

# Refining Basis Functions in Least-Square Approximations of Zero-Sum Markov Games

Xiaolei Li  
xli10@uiuc.edu

CS 497 SML Course Project

## Abstract

This paper extends the work done in [4] for value function approximation of zero-sum Markov games. In the original work, the authors used their least-squares policy iteration algorithm (LSPi) to approximate the Q-values and calculate the optimal policy in the context of reinforcement learning. In the approximation process, a set of basis functions was needed. The authors created these functions using domain knowledge and experimental data. We will demonstrate a method of refining a set of these functions by means of Vector Quantization (VQ). In the experimental section, we will show that this process produces competitive results at a fraction of the cost.

## 1 Introduction

Markov games are a generalization of Markov decision processes and matrix games. Many practical problems can be modeled by the framework. Pursuit-evasion and network flow are just two examples. Initial work on these problems neglected the presence of the opponent and treated the problem as a simple MDP. Recent work [8, 1] have emphasized exploiting the opponents and learning with them in mind. However, in games with large state spaces or action spaces, a prohibitively large number of linear programs are needed to solve the game.

In [3, 4, 5, 6], the authors used least-square approximation and policy iteration to try to remove the burden of linear programming. They first approximate the Q-table in the context of reinforcement learning and then learn the optimal policy through policy iteration from the approximation. In this manner, much of the linear programming needed for constructing the Q-table are avoided.

In their algorithm, a set of basis functions are needed to approximate the Q-table. This is a definite weakness because a poor set of basis functions would probably yield a poor policy. It is the task of the domain expert to design a set that is accurate yet

of reasonable size. We propose that by using Vector Quantization (VQ), we can refine an initial set of basis functions by pruning away some of the poor ones. In the domain of Littman's soccer game, we will show that this still produces a competitive policy.

The rest of the paper is organized as follows. In Section 2, we review the Markov game framework. In Section 3, we examine the traditional reinforcement learning approach to solving these problems. In Section 4, we introduce the least-square policy iteration algorithm. In Section 5, we examine how VQ can be applied to LSPi. And finally, we show experimental results in Section 6 and conclude with Section 7.

## 2 Markov Games

A two-player zero-sum Markov game is defined as a tuple  $(\mathcal{S}, \mathcal{A}, \mathcal{O}, P, R)$ , where

- $\mathcal{S}$  is the set of states.
- $\mathcal{A}$  is the set of actions for the agent.
- $\mathcal{O}$  is the set of actions for the opponent.
- $P$  is the state transition function:  $\mathcal{S} \times \mathcal{A} \times \mathcal{O} \times \mathcal{S} \rightarrow [0, 1]$ .
- $R(s, a, o, s')$  is the payoff for the agent:  $\mathcal{S} \times \mathcal{A} \times \mathcal{O} \times \mathcal{S} \rightarrow \mathbb{R}$ .

The game is zero-sum in the sense that the opponent will receive the negative of the payoff received by the agent. For simplicity, we'll use  $R(s, a, o)$  as the expected reward:  $R(s, a, o) = \sum_{s'} P(s, a, o, s') R(s, a, o, s')$ . It is noteworthy that when  $|\mathcal{O}|$  is 1, the game becomes a simple MDP. And when  $|\mathcal{S}|$  is 1, the game degenerates to a simple matrix game.

In the game, a policy  $\pi$  for our agent is stochastic. It's not hard to see that a deterministic strategy could be easily defeated (*e.g.*, paper-rock-scissors). Formally,  $\pi$  maps a particular state to a probability distribution over  $\mathcal{A}$ :  $\mathcal{S} \rightarrow \Omega(\mathcal{A})$ .  $\pi(s)$  denotes the probability distribution over actions in state  $s$ .  $\pi(s, a)$

denotes the probability of executing action  $a$  in state  $s$ .

The goal of the game is for us, the agent, to maximize the final discounted value of the game. As in most two-player games, a minimax approach is taken. In other words, the policy will assume the worst case of the opponent and try to maximize based on that. The value function and Q function are given below.  $\gamma$  is the discount factor.

$$V(s) = \max_{\pi(s) \in \Omega(\mathcal{A})} \min_{o \in \mathcal{O}} \sum_{a \in \mathcal{A}} Q(s, a, o) \pi(s, a) \quad (1)$$

$$Q(s, a, o) = R(s, a, o) + \gamma \sum_{s'} P(s, a, o, s') V(s') \quad (2)$$

### 3 Reinforcement Learning

When  $P$  and  $R$  are unknown, reinforcement learning is a popular approach to solving Markov games. In the work by [8], a solution is given by augmenting the traditional Q-Learning algorithm with the opponent's actions. They called this Minimax-Q. The update formulas remain very similar. However, due to the minimax formulation, linear programming is needed at every step to update the values in Equation 1. The linear program is as follows.

$$\begin{aligned} \text{Maximize:} & \quad V(s) \\ \text{Subject to:} & \quad \sum_{a \in \mathcal{A}} \pi(s, a) = 1 \\ & \quad \forall a \in \mathcal{A}, \pi(s, a) \geq 0 \\ & \quad \forall o \in \mathcal{O}, V(s) \leq \sum_{a \in \mathcal{A}} Q(s, a, o) \pi(s, a) \end{aligned}$$

In addition, just like Q-Learning, Minimax-Q requires an infinite amount of data in order to converge to the optimal policy. These hefty requirements make Minimax-Q impractical in many real world problems.

### 4 Least Square Policy Iteration (LSPI)

The authors in [4, 3, 6] propose a new method of solving the problem. They approximate the Q-values and find a policy from the approximation. This greatly reduces the number of linear programs needed and also the number of samples required.

LSPI begins by first creating  $k$  basis functions. These functions are to be designed by a human expert.  $k$  is usually much smaller than the size of the original Q-table ( $\mathcal{S} \times \mathcal{A} \times \mathcal{O}$ ). One entry in the Q-table can now be expressed as a linear combination of these functions.

$$\hat{Q}(s, a, o) = \sum_{i=1}^k \phi_i(s, a, o) w_i = \phi(s, a, o)^\top w$$

$\phi_i(s, a, o)$  are the individual basis functions,  $\phi(s, a, o)$  is a vector of them, and  $w$  are the weights. The optimal

policy is calculated in the minimax sense over  $\phi$  and  $w$ .

$$\pi(s) = \operatorname{argmax}_{\pi(s) \in \Omega(\mathcal{A})} \min_{o \in \mathcal{O}} \sum_{a \in \mathcal{A}} \pi(s, a) \phi(s, a, o)^\top w$$

The equivalent linear program is as follows.

$$\begin{aligned} \text{Maximize:} & \quad V(s) \\ \text{Subject to:} & \quad \sum_{a \in \mathcal{A}} \pi(s, a) = 1 \\ & \quad \forall a \in \mathcal{A}, \pi(s, a) \geq 0 \\ & \quad \forall o \in \mathcal{O}, V(s) \leq \sum_{a \in \mathcal{A}} \pi(s, a) \phi(s, a, o)^\top w \end{aligned}$$

There are  $(1 + |\mathcal{A}|)$  variables in the linear program and  $(|\mathcal{A}| + |\mathcal{O}| + 1)$  constraints.

Now suppose we didn't know  $w$ . To solve for it in the least square sense, we first define  $\Phi$  ( $|\mathcal{S}| |\mathcal{A}| |\mathcal{O}| \times k$  matrix) as

$$\Phi = \begin{pmatrix} \phi(s_1, a_1, o_1)^\top \\ \dots \\ \phi(s, a, o)^\top \\ \dots \\ \phi(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}, o_{|\mathcal{O}|})^\top \end{pmatrix}$$

We can now re-write Equation 2 with  $\Phi$  and  $w$ .

$$\begin{aligned} \Phi w & \approx \mathcal{R} + \gamma \mathbf{P} \Phi w \\ (\Phi - \gamma \mathbf{P} \Phi) w & \approx \mathcal{R} \\ & \quad \vdots \\ w & = (\Phi^\top (\Phi - \gamma \mathbf{P} \Phi))^{-1} \Phi^\top \mathcal{R} \end{aligned}$$

Because  $k \ll |\mathcal{S}| |\mathcal{A}| |\mathcal{O}|$ , the linear system is an *over-constrained* system over  $k$  parameters. Thus a least-square approximation of the Q-table can be calculated. Solving this system in the least-square sense for  $w$  gives us a good approximation of  $Q$ . The authors term this Least-Square Approximation of Q Function or LSQ[3]. This approach is similar to least-square temporal difference (LSTD) learning in [2]. In LSTD, the approximations are over the values  $V$  in Equation 1 whereas LSQ approximates over  $Q$  in Equation 2.

Supposing that the probability transition functions and the reward functions are known,  $w$  can be solved by the equation  $w = \mathbf{A}^{-1} b$  where  $\mathbf{A} = \Phi^\top (\Phi - \gamma \mathbf{P} \Phi)$  and  $b = \Phi^\top \mathcal{R}$ .  $\Phi$  is defined as before.  $\mathbf{P} \Phi$  and  $\mathcal{R}$  are the following.

$$\mathbf{P} \Phi = \begin{pmatrix} \sum_{s'} P(s_1, a_1, o_1, s') \phi(s', \pi(s'), o_1)^\top \\ \dots \\ \sum_{s'} P(s, a, o, s') \phi(s', \pi(s'), o)^\top \\ \dots \\ \sum_{s'} P(s_{|\mathcal{S}|}, a_{|\mathcal{A}|}, o_{|\mathcal{O}|}, s') \phi(s', \pi(s'), o_{|\mathcal{O}|})^\top \end{pmatrix}$$

$$\mathcal{R} = \begin{pmatrix} \sum_{s'} P(s_1, a_1, o_1, s') R(s_1, a_1, o_1, s') \\ \dots \\ \sum_{s'} P(s, a, o, s') R(s, a, o, s') \\ \dots \\ \sum_{s'} P(s|_{\mathcal{S}}, a|_{\mathcal{A}}, o|_{\mathcal{O}}, s') R(s|_{\mathcal{S}}, a|_{\mathcal{A}}, o|_{\mathcal{O}}, s') \end{pmatrix}$$

When  $P$  and  $R$  are unknown, the goal is to approximate them using the sample data we have. Given a set of examples,  $D = \{s_{d_i}, a_{d_i}, o_{d_i}, s'_{d_i}, r_{d_i} | i = 1, 2, \dots, L\}$ , the authors show that  $\widehat{\Phi}$ ,  $\widehat{\mathbf{P}}\widehat{\Phi}$ , and  $\widehat{\mathcal{R}}$  can be approximated by the following.

$$\widehat{\Phi} = \begin{pmatrix} \phi(s_{d_1}, a_{d_1}, o_{d_1})^\top \\ \dots \\ \phi(s_{d_i}, a_{d_i}, o_{d_i})^\top \\ \dots \\ \phi(s_{d_L}, a_{d_L}, o_{d_L})^\top \end{pmatrix}$$

$$\widehat{\mathbf{P}}\widehat{\Phi} = \begin{pmatrix} \phi(s'_{d_1}, \pi(s'_{d_1}), o'_{d_1})^\top \\ \dots \\ \phi(s'_{d_i}, \pi(s'_{d_i}), o'_{d_i})^\top \\ \dots \\ \phi(s'_{d_L}, \pi(s'_{d_L}), o'_{d_L})^\top \end{pmatrix}$$

$$\widehat{\mathcal{R}} = \begin{pmatrix} r_{d_1} \\ \dots \\ r_{d_i} \\ \dots \\ r_{d_L} \end{pmatrix}$$

Through these equations,  $\mathbf{A}$  and  $b$  can be approximated quite easily given a set of examples. The calculations even satisfy the additive property. Thus more data can be added painlessly. The update equations are

$$\widehat{\mathbf{A}} \leftarrow \widehat{\mathbf{A}} + \phi(s, a, o) \left( \phi(s, a, o) - \gamma \sum_{a' \in \mathcal{A}} \pi(s', a') \phi(s', a', o') \right)^\top$$

$$\widehat{b} \leftarrow \widehat{b} + \phi(s, a, o) r$$

for an example  $(s, a, o, s', r)$ .  $o'$  is the minimizing opponent action in the previously mentioned linear program. In other words, it is the  $o'$  of the constraint with zero slack.

A natural question to ask is how close are these approximations. The author state that

$$E(\widehat{\mathbf{A}}) = \frac{L}{|\mathcal{S}||\mathcal{A}||\mathcal{O}|} \mathbf{A} \quad E(\widehat{b}) = \frac{L}{|\mathcal{S}||\mathcal{A}||\mathcal{O}|} b$$

$$E(\widehat{w}) = E(\widehat{\mathbf{A}}^{-1} \widehat{b}) = \frac{L^2}{|\mathcal{S}|^2 |\mathcal{A}|^2 |\mathcal{O}|^2} \mathbf{A}^{-1} b$$

Hence, when  $L$  is sufficiently large,  $\widehat{w}$  will be close to  $w$ . The cost of constructing  $\widehat{\mathbf{A}}$  is  $O(Lk^2)$  and the cost of solving for  $w$  given  $\widehat{\mathbf{A}}^{-1}$  and  $b$  is  $O(k^3)$ .

## 5 Refining Basis Functions

The foundation of LSPI lies in the basis functions. They need to be precise and preferably compact. It's not hard for a knowledgeable expert to create an intuitive set of functions. But when the size of the set is large, constructing  $\widehat{\mathbf{A}}$  and solving for  $w$  can become quite expensive. So it would be nice if there was a way to prune away basis functions that are useless or redundant. A well-known technique from multivariate statistics that will do this is Vector Quantization (VQ). In VQ one can represent a vector of values by using a fixed subset of representative values. The goal is to get a good approximation by sacrificing precision.

Given  $\widehat{\Phi}$ , we can perform VQ on it and get  $k$  weights back. Each weight will correspond to a particular basis function and the bigger the weight is, the better the corresponding function is. With these weights, we can then choose a new set of basis functions and continue with LSPI.

## 6 Experimental Results

We tested our method of refining the basis functions in the domain of a simple Littman's soccer game [4, 8]. The game is defined on a  $4 \times 4$  grid with two players. Each player occupies one cell at a time. Initially, the players are placed randomly in the first and last column, and the ball is given to a player randomly. Each player has 5 possible actions: up, down, left, right, and stand. The order of the players' executing their action at each time step is randomly determined. If there is a collision with the border or another player, the player stays still. If the player with the ball attempts to move into a cell with the other player in it, he will lose possession of the ball. The goals of the game are 2-cell wide regions extending off the middle 2 rows at the first and fourth column of the grid. Players score by moving into the goals while possessing the ball, but they cannot score into their own goals. Scoring for the agent results in a +1 reward while letting the opponent score results in a -1 reward. The discount factor of the game is set to 0.9.

First, a policy was generated using the human-created basis functions given in [4]. The basis functions were based on basic blocks of 36 functions. This block is replicated for each of the 25 action pairs with only one block active during any time step. This makes up for a total of 900 basis functions or parameters in  $w$ . The 36 functions in the basic block are further divided into 4 partitions based on all possible dichotomies of the following binary propositions:

- $P_1$  “The attacker is closer to the defender’s goal than the defender”.
- $P_2$  “The defender is within 2 Manhattan distances to the attacker”.

With the 4 possible dichotomies of  $P_1$  and  $P_2$ , we get the following 36 basis functions. The attacker is the player with the ball.

- $P_1$  and not  $P_2$ 
  - 1.0 (constant)
  - $D_H P_A G_D$  : horizontal distance<sup>1</sup> from attacker to defender’s goal.
  - $S D_V P_A G$  : signed distance from goal zone to attacker.
  - $D_H P_A G_D \times S D_V P_A G$
- $P_1$  and  $P_2$ 
  - 1.0,  $D_H P_A G_D$ ,  $S D_V P_A G$ ,  $D_H P_A G_D \times S D_V P_A G$
  - $P_A U G$  : proposition that’s true if attacker is in upper row of goal zone.
  - $P_A L G$  : proposition that’s true if attacker is in lower row of goal zone.
  - $S D_H P_A P_D$  : signed distance distance between the players.
  - $S D_V P_A P_D$  : signed vertical distance between the players.
- not  $P_1$  and not  $P_2$ 
  - same as above, excluding  $S D_H P_A P_D$ .
- not  $P_1$  and  $P_2$ 
  - $D_H P_A G_D$ ,  $S D_V P_A G$ ,  $D_H P_A G_D \times S D_V P_A G$ ,  $P_A U G$ ,  $P_A L G$
  - $P_D G_D L$  : proposition that’s true if defender is by his goal line.
  - $P_D W G$  : proposition that’s true if defender is within goal zone.
  - 10 indicators for the 10 possible positions<sup>2</sup> of the defender within Manhattan distance of 2.

The second player in our experiments was a policy calculated similarly to the first except the basis functions are derived by VQ. Using VQ on  $\hat{\Phi}$  of 30000 examples, the best 15 basis functions out of the basic set of 36 was chosen. With 25 blocks, this makes a total of 375 basis functions. The basic block is as follows.

- $P_1$  and not  $P_2$

<sup>1</sup> All distances are Manhattan distances scaled to [0, 1]. They are unsigned unless noted.

<sup>2</sup> There are actually 12 such positions. But the 2 cells behind the attacker are excluded because they imply  $P_1$ .

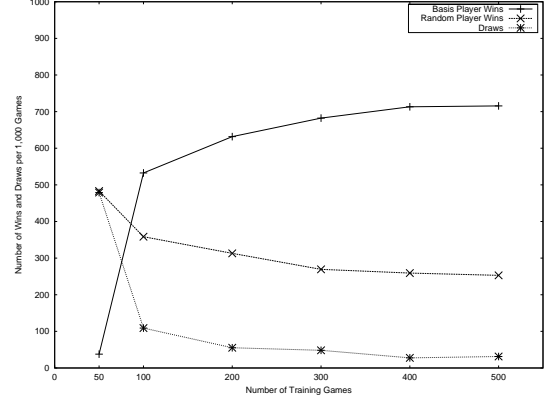


Figure 1: Original basis function player vs. random player

- 1.0 (constant)
- $P_1$  and  $P_2$ 
  - 1.0
  - $D_H P_A G_D$  : horizontal distance from attacker to defender’s goal.
  - $P_A U G$  : proposition that’s true if attacker is in upper row of goal zone.
- not  $P_1$  and not  $P_2$ 
  - 1.0,  $D_H P_A G_D$ ,  $P_A U G$
  - $P_A L G$  : proposition that’s true if attacker is in lower row of goal zone.
  - $S D_V P_A P_D$  : signed vertical distance between the players.
- not  $P_1$  and  $P_2$ 
  - $D_H P_A G_D$ ,  $P_A U G$ ,  $P_A L G$
  - $P_D G_D L$  : proposition that’s true if defender is by his goal line.
  - $P_D W G$  : proposition that’s true if defender is within goal zone.
  - $P_D F A$  : proposition that’s true if the defender is in the cell in front of the attacker.

This is a much more compact set of basis functions than the original. As a result, constructing  $\hat{\Phi}$  and solving for  $w$  took much less time than the original set. But surprisingly, the results given by the new set was competitive (and sometimes even better) with the original.

First, in Figure 1, we show the policy derived from the original set of basis functions vs. a random player (uniform probability over the actions). The results are averages of 5 separate tournaments of 1000 games each. A game was a draw if 100 time steps passed without a winner. For each data set, LSP1 was run until

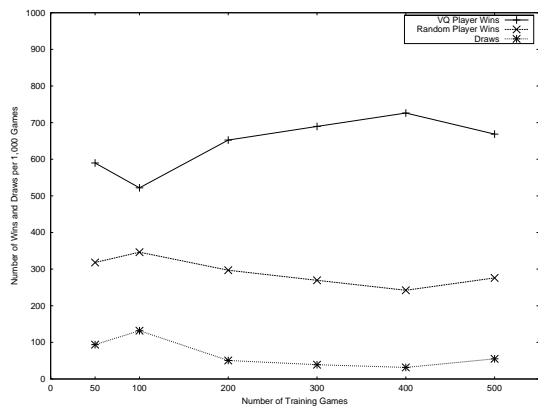


Figure 2: VQ basis function player vs. random player

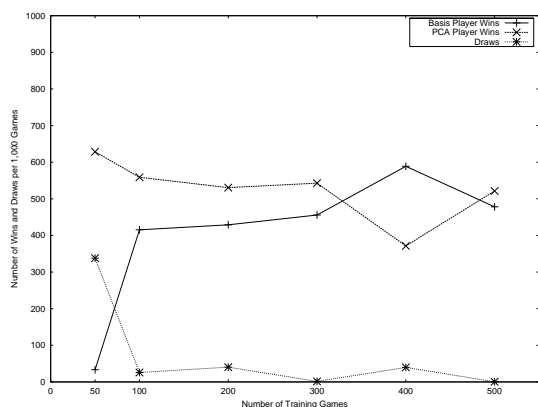


Figure 3: Original basis function player vs. VQ basis function player

convergence<sup>3</sup> or 25 maximum iterations. It’s not surprising that the policy using the basis functions quickly gained an advantage over the random player even after just 100 training games.

In Figure 2, we show the policy derived from the VQ set of basis functions vs. the random player. The results are similar to Figure 1 but it’s surprising that with only 50 training games, the VQ player is able to win a majority of the games, whereas the original player lost the majority of them. This could be attributed to the refined set of basis functions that provided a better approximation of the Q-table for LSPI.

And lastly, in Figure 3, we show the policy derived from the original 900 basis functions vs. the policy derived from the reduced 375 basis functions. Just like in Figure 2, the VQ player won a majority of the games with only 50 training games. As the number of training games increased, the 2 players became comparable. However, when using the VQ set of basis functions, the calculations in LSPI took only a fraction of the time required by using the original basis functions.

<sup>3</sup>A policy is considered converged when its  $L_2$  distance to the previous iteration is less than 0.005.

## 7 Conclusion and Future Work

In this paper, we have used VQ to refine an initial set of basis functions. In the domain of a simple Littman soccer game, we demonstrate that this produces good results. The refined player performed competitively with the original player while requiring much less computation. Using this filtering technique removes the burden of constructing a perfect set of basis functions from the expert designer. Instead, we can just start with a big initial set and let VQ prune away the useless functions.

In our experiments, we showed that the pruned set performed just as well if not better than the original. This was especially true when little training was provided. Intuitively, we can think of the pruned set as carrying better information. But this effect wore out as the number of training examples increased.

In future work, we would like to try other techniques of pruning basis functions and perhaps even try generating the basis functions from scratch using the given sample data. And it would be interesting to see performances in other domains.

## References

- [1] Bowling, M. & Veloso, M. Rational and Convergent Learning in Stochastic Games.
- [2] Bradtke, S. & Barto, A. Linear Least-Square Algorithms for Temporal Difference Learning.
- [3] Lagoudakis, Michail G. & Parr, Ronald. Model-Free Least-Square Policy Iteration.
- [4] Lagoudakis, Michail G. & Parr, Ronald. Value Function Approximation in Zero-Sum Markov Games.
- [5] Lagoudakis, Michail G. & Parr, Ronald. Model-Free Least-Squares Policy Iteration.
- [6] Lagoudakis, Michail G. & Parr, Ronald. Learning in Zero-Sum Team Markov Games using Factored Value Functions.
- [7] Littman, Michael L. Algorithms for Sequential Decision Making. Ph. D. Dissertation.
- [8] Littman, Michael L. Markov Games as a Framework for Multi-Agent Reinforcement Learning.

## 8 Misc. Comments

All the code for the project was done in MATLAB. VQ in the experiments was accomplished by Principal Component Analysis (PCA) provided by MATLAB. The first vector returned by PCA is the result of VQ. For this specific problem, because the original set of basis function was in blocks, I averaged the weights by the block. PCA return 900 weights but there are really only 36 distinct basis functions. So I computed the average weight of each of these 36 functions across the 25 blocks. Sorting the final 36 weights by their

absolute values, I then took the top 15. It's interesting that out of the 15 chosen basis functions, 10 was related to the goal. Many of the functions relating the 2 players were neglected. While this is probably just a coincidence, the goal-related functions are probably the most important in winning games.

In addition,  $\hat{\mathbf{A}}$  is not guaranteed to be non-singular. In my implementation, if it is singular, I used MATLAB's pseudo-inverse function. In training, all the examples are generated from two random players playing each other. The examples were given to LSP1 all at once.

For the experiments, I would like to have done more than just 500 training games. The original paper did up to 1000. But as I found out, LSP1 with just 400 or 500 games takes a *long* time. With 500 games, there are about 30000 samples. This makes the construction of  $\hat{\mathbf{A}}$  a very expensive task and inverting it a tough one, too. On the CSIL Sun machines, performing 9 iterations of LSP1 with 500 training games and the original 900 basis functions took over 15 hours. Using the VQ refined 375 functions, 7 iterations with 500 training games took only 1 hour.