best notion of equilibrium and how much emphasis to place on it. While techniques are continuously evolving, the method of multi-agent fitted Q-V iteration presented here, provides a practical starting point for economic problems.

[6]In large state spaces we can either turn to tree based approximation methods such as 'random forests', or consider some form of state 'abstraction' / feature selection method to omit or aggregate the least relevant state variables

References

Abhishek, V., Adlakha, S., Johari, R. and Weintraub, G. 2007, Oblivious equilibrium for general stochastic games with many players, *in* 'Allerton Conference on Communication, Control and Computing'.

Albus, J. S. 1975, 'A new approach to manipulator control: The cerebellar model articulation controller (CMAC)', *Journal of Dynamic Systems, Measurement, and Control*, vol. 97, no. 3, pp. 220–227.

Amir, R. 2005, Discounted Supermodular Stochastic Games: Theory and Applications.

Balbus, L., Reffet, K. and Wozny, L. 2011, A Constructive Study of Markov Equilibria in Stochastic Games with Strategic Complementarities, *in* 'Conference on Theoretical Economics, University of Kansas'.

Bertsekas, D. and Tsitsiklis, J. 1995, Neuro-dynamic programming: an overview, *in* 'Decision and Control, 1995., Proceedings of the 34th IEEE Conference on', Vol. 1, IEEE, pp. 560–564.

Borkovsky, R., Doraszelski, U. and Kryukov, Y. 2008, 'A user's guide to solving dynamic stochastic games using the homotopy method'.

Bottou, L. 2010, Large-scale machine learning with stochastic gradient descent, *in* 'Proceedings of COMPSTAT'2010'.

Bowling, M. and Veloso, M. 2001, Rational and convergent learning in stochastic games, *in* 'International joint conference on artificial intelligence', Vol. 17, pp. 1021–1026.

Brown, M. and Harris, C. J. 1994, 'Neurofuzzy adaptive modelling and control'.

Burt, O. and Provencher, B. 1993, 'The externalities associated with the common property exploitation of groundwater', *Journal of Environmental Economics and Management*, vol. 24, no. 1, pp. 39–15K.

Busoniu, L., Babuska, R. and De Schutter, B. 2008, 'A comprehensive survey of multiagent reinforcement learning', *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, vol. 38, no. 2, pp. 156–172.

Crawford, V. P. 2013, 'Boundedly Rational versus Optimization-Based Models of Strategic Thinking and Learning in Games', *The Journal of Economic Literature*, vol. 51, no. 2, pp. 512–27.

Doraszelski, U. and Satterthwaite, M. 2010, 'Computable Markov-perfect industry dynamics', *The RAND Journal of Economics*, vol. 41, no. 2, pp. 215–243.

Duggan, J. 2012, 'Noisy stochastic games', *Econometrica*, vol. 80, no. 5, pp. 2017–2045.

Erev, I. and Roth, A. E. 1998, 'Predicting how people play games: Reinforcement learning in experimental games with unique, mixed strategy equilibria', *American economic review*, vol. pp. 848–881.

Ericson, R. and Pakes, A. 1995, 'Markov-perfect industry dynamics: A framework for empirical work', *The Review of Economic Studies*, vol. 62, no. 1, pp. 53–82.

Ernst, D., Geurts, P. and Wehenkel, L. 2005, 'Tree-based batch mode reinforcement learning', , vol. pp. 503–556.

Escobar, J. 2011, 'Equilibrium Analysis of Dynamic Models of Imperfect Competition'.

Fudenberg, D. and Levine, D. K. 1998, *The theory of learning in games*, Vol. 2, MIT press.

Fudenberg, D. and Levine, D. K. 2007, 'An economist's perspective on multi-agent learning', *Artificial intelligence*, vol. 171, no. 7, pp. 378–381.

Ganji, A., Khalili, D. and Karamouz, M. 2007, 'Development of stochastic dynamic Nash game model for reservoir operation. I. The symmetric stochastic model with perfect information', *Advances in water resources*, vol. 30, no. 3, pp. 528–542.

Gordon, G. J. 1995, Stable function approximation in dynamic programming, Technical report, DTIC Document.

Horst, U. 2005, 'Stationary equilibria in discounted stochastic games with weakly interacting players', *Games and Economic Behavior*, vol. 51, no. 1, pp. 83–108.

Judd, K. L., Maliar, L. and Maliar, S. 2010, A cluster-grid projection method: solving problems with high dimensionality, Technical report, National Bureau of Economic Research.

Kennedy, J. 1987, 'A computable game theoretic approach to modelling competitive fishing', *Marine Resource Economics*, vol. 4, no. 1, pp. 1–14.

Krusell, P. and Smith, J. A. A. 1998, 'Income and wealth heterogeneity in the macroeconomy', *Journal of Political Economy*, vol. 106, no. 5, pp. 867–896.

Kutschinski, E., Uthmann, T. and Polani, D. 2003, 'Learning competitive pricing strategies by multi-agent reinforcement learning', *Journal of Economic Dynamics and Control*, vol. 27, no. 11, pp. 2207–2218.

Levhari, D. and Mirman, L. 1980, 'The great fish war: an example using a dynamic Cournot-Nash solution', *The Bell Journal of Economics*, vol. pp. 322–334.

Maskin, E. and Tirole, J. 1988, 'A theory of dynamic oligopoly, I: Overview and quantity competition with large fixed costs', *Econometrica: Journal of the Econometric Society*, vol. pp. 549–569.

Murphy, F., Toman, M. and Weiss, H. 1987, 'A stochastic dynamic Nash game analysis of policies for managing the strategic petroleum reserves of consuming nations', *Management science*, vol. pp. 484–499.

Negri, D. 1989, 'The common property aquifer as a differential game', *Water Resources Research*, vol. 25, no. 1, pp. 9–15.

Pakes, A. and McGuire, P. 2001, 'Stochastic algorithms, symmetric Markov perfect equilibrium, and the curseof dimensionality', *Econometrica*, vol. 69, no. 5, pp. 1261–1281.

Riedmiller, M. 2005, Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method, *in* 'Machine Learning: ECML 2005', Springer, pp. 317–328.

Rubio, S. and Casino, B. 2001, 'Competitive versus efficient extraction of a common property resource: The groundwater case', *Journal of Economic Dynamics and Control*, vol. 25, no. 8, pp. 1117–1137.

Rui, X. and Miranda, M. 1996, 'Solving nonlinear dynamic games via orthogonal collocation: An application to international commodity markets', *Annals of Operations Research*, vol. 68, no. 1, pp. 89–108.

Shapley, L. 1953, 'Stochastic games', *Proceedings of the National Academy of Sciences of the United States of America*, vol. 39, no. 10, pp. 1095.

Shoham, Y., Powers, R. and Grenager, T. 2007, 'If multi-agent learning is the answer, what is the question?', *Artificial Intelligence*, vol. 171, no. 7, pp. 365–377.

Stachurski, J. 2008, 'Continuous state dynamic programming via nonexpansive approximation', *Computational Economics*, vol. 31, no. 2, pp. 141–160.

Sutton, R. and Barto, A. 1998, *Reinforcement learning: An introduction*, Vol. 1, Cambridge University Press.

Tesauro, G. and Kephart, J. O. 2002, 'Pricing in agent economies using multi-agent Q-learning', *Autonomous Agents and Multi-Agent Systems*, vol. 5, no. 3, pp. 289–304.

Tesfatsion, L. and Judd, K. L. 2006, *Agent-based computational economics*, Vol. 2.

Timmer, S. and Riedmiller, M. 2007, Fitted q iteration with cmacs, *in* 'Approximate Dynamic Programming and Reinforcement Learning, 2007. ADPRL 2007. IEEE International Symposium on', IEEE, pp. 1–8.

Watkins, C. and Dayan, P. 1992, 'Q-learning', *Machine learning*, vol. 8, no. 3, pp. 279–292.

Weintraub, G., Benkard, C. and Van Roy, B. 2008, 'Markov Perfect Industry Dynamics With Many Firms', *Econometrica*, vol. 76, no. 6, pp. 1375–1411.

Weintraub, G., Benkard, C. and Van Roy, B. 2010, 'Computational methods for oblivious equilibrium', *Operations research*, vol. 58, no. 4-Part-2, pp. 1247–1265.

Weiring, M. and Otterlo, M., eds 2012, *Reinforcement learning: State-of-the-Art*, Vol. 12 of *Adaptaion, Learning and Optimization*, Springer.

Whiteson, S., Taylor, M. E., Stone, P. et al. 2007, *Adaptive tile coding for value function approximation*, Computer Science Department, University of Texas at Austin.

Wolpert, D. H. 1996, 'The lack of a priori distinctions between learning algorithms', *Neural computation*, vol. 8, no. 7, pp. 1341–1390.