Reinforcement learning with a bilinear Q function

Charles Elkan

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404
elkan@ucsd.edu

Abstract. Many reinforcement learning methods are based on a function $Q(s, a)$ whose value is the discounted total reward expected after performing the action a in the state s. This paper explores the implications of representing the Q function as $Q(s, a) = s^T W a$, where W is a matrix that is learned. In this representation, both s and a are real-valued vectors that may have high dimension. We show that action selection can be done using standard linear programming, and that W can be learned using standard linear regression in the algorithm known as fitted Q iteration. Experimentally, the resulting method learns to solve the mountain car task in a sample-efficient way. The same method is also applicable to an inventory management task where the state space and the action space are continuous and high-dimensional.

1 Introduction

In reinforcement learning (RL), an agent must learn what actions are optimal for varying states of the environment. Let s be a state, and let a be an action. Many approaches to RL are centered around the idea of a Q function. This is a function $Q(s, a)$ whose value is the total reward achieved by starting in state s, performing action a, and then performing the optimal action, whatever that might be, in each subsequent state.

In general, a state can have multiple aspects and an action can have multiple sub-actions, so s and a are vectors. In general also, the entries in state and action vectors can be continuous or discrete. For domains where s and a are both real-valued vectors, this paper investigates the advantages of representing $Q(s, a)$ as a linear function of both s and a. Specifically, this representation, which is called bilinear, is $Q(s, a) = s^T W a$, where W is a matrix. The goal of reinforcement learning is then to induce appropriate values for the entries of W. We show that learning W can be reduced to standard linear regression, by applying a previously published batch RL method known as fitted Q iteration [Murphy, 2005] [Ernst et al., 2005]. Experiments show that the resulting algorithm learns to solve the well-known mountain car task with very few training episodes, and that the algorithm can be applied to a challenging problem where the state space is $\mathbb{R}^{33}$ and the action space is $\mathbb{R}^{82}$.

The approach suggested in this paper is surprisingly simple. We hope that readers see this as an advantage, not as a lack of sophistication; although simple, the approach is novel. The paper is organized as follows. First, Section 2 explains the consequences of writing $Q(s, a)$ as $s^T W a$. Next, Section 3 discusses using fitted Q iteration to learn W. Then, Section 4 explains how learning W within fitted Q iteration can be reduced to linear regression. Finally, Sections 5 and 6 describe initial experimental results, and Section 7 is a brief conclusion. Related work is discussed throughout the paper, rather than in a separate section.

2 The bilinear representation of the Q function

As mentioned, we propose to represent Q functions as bilinear, that is, to write $Q(s, a) = s^T W a$ where s and a are real-valued column vectors. Given a Q function, the recommended action a^* for a state s is $a^* = \text{argmax}_a \; Q(s, a)$. With the bilinear representation, the recommended action is

$$a^* = \text{argmax}_a \; Q(s, a) = \text{argmax}_a \; x \cdot a$$

where $x = s^T W$. This maximization is a linear programming (LP) task. Hence, it is tractable in practice, even for high-dimensional action and state vectors.

When actions are real-valued, in general they must be subject to constraints in order for optimal actions to be well-defined and realistic. The maximization $\text{argmax}_a \; x \cdot a$ remains a tractable linear programming task as long as the constraints on a are linear. As a special case, linear constraints may be lower and upper bounds for components (sub-actions) of a. Constraints may also involve multiple components of a. For example, there may be a budget limit on the total cost of some or all sub-actions, where each sub-action has a different cost proportional to its magnitude.[1]

When following the policy defined directly by a Q function, at each time step the action vector a is determined by an optimization problem that is a function only of the present state s. The key that can allow the approach to take into account future states, indirectly, is that the Q function can be learned. A learned Q function can emphasize aspects of the present state and action vector that are predictive of long-term reward. With the bilinear approach, the Q function is learnable in a straightforward way. The bilinear approach is limited, however, because the optimal Q function that represents long-term reward perfectly may be not bilinear. Of course, other parametric representations for Q functions are subject to the same criticism.

The solution to an LP problem is typically a vertex in the feasible region. Therefore, the maximization operation selects a so-called "bang-bang" action

[1] Given constraints that combine two or more components of the action vector, the optimal value for each action component is in general not simply its lower or upper bound. Even when optimal action values are always lower or upper bounds, the LP approach is not trivial in high-dimensional action spaces: it finds the optimal value for every action component in polynomial time, whereas naive search might need $O(2^d)$ time where d is the dimension of the action space.

vector, each component of which is an extreme value. In many domains, bang-bang actions are optimal given simplified reward functions, but actions that change smoothly over time are desirable. In at least some domains, the criteria to be minimized or maximized to achieve desirably smooth solutions are known, for example the "minimum jerk" principle [Viviani and Flash, 1995]. These criteria can be included as additional penalties in the maximization problem to be solved to find the optimal action. If the criteria are nonlinear, the maximization problem will be harder to solve, but it may still be convex, and it can still be possible to learn a bilinear Q function representing how long-term reward depends on the current state and action vectors.

Related work. Other approaches to reinforcement learning with continuous action spaces include [Lazaric et al., 2007], [Melo and Lopes, 2008], and [Pazis and Lagoudakis, 2009]. Linear programming has been used before in connection with reinforcement learning, in [De Farias and Van Roy, 2003] and in [Pazis and Parr, 2011] recently, among other papers. However, previous work does not use a bilinear Q function.

The bilinear representation of Q functions contains an interaction term for every component of the action vector and of the state vector. Hence, it can represent the consequences of each action component as depending on each aspect of the state. Some previously proposed representations for Q functions are less satisfactory. The simplest representation is tabular: a separate value is stored for each combination of a state value s and an action value a. This representation is usable only when both the state space and the action space are discrete and of low cardinality. A linear representation is a weighted combination of fixed basis functions [Lagoudakis and Parr, 2003]. In this representation $Q(s,a) = w \cdot [\phi_1(s,a), \cdots, \phi_p(s,a)]$ where ϕ_1 to ϕ_p are fixed real-valued functions and w is a weight vector of length p. The bilinear approach is a special case of this representation where each basis function is the product of one state component and one action component; see Equation (1) below. An intuitive drawback of previously used basis function representations is that they ignore the distinction between states and actions: essentially, s and a are concatenated as inputs to the basis functions ϕ_i.

When the action space is continuous, how to perform the maximization operation $\text{argmax}_a Q(s,a)$ efficiently is a crucial issue; see the discussion in [Pazis and Parr, 2011] where the problem is called action selection. With the bilinear representation, finding the optimal action exactly in a multidimensional continuous space of candidate actions is tractable, and fast in practice. No similarly general approach is known when other basis function representations are used, or when neural networks [Riedmiller, 2005] or ensembles of decision trees [Ernst et al., 2005] are used. Note however that in the context of computational neuroscience, a deep approach to avoiding the maximization has been proposed [Todorov, 2009].

3 Fitted Q iteration

In what is called the batch setting for RL, an optimal policy or Q function must be learned from historical data [Neumann, 2008]. A single training example is a quadruple $\langle s, a, r, s' \rangle$ where s is a state, a is the action actually taken in this state, r is the immediate reward that was obtained, and s' is the state that was observed to come next. The method named fitted Q iteration is the following algorithm:

Define $Q_0(s, a) = 0$ for all s and a.
For horizon $h = 0, 1, 2, \ldots$
    For each example $\langle s, a, r, s' \rangle$
        let the label $v = r + \gamma \max_b Q_h(s', b)$
    Train Q_{h+1} with labeled tuples $\langle s, a, v \rangle$

The next section shows how to train Q_{h+1} efficiently using the bilinear representation. The variable h is called the horizon, because it counts how many steps of lookahead are implicit in the learned Q function. In some applications, in particular medical clinical trials, the maximum horizon is a small fixed integer [Murphy, 2005]. In these cases the discount factor γ can be set to one.

The Q iteration algorithm is deceptively simple. It was proposed essentially in the form above in parallel by [Murphy, 2005] and [Ernst et al., 2005], and it can be traced back to [Gordon, 1995b,Gordon, 1995a]. However, learning a Q function directly from multiple examples of the form $\langle s, a, v \rangle$ is also part of the Q-RRL algorithm of [Džeroski et al., 2001], and there is a history of related research in papers by economists [Judd and Solnick, 1994] [Stachurski, 2008].

Some of the advantages of Q iteration can be seen by considering the update rule of standard Q learning, which is

$$Q(s, a) := (1 - \alpha)Q(s, a) + \alpha[r + \gamma \max_b Q(s', b)].$$

This rule has two drawbacks. First, no general method is known for choosing a good learning rate α. Second, if $Q(s, a)$ is approximated by a continuous function, then when its value is updated for one $\langle s, a \rangle$ pair, its values for different state-action pairs are changed in unpredictable ways. In contrast, in the Q iteration algorithm all $Q(s, a)$ values are fitted simultaneously, so the underlying supervised learning algorithm (linear regression in our case) can make sure that all of them are fitted reasonably well, and no learning rate is needed.

Another major advantage of the fitted Q iteration algorithm is that historical training data can be collected in alternative ways. There is no need to collect trajectories starting at specific states. Indeed, there is no need to collect consecutive trajectories of the form $\langle s_1, a_1, r_1, s_2, a_2, r_2, s_3, \ldots \rangle$. And, there is no need to know the policy π that was followed to generate historical episodes of the form $\langle s, a = \pi(s), r, s' \rangle$. Q iteration is an off-policy method, meaning that data collected while following one or more non-optimal policies can be used to learn a better policy.

Off-policy methods for RL that do not do active exploration face the problem of sample selection bias. The probability distribution of training examples $\langle s, a\rangle$ is different from the probability distribution of optimal examples $\langle s, \pi^*(s)\rangle$. A reason why Q iteration is correct as an off-policy method is that it is discriminative as opposed to generative. Q iteration does not model the distribution of state-action pairs $\langle s, a\rangle$. Instead, it uses a discriminative method to learn only how the values to be predicted depend on s and a.

A different type of bias faced by fitted Q iteration in general is excessive optimism implicit in the maximization operation [Chakraborty et al., 2008]. In the experiments below, this bias does not cause major problems. In future work we will explore how the double Q learning idea [van Hasselt, 2010] can be combined with fitted Q iteration to reduce the optimism bias.

4 Learning the matrix W

A function of the form $s^T W a$ is called bilinear because it is linear in its first argument s and also linear in its second argument a. Consider a training set of examples of the form $\langle s, a, v\rangle$ where v is a target value. It is not immediately obvious how to learn a matrix W such that $s^T W a = v$ approximately. However, we show that the training task can be reduced to standard linear regression. Let the vectors s and a be of length m and n respectively, so that $W \in \mathbb{R}^{m \times n}$. The key is to notice that

$$s^T W a = \sum_{i=1}^{m} \sum_{j=1}^{n} (W \circ sa^T)_{ij} = vec(W) \cdot vec(sa^T). \tag{1}$$

In this equation, sa^T is the matrix that is the outer product of the vectors s and a, and $\circ$ denotes the elementwise product of two matrices, which is sometimes called the Hadamard product. The notation $vec(A)$ means the matrix A converted into a vector by concatenating its columns.

Equation (1) leads to faster training than alternative approaches. In particular, for any square or rectangular matrices A and B with compatible dimensions, $trace(AB) = trace(BA)$. This identity implies the cyclic property of traces, namely $trace(ABC) = trace(CBA)$. A special case of the cyclic property is $s^T W a = trace(s^T W a) = trace(W a s^T)$ which resembles Equation (1). However, computing the product $W(as^T)$ has cubic time complexity while applying Equation (1) has time complexity only $O(mn)$.

Equation (1) is simple, but as far as we know its implications have not been explored before now. Based on it, each training triple $\langle s, a, v\rangle$ can be converted into the pair $\langle vec(sa^T), v\rangle$ and $vec(W)$ can be learned by standard linear regression. The vectors $vec(sa^T)$ are potentially large, since they have length mn, so the exact linear regression computation may be expensive. Stochastic gradient descent is faster than an exact solution, especially if there are many training examples and the matrices sa^T are sparse.

Each entry of the matrix sa^T represents the interaction of a feature of the state vector and a feature of the action vector. In some domains, background

knowledge may tell us that many interaction terms have no predictive value. These terms can simply be omitted from the vectors $vec(sa^T)$, making them shorter and making the linear regression training faster. The corresponding entries of the vectorized W matrix are set to zero.

In a reinforcement learning domain, some states may have high value regardless of what action is selected. The basic bilinear model cannot represent this directly; specifically, it does not allow some components of s to lead to high values $Q(s, a)$ regardless of a. We add a constant additional component to each a vector to obtain this extra expressiveness. The additional constant component is a pseudo-action that always has magnitude 1. For similar reasons, we also add a constant component to each s vector.

If the s and a vectors are short, then the matrix W may not be sufficiently expressive. In this case, it is possible to expand the s vector and/or the a vector before forming the outer product sa^T. For example, s itself may be re-represented as ss^T. This re-representation allows the bilinear model to be a quadratic function of the state. Achieving nonlinearity by expanding training examples explicitly, rather than by using a nonlinear kernel, is in line with current trends in machine learning [Chang et al., 2010]. It may sometimes be appropriate to expand just one of s and a, but not both.

To prevent overfitting, regularized linear regression can be used to learn $vec(W)$. Another approach to prevent overfitting is to require W to have a simplified structure. This can be achieved by defining $W = AB^T$ where A and B are rectangular matrices, and learning A and B instead of W. Training W directly by linear regression is a convex optimization problem, with or without many forms of regularization, and with or without some entries of W constrained to be zero. However, finding the optimal representation AB^T is in general not a convex problem, and has multiple local minima.

5 Mountain car experiments

The mountain car task is perhaps the best-known test case in research on reinforcement learning [Sutton and Barto, 1998]. The state space has two dimensions, the position x and velocity v of the car. The action space has one dimension, acceleration a. In most previous work acceleration is assumed to be a discrete action. Here, it is allowed to be any real value between -0.001 and $+0.001$, but as explained above, linear programming leads to it always being either -0.001 or $+0.001$. In order to allow the bilinear model W to be sufficiently expressive, we use a six-dimensional expanded state space $\langle x, v, x^2, xv, v^2, x^3 \rangle$, and a two-dimensional action vector $\langle 1, a \rangle$ whose first component is a constant pseudo-action. The matrix W then has 12 trainable parameters.

Table 1 shows that fitted Q iteration with a bilinear Q function learns to control the car well with just 400 training tuples. This sample efficiency (that is, speed of learning) is orders of magnitude better than what is achievable with variants of Q learning [Smart and Kaelbling, 2000]. The most sample-efficient previously published method appears to be fitted Q iteration with a neural net-

Table 1. Average length (number of steps) of a testing episode as a function of training set size, in the mountain car domain.

tuples	length
100	285.6
200	317.8
400	88.1
800	88.6
1600	87.4
3200	87.2
6400	87.3
12800	85.9

Notes: The size of a training set is the number of $\langle s, a, r, s' \rangle$ tuples used for bilinear Q iteration. For each tuple independently, s and a are chosen from a uniform distribution over the legal state and action spaces. The discount factor is 0.9; performance is not sensitive to its exact value. Following previous work, the starting state for each testing episode is the bottom of the valley with zero velocity. A testing episode terminates when the goal state is reached, or after 500 steps.

work [Riedmiller, 2005]; the bilinear method requires several times fewer training examples. We conjecture that the bilinear method is sample-efficient because it needs to learn values for fewer parameters. The bilinear method also requires less computation, because both learning and the argmax operation are linear.

Some details of the experiment are important to mention. The precise scenario described in [Sutton and Barto, 1998] is used. Many subsequent papers, including [Riedmiller, 2005], have made changes to the scenario, which makes experimental results not directly comparable. Table 1 is based on testing from a fixed state; we can also consider test episodes starting in arbitrary states. About 1/4 of training sets of size 400 yield Q functions that are successful (reach the goal) from every possible initial state. The other 3/4 of training sets lead to Q functions that are successful for the majority of initial states.

6 Inventory management experiments

In many potential applications of reinforcement learning, the state space, and possibly also the action space, is high-dimensional. For example, in many business domains the state is essentially the current characteristics of a customer, who is represented by a high-dimensional real-valued vector [Simester et al., 2006]. Most existing RL methods cannot handle applications of this nature, as discussed by [Dietterich, 2009]. Research in this area is multidisciplinary, with successful current methods arising from both the operations research community [Powell, 2007] and the machine learning community [Hannah and Dunson, 2011].

In inventory management applications, at each time step the agent sees a demand vector and/or a supply vector for a number of products. The agent must decide how to satisfy each demand in order to maximize long-term benefit, subject to various rules about the substitutability and perishability of products. Managing the stocks of a blood bank is an important application of this type. In this section we describe initial results based on the formulation described in [Yu, 2007] and Chapter 12 of [Powell, 2007]. The results are preliminary because some implementation issues are not yet resolved.

The blood bank stores blood of eight types: AB+, AB-, A+, A-, B+, B-, O+, and O-. Each period, the manager sees a certain level of demand for each

Table 2. Short-term reward functions for blood bank management.

	[Yu, 2007]	new
supply exact blood type	50	0
substitute O- blood	60	0
substitute other type	45	0
fail to meet demand	0	-60
discard blood	-20	0

type, and a certain level of supply. Some types of blood can be substituted for some others; 27 of the 64 conceivable substitutions are allowed. Blood that is not used gets older, and beyond a certain age must be discarded. With three discrete ages, the inventory of the blood bank is a vector in $\mathbb{R}^{24}$. The state of the system is this vector concatenated with the 8-dimensional demand vector and a unit constant component. The action vector has $3 \cdot 27 + 1 = 82$ dimensions since there are 3 ages, 27 possible allocations, and a unit constant pseudo-action. Each component of the action vector is a quantity of blood of one type and age, used to meet demand for blood of the same or another type. The LP solved to determine the action at each time step has learned coefficients in its objective function, but fixed constraints such as that the total quantity supplied from each of the 24 stocks must be not be more than the current stock amount.

The blood bank scenario is an abstraction of a real-world situation where the objectives to be maximized, both short-term and long-term, are subject to debate. Previous research has used measures of long-term success that are intuitively reasonable, but not mathematically consistent with the immediate reward functions used. Table 2 shows two different immediate reward functions. Each entry is the benefit accrued by meeting one unit of demand with supply of a certain nature. The middle column is the short-term reward function used in previous work, while the right column is an alternative that is more consistent with the long-term evaluation measure used previously (described below).

For training and for testing, a trajectory is a series of periods. In each period, the agent sees a demand vector drawn randomly from a specific distribution. The agent supplies blood according to its policy and sees its immediate reward as a consequence. Note that the immediate reward is defined by a function given in Table 2, but the agent does not know this function. All training and testing is in the standard reinforcement learning scenario, where the agent sees only random realizations of the MDP.

Then, blood remaining in stock is aged by one period, blood that is too old is discarded, and fresh blood arrives according to a different probability distribution. Training is based on 1000 trajectories, each 10 periods long, in which the agent follows a greedy policy.[2] Testing is based on trajectories of

[2] The greedy policy is the policy that supplies blood to maximize one-step reward, at each time step. Many domains are like the blood bank domain in that the optimal

Table 3. Success of alternative learned policies.

	average unmet A+ demand	frequency of severe unmet A+ demand
greedy policy	7.3%	46%
policy of [Yu, 2007]	7.9%	30%
bilinear method (i)	18.4%	29%
bilinear method (ii)	7.55%	12%

Notes: Column headings are explained in the text. Current and previous percentages are not directly comparable, because of differing numbers of periods and other differences.

the same length where supply and demand vectors are drawn from the same distributions, but the agent follows a learned policy.

According to [Yu, 2007], the percentage of unsatisfied demand is the best long-term measure of success for a blood bank. If a small fraction of demand is not met, that is acceptable because all high-priority demands can still be met. However, if more than 10% of demand for any type is not met in a given period, then some patients may suffer seriously. Therefore, we measure both the average percentage of unmet demand and the frequency of unmet demand over 10%. Again according to [Yu, 2007], the A+ blood type is a good indicator of success, because the discrepancy between supply and demand is greatest for it: on average, 34.00% of demand but only 27.94% of supply is for A+ blood.

Table 3 shows the performance reported by [Yu, 2007], and the preliminary performance of the bilinear method using the two reward functions of Table 2. Given the original immediate reward function, (i), fitted Q iteration with a bilinear Q function satisfies on average less of the demand for A+ blood. However, the original reward function does not include an explicit penalty for failing to meet demand. The implementation of [Yu, 2007] is tuned in a domain-specific way to satisfy more demand for A+ blood even though doing so does not accrue explicit immediate reward. The last column of Table 2 shows a different immediate reward function that does penalize failures to meet demand. With this reward function, the bilinear approach learns a policy that reduces the frequency of severe failures.

7 Discussion

Reinforcement learning has been an important research area for several decades, but it is still a major challenge to use training examples efficiently, to be computationally tractable, and to handle action and state spaces that are high-dimensional and continuous. Fitted Q iteration with a bilinear Q function can

one-step policy, also called myopic or greedy, can be formulated directly as a linear programming problem with known coefficients. With bilinear fitted Q iteration, coefficients are learned that formulate a long-term policy as a problem in the same class. The linear representation is presumably not sufficiently expressive to represent the optimal long-term policy, but it is expressive enough to represent a policy that is better than the greedy one.

meet these criteria, and initial experimental results confirm its usefulness. The constraints on what actions are legal in each state are accommodated in a straightforward way by the linear programming algorithm that computes the optimal action vector at each time step. We intend to explore this approach further in the blood bank domain and in other large-scale reinforcement learning scenarios.

Acknowledgments

The author is grateful to Vivek Ramavajjala for the initial implementation of the bilinear Q iteration method, used for the experiments described above. Thanks are also due to anonymous referees and others for beneficial comments that led to definite improvements in the paper.

References

[Chakraborty et al., 2008] Chakraborty, B., Strecher, V., and Murphy, S. (2008). Bias correction and confidence intervals for fitted Q-iteration. In *NIPS Workshop on Model Uncertainty and Risk in Reinforcement Learning*.

[Chang et al., 2010] Chang, Y. W., Hsieh, C. J., Chang, K. W., Ringgaard, M., and Lin, C. J. (2010). Training and testing low-degree polynomial data mappings via linear SVM. *Journal of Machine Learning Research*, 11:1471–1490.

[De Farias and Van Roy, 2003] De Farias, D. P. and Van Roy, B. (2003). The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865.

[Dietterich, 2009] Dietterich, T. G. (2009). Machine learning and ecosystem informatics: Challenges and opportunities. In *Proceedings of the 1st Asian Conference on Machine Learning (ACML)*, pages 1–5. Springer.

[Džeroski et al., 2001] Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine Learning*, 43(1):7–52.

[Ernst et al., 2005] Ernst, D., Geurts, P., and Wehenkel, L. (2005). Tree-based batch mode reinforcement learning. *Journal of Machine Learning Research*, 6(1):503–556.

[Gordon, 1995a] Gordon, G. J. (1995a). Stable fitted reinforcement learning. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1052–1058.

[Gordon, 1995b] Gordon, G. J. (1995b). Stable function approximation in dynamic programming. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 261–268.

[Hannah and Dunson, 2011] Hannah, L. A. and Dunson, D. B. (2011). Approximate dynamic programming for storage problems. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[Judd and Solnick, 1994] Judd, K. L. and Solnick, A. J. (1994). Numerical dynamic programming with shape-preserving splines. Unpublished paper from the Hoover Institution available at `http://bucky.stanford.edu/papers/dpshape.pdf`.

[Lagoudakis and Parr, 2003] Lagoudakis, M. G. and Parr, R. (2003). Least-squares policy iteration. *Journal of Machine Learning Research*, 4:1107–1149.

[Lazaric et al., 2007] Lazaric, A., Restelli, M., and Bonarini, A. (2007). Reinforcement learning in continuous action spaces through sequential Monte Carlo methods. In *Advances in Neural Information Processing Systems 20 (NIPS)*. MIT Press.

[Melo and Lopes, 2008] Melo, F. S. and Lopes, M. (2008). Fitted natural actor-critic: A new algorithm for continuous state-action MDPs. In *Machine Learning and Knowledge Discovery in Databases, European Conference (ECML/PKDD)*, volume 5212 of *Lecture Notes in Computer Science*, pages 66–81. Springer.

[Murphy, 2005] Murphy, S. A. (2005). A generalization error for Q-learning. *Journal of Machine Learning Research*, 6:1073–1097.

[Neumann, 2008] Neumann, G. (2008). Batch-mode reinforcement learning for continuous state spaces: A survey. *ÖGAI Journal*, 27(1):15–23.

[Pazis and Lagoudakis, 2009] Pazis, J. and Lagoudakis, M. G. (2009). Binary action search for learning continuous-action control policies. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML)*, pages 100–107.

[Pazis and Parr, 2011] Pazis, J. and Parr, R. (2011). Generalized value functions for large action sets. In *Proceedings of the International Conference on Machine Learning (ICML)*.

[Powell, 2007] Powell, W. B. (2007). *Approximate Dynamic Programming*. John Wiley & Sons, Inc.

[Riedmiller, 2005] Riedmiller, M. (2005). Neural fitted Q iteration–first experiences with a data efficient neural reinforcement learning method. In *Proceedings of the 16th European Conference on Machine Learning (ECML)*, pages 317–328.

[Simester et al., 2006] Simester, D. I., Sun, P., and Tsitsiklis, J. N. (2006). Dynamic catalog mailing policies. *Management Science*, 52(5):683–696.

[Smart and Kaelbling, 2000] Smart, W. D. and Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 903–910.

[Stachurski, 2008] Stachurski, J. (2008). Continuous state dynamic programming via nonexpansive approximation. *Computational Economics*, 31(2):141–160.

[Sutton and Barto, 1998] Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning: An introduction*. MIT Press.

[Todorov, 2009] Todorov, E. (2009). Efficient computation of optimal actions. *Proceedings of the National Academy of Sciences*, 106(28):11478–11483.

[van Hasselt, 2010] van Hasselt, H. P. (2010). Double Q-learning. *Advances in Neural Information Processing Systems (NIPS)*, 23.

[Viviani and Flash, 1995] Viviani, P. and Flash, T. (1995). Minimum-jerk, two-thirds power law, and isochrony: converging approaches to movement planning. *Journal of Experimental Psychology*, 21:32–53.

[Yu, 2007] Yu, V. (2007). Approximate dynamic programming for blood inventory management. Honors thesis, Princeton University.