Off-policy Learning with Linear Action Models: An Efficient "One-Collection-For-All" Solution

Hengshuai Yao HENGSHUA@CS.UALBERTA.CA
Department of Computing Science
University of Alberta
Edmonton, AB, T6G2E8

Abstract

We propose a model-based off-policy learning method that can be used to evaluate any target policy using data collected from arbitrary sources. The key of this method is a set of linear action models (LAM) learned from data. The method is simple to use. First, a target policy tells what actions are taken at some features and LAM project what would happen for the actions. Second, a convergent off-policy learning algorithm such as LSTD and gradient TD algorithms evaluates the projected experience. We focus on two off-policy learning algorithms with LAM, i.e., the stochastic LAM-LSTD and the deterministic LAM-LSTD. Empirical results show that the two LAM-LSTD algorithms give more accurate predictions for various target policies than the on-policy LSTD learning. LAM based off-policy learning algorithms are also exclusively useful in difficult control tasks where one could not collect sufficient "on-policy samples" for on-policy learning. This work leads us to advocate using off-policy learning to evaluate many policies in place of on-policy learning, improving the efficiency of using data.

1. Introduction

Off-policy learning, which aims to evaluate a policy based on the data generated/collected from another policy, is an interesting problem in reinforcement learning (RL) (Watkins, 1989; Sutton & Barto, 1998; Lagoudakis & Parr, 2003). Off-policy learning is a very important way of increasing our knowledge about the world. The amazing feature of off-policy learning is that a single stream of data from arbitrary sources can provide us knowledge about many policies. Therefore off-policy learning is an important way of improving the efficiency of using sample.

Though charming in definition, learning many policies from a single-stream of data has rarely been practiced in RL, mainly because off-policy learning has pitfalls and is inherently hard. The difficulty lies in that the distribution of the data is from the policy that generates/collects the data, which causes many RL algorithms to diverge (Sutton & Barto, 1998). Importance sampling was first proposed to correct the distribution of data for off-policy learning (Precup et al., 2001). However, it has a high variance in estimation. Recently, several proposed gradient temporal difference (TD) methods with linear function approximation are proved to converge for off-policy learning (Sutton et al., 2009).

In this paper, we study off-policy learning in an offline setting, in which a data set of samples are collected before hand using an arbitrary policy. We propose a general approach for off-policy learning, which stands out from existing off-policy learning solutions in that it is model-based. The key component of this framework is a set of approximate action models with linear function approximation, which are called *linear action models(LAM)* for short. LAM belong to the family of linear models. Boyan built a compressed model for the prediction problem, and gave a new interpretation for the previous LSTD (Bradtke & Barto, 1996), and extended it to eligibility traces (Boyan, 2002). It was shown that the fixed points of model-free and model-based value function approximation are equivalent given the same set of features (Parr et al., 2008). LAM differs from these models in that it models the effects of actions with linear function approximation. LAM was first explored in a linear Dyna algorithm

Appearing in *Planning and Acting with Uncertain Models Workshop at the 28th ICML*, Bellevue, WA, USA, 2011.

for online planning and control (Sutton et al., 2008). In their algorithm, learning, modeling and planning proceed simultaneously. Action selection in learning and planning is performed according to LAM. However, their paper focused mainly on the on-policy prediction problem, and LAM were only briefly studied. They also used a gradient descent algorithm to learn LAM, which is slow and requires tuning a step-size. The gradient descent method is problematic also in that the induced LAM can be biased because of the choice of the step-size parameter.

Apparently LAM is policy-indepdent. It is this policy independence property that makes it suitable for off-policy learning. Because of policy independence, LAM-based off-policy learning does not have to use importance sampling to correct the behavior/collection policy for the target policy. So LAM is free of the high variance caused by importance sampling. Because of policy independence, LAM-based off-policy learning considers the target policy not at the time of modeling, but only when learning is requested. This clear separation of modeling from learning is the key to the simplicity and effectiveness of the proposed off-policy learning solution. LAM-based off-policy learning is simple to use. First, LAM are learned from a given set of samples with some chosen features, using an efficient least-squares method that can guarantee the quality of LAM. Second, a target policy tells what actions are taken on some features. We then apply LAM to project what would happen from these features according to the policy. Third, we use an algorithm to evaluate the projected experience. Notice the learning is off-policy, so we need convergent off-policy learning algorithms such as gradient TD algorithms or LSTD algorithm. We focus on the use of LSTD since it is efficient in using samples, and does not require tuning a step-size. However, other learning algorithms such as gradient TD can also be used to evaluate the projected experience by LAM. The emphasis of this paper is not on the comparisons between LSTD and gradient TD since it is already well known that least-squares methods are more data efficient (Bradtke & Barto, 1996; Boyan, 2002; Xu et al., 2002).

We demonstrate that off-policy learning can be much more accurate than on-policy learning. Our two off-policy learning algorithms perform very well in evaluating various target policies. In particular, for those policies that are *ill distributed*, according to which some actions are rarely taken or some states are rarely visited, the advantages of our algorithms are very pronounced. Notice that the problem is inherently hard because of the rareness caused by the nature of these policies. In a related rareness problem studied by (Frank et al., 2008), rare events occur independently of actions. The rareness we study in this paper is caused by the policies themselves, and is much more common in RL.

Algorithm 1 Learning LAM from a set of samples using a least-squres method.

Input: a data set, $\mathbb{D} = \{< \phi_i, a_i, \phi_{i+1}, r_i >\}$, or $\mathbb{D}_s = \{(s_i, a_i, s_{i+1}, r_i >\}$.
Output: a set of LAM, $\{< F^a, f^a >\}$.
Initialize H^a, E^a and e^a for all a
for $i = 1, 2, \ldots, d$ **do**
  Read the transition: $< \phi_i, a_i, \phi_{i+1}, r_i >$
  if using $\mathbb{D}_s$
    Set $\phi_i = \phi(s_i)$, $\phi_{i+1} = \phi(s_{i+1})$
  end
  for $a = a_i$, update LAM of a by
  $H^a = H^a + \phi_i \phi_i^T$
  $E^a = E^a + \phi_{i+1} \phi_i^T$
  $e^a = e^a + \phi_i r_i$
end
for all a, solve LAM by least-squares:
  $F_d^a = E^a (H^a)^{-1}$
  $f_d^a = (H^a)^{-1} e^a$

2. Learning LAM

Suppose the state space is denoted by $\mathbb{S}$, and we have N states. First we are given a data set of samples, $\mathbb{D} = \{< \phi_i, a_i, \phi_{i+1}, r_i >\}$, [1] where ϕ_i is some feature at which action a_i is taken, ϕ_{i+1} is the resulting feature, and r_i is the resulting reward, $i = 1, 2, \ldots, d$, $d = |\mathbb{D}|$. The samples can be collected from a single policy or many different policies, by a single agent or many different agents. There is no restriction on the data set. However, to guarantee the quality of LAM and off-policy learning, the data set should contain sufficient samples.

LAM are learned using linear function approximation. Given n ($n \leq N$) feature functions $\varphi_j(\cdot) : \mathbb{S} \mapsto \mathbb{R}$, $j = 1, \ldots, n$, the feature vector (*feature* for short) of state i is $\phi(i) = [\varphi_1(i), \varphi_2(i), \ldots, \varphi_n(i)]^T$. Let Φ be the feature matrix whose entries are $\Phi_{i,j} = \varphi_j(i)$, $i = 1, \ldots, N$; $j = 1, \ldots, n$. We assume the columns of Φ are linearly independent. Each LAM is composed of a matrix and a vector, $< F^a_{n \times n}, f^a_{n \times 1} >$, where F^a approximates the transition dynamics and f^a approximates the rewards of taking action a in the feature

[1] Notice that the data set can also be the experience of transitioning among states, $\mathbb{D}_s = \{< s_i, a_i, s_{i+1}, r_i >\}$, where $s_i, s_{i+1} \in \mathbb{S}$.

Algorithm 2 The stochastic LAM-LSTD algorithm: a simulation-based-projection on/off-policy learning with the pre-learned LAM. No iteration is required.

Input: a data set of features, $\mathbb{D}_\phi = \{\phi_i\}$, a set of LAM, $\{< F^a, f^a >\}$ learned from $\mathbb{D}$; and a target policy π.
Output: a parameter vector θ for policy π.
for $i = 1, 2, \ldots, d$ **do**
  Read ϕ_i
  Select an action a according to π at ϕ_i
    /* for on-policy learning, $a = a_i$*/
  $\tilde{\phi}_{i+1} = F^a \phi_i$
  $\tilde{r}_i = \phi_i^T f^a$
  $A = A + \phi_i (\gamma \tilde{\phi}_{i+1} - \phi_i)^T$
  $b = b + \phi_i \tilde{r}_i$
end
$\theta = -A^{-1} b$

space. Algorithm 1 shows an efficient least-squares method of learning LAM.

3. Off-policy Learning Algorithms with LAM

3.1. A Simulation-based Method

The first algorithm, a simulation-based method, is shown in Algorithm 2. The algorithm is run on a data set of features, $\mathbb{D}_\phi = \{\phi_i\}$. Thus at this stage the transitioning experience is no longer necessary. In the experiments, we set $\mathbb{D}_\phi$ to be the set of features where the transitioning samples were collected from. That is, $\mathbb{D}_\phi = \{\phi_i | \phi_i \in \mathbb{D}\}$. This, however, is not a constraint, as one can choose freely for $\mathbb{D}_\phi$.

Notice that $F^a \phi$ is the expected next feature, and $\phi^T f^a$ is the expected reward of taking a at ϕ. To evaluate a policy with LAM, one follows the policy, generating an action a at a feature ϕ, and do the projection operation, which gives the imaginary transition experience, $< \phi, \phi^T f^a, F^a \phi >$. Notice that evaluating the imaginary experience is an off-policy learning problem. To guarantee convergence and data efficiency, we use LSTD for policy evaluation in Algorithm 2.

Notice that Algorithm 2 can also be used for on-policy learning, in which the target policy π is also the policy we used to collect the samples, i.e., $\pi(\phi_i) = a_i$, $i = 1, 2, \ldots, d$. On-policy learning can then be based on the projected experience under the action selected according to the policy.

For off-policy learning, Algorithm 2 can be used to evaluate a target policy known beforehand, or a target policy only known when processing samples, such as greedy policies. In a recent paper (Yao, 2010), we proposed approximate policy iteration using LAM. In that case, learning is still off-policy, and the goal is to evaluate the greedy/optimal policy. The focus of this paper is on evaluation of various policies that are generally not greedy or optimal.

Algorithm 3 The deterministic LAM-LSTD algorithm: an analytical-projection based off-policy learning with the pre-learned LAM. No iteration or simulation is required.

Input, Output: the same as Algorithm 2
for $i = 1, 2, \ldots, d$ **do**
  Read ϕ_i
  Set $\tilde{\phi}^\pi_{i+1} = 0, \tilde{r}^\pi_i = 0$
  for each action a
    $\tilde{\phi}^\pi_{i+1} = \tilde{\phi}^\pi_{i+1} + \pi(\phi_i, a) F^a \phi_i$
    $\tilde{r}^\pi_i = \tilde{r}^\pi_i + \pi(\phi_i, a) \phi_i^T f^a$
  end
  $A = A + \phi_i (\gamma \tilde{\phi}^\pi_{i+1} - \phi_i)^T$
  $b = b + \phi_i \tilde{r}^\pi_i$
end
$\theta = -A^{-1} b$

3.2. A Deterministic Method

If the target policy is known, Algorithm 2 can be made more efficient in projection. We can generate the next feature and reward for a given feature under the target policy, by taking advantage of the target policy and LAM.

Given a feature, we can project the experience under the target policy at once, without simulating step by step. In particular, for a feature ϕ_i, the expected next feature according to policy π is

$$\tilde{\phi}^\pi_{i+1} = \sum_a \pi(\phi_i, a) F^a \phi_i,$$

and the reward is

$$\tilde{r}^\pi_i = \sum_a \pi(\phi_i, a) \phi_i^T f^a.$$

Algorithm 3 shows this more efficient method. The algorithm is deterministic and does not require any simulation. The algorithm can also be used for on-policy learning if the collection policy (which is π) is known. However, in practice, the samples may be from various sources (e.g., collected by many agents following different policies), and hence the collection policy can be unknown. In this case, Algorithm 3 is not applicable for on-policy learning, and one has to use Algorithm 2.

4. Empirical Results

4.1. Boyan MDP

The problem is slightly modified from Boyan chain (Boyan, 2002). We interpret it as a MDP problem. At each state, there are two actions available. Action a_1 "walks" the state i to state $i-1$. Action a_2 "jumps" state i to state $i-2$ except that at state 1 it takes the agent to state 0. Both actions are deterministic: taking an action leads to the intended state without any problem.

We consider evaluating the following three target policies:

policy 1: walking with probability 50%, and jumping with 50% at each state (this is the original policy of Boyan chain); and

policy 2: walking with probability 90%, and jumping with 10% at each state;

policy 3: walking with probability 0.00001%, and jumping with 99.99999% at each state.

We compared on/off-policy learning of the three policies. For on-policy learning of a policy, samples were collected from a number of episodes following the policy. For off-policy learning, samples were collected in a number of episodes following a purely random policy (taking uniformly random actions at each state). For both on-policy and off-policy learning, 1000 episodes of samples were collected. All episodes start from state 12 and terminate in state 0. In both on-policy and off-policy evaluation, LSTD was used. For off-policy learning, two LAM were first learned from the samples using Algorithm 1. Then we projected features ϕ_i in the samples and applied LSTD to evaluate the projected experience. We also compared the two ways of projecting experience: the stochastic way (Algorithm 2) and the deterministic way (Algorithm3). Notice that the original features by Boyan can only represent the value function of policy 1. In order to represent all the policies exactly, we also used the tabular features in addition to the original linear interpolation features.

Figure 1 shows the results of evaluating policy 1 using the original linear features. For this policy, on-policy learning and the stochastic LAM-LSTD have a similar convergence rate, for two reasons. First, the state/action distribution under policy 1 is very smooth, and the learned LAM do not provide a significant advantage in the coverage of state space over on-policy learning. Second, they both depend on the sampling of the policy in policy evaluation. However, the deterministic LAM-LSTD wins on the second aspect. The deterministic LAM-LSTD does not depend on the sampling of the policy in projecting experience for policy evaluation, and the convergence is very fast seen from the figure. The case of tabular features is similar and thus the figure is not included here.

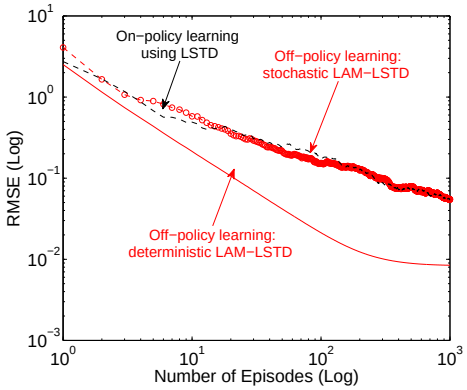

Figure 1. The Boyan MDP: learning policy 1 with the linear features.

Figure 2 shows the results of evaluating policy 2 using the linear interpolation features. For this policy, the state/action distribution under the policy is not very smooth. Typically, the "jumping" action is more frequently taken, and half of the states are more frequently visited. Hence the accuracy of on-policy learning is bottlenecked by those infrequently visited states. The LAM have a much finer accuracy because it is learned from the data collected from the random policy which is almost uniformly distributed in both states and actions. Thus both the stochastic and deterministic LAM-LSTD converge faster than on-policy learning. The two have a similar convergence rate because the RMSE quickly achieves the bound that is enforced by the features. Figure 3 shows the results of using the tabular features. This time the deterministic LAM-LSTD is much faster than the stochastic LAM-LSTD, since the tabular features can represent value functions exactly.

We continue to learning policy 3. Notice that policy 3 is an extremely ill distributed policy. Because the "walking" action is rarely taken, it requires more samples for on-policy learning to reflect the dynamics of policy 3 than off-policy learning. If one learns policy 3 using samples from itself (on-policy evaluation), the convergence is very slow since it can almost only learn the value functions of states $12, 10, 8, 6, 4, 2, 0$. This kind of policies does exist in practice. For example, Koller and Parr (2000) showed that the uneven distribution of states/actions under a policy can cause problems for approximate policy iteration.

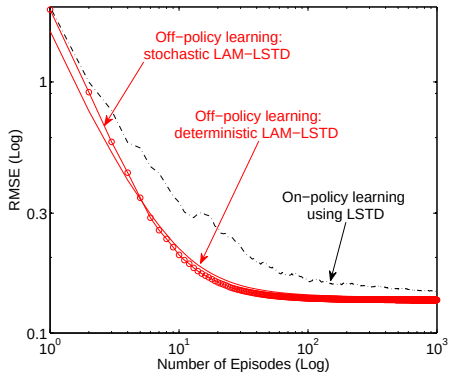

Figure 2. The Boyan MDP: learning policy 2 with the linear features.

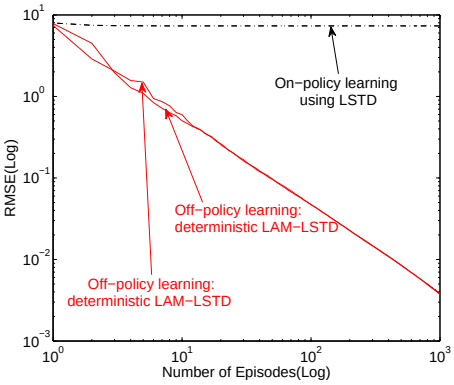

Figure 4. The Boyan MDP: learning policy 3 with tabular features.

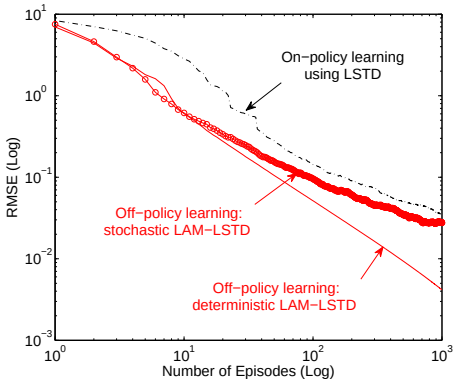

Figure 3. The Boyan MDP: learning policy 2 with tabular features.

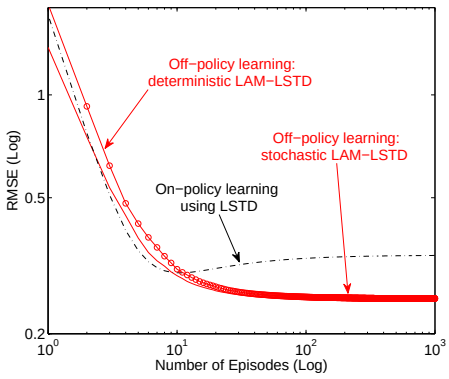

Figure 5. The Boyan MDP: learning policy 3 with the linear features.

In this case, policy 3 is almost the optimal policy, which however is poorly evaluated using on-policy learning. The value function of policy 3 is almost [-18, -17, -15, -14, -12, -11, -9, -8, -6, -5, -3, -2, 0]. Figure 4 shows that on-policy learning is only able to learn the value functions of states $12, 10, 8, 6, 4, 2, 0$, because the other states are rarely seen in the episodes. In fact, for on-policy learning the estimation of these value functions remains at the initial guess, which was 0 for the experiment because LSTD's data structures were initialized to 0. Then the values of the rarely seen states are rarely updated, leading to a rather large error for on-policy learning. The problem is inherent with on-policy learning. In extreme situations like this, it takes an agent a life time in getting a good estimation of the rarely visited states using on-policy learning.

Off-policy learning just doesn't have such a problem. It is not influenced by whatever frequencies that the states/actions are visited/taken by target policies. As long as there are good LAM, evaluation of the target policies is as accurate as the features permit. The quality of LAM is dependent on whether the collection policy can collect sufficient samples. Therefore in practice, designing good collection policies is a key issue in learning LAM, which is however beyond the topic of this paper. Figure 4 shows that the two versions of LAM-LSTD both perform very well. Their RMSEs are close because the stochastic LAM-LSTD is almost deterministic. Finally, Figure 5 shows the results of using the linear features. On-policy learning does a better job than the tabular case simply because of the generalization in the features.

4.2. Grid-world

A 11×11 grid-world example is shown in Figure 6. There are four actions available in each state. An action moves the agent one grid in the intended direction, except that when leading to the boundary the agent remains in the original state. Reaching the left-up and right-down corners receives a reward 1.0; reaching the

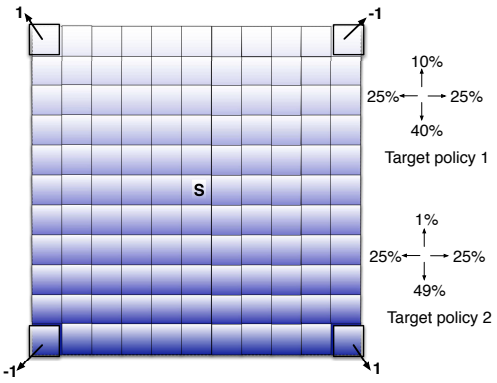

Figure 6. A Grid-world example. The darker is a region, the more the region is covered by the target policies.

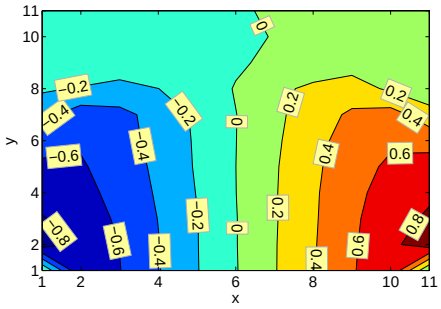

Figure 7. On-policy learning of target policy 1 on Grid-world: the contour of the averaged learned policy.

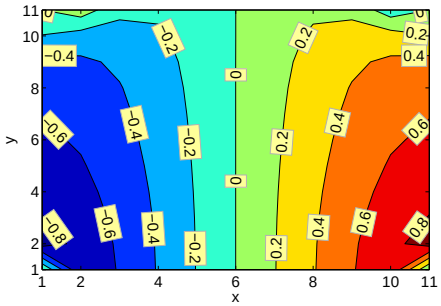

Figure 8. Off-policy learning of target policy 1 on Grid-world: the contour of the learned policy (averaged over 30 runs).

left-down and right-up corners receives a reward -1.0; and the other rewards are 0. The task is to evaluate two target policies as shown in the figure. Each run consists of 100 episodes of data up to 1000 steps. In each episode, the agent started from the position "S" in the figure, and behaved according to the target-policy (for on-policy learning) or the collection policy (for off-policy learning). For on-policy learning experiments, we used LSTD for evaluating the two target policies. For off-policy learning experiments, the agent followed a purely random policy for collection of samples, and used the deterministic LAM-LSTD for evaluating the projected experience by LAM. Features are tabular for both on-policy and off-policy learning.

The on-policy learning result for policy 1 is shown in Figure 7, and the off-policy learning result is shown in Figure 8. The RMS errors of the two learning were compared in Figure 9. The results in all the three figures were averaged over 30 runs. The results are very intuitive. For on-policy learning, the agent was exploring the lower part of the world much more often; the upper part was not well covered. The effect can be clearly seen from the boundary, $x = 6$. Because the policy goes to the left and right equally often, so along $x = 6$ the values of the states are 0 (resulting from the fact that the rewards on the left and right of the world have the same magnitude but opposite signs). For the states $(x = 6, y \leq 6)$, their values were learned accurately, reflected in that the 0-value boundary is almost $x = 6$. However, for the states $(x = 6, y > 6)$, their values were learned poorly, reflected in that the 0-value boundary is twisted from $x = 6$. This, in general, is an inherent problem with on-policy learning, regardless what algorithms are used. [2]

The off-policy learning doesn't have this problem. In Figure 8, the 0-value boundary is sharply close to $x = 6$. Also, the learned values of the upper states are much more accurate than those by on-policy learning. This leads to a much smaller RMSE for off-policy learning. The problem of on-policy learning becomes much severe for those policies that rarely visit some states. For the second policy, the agent goes to the lower part almost sure, leaving the learning of the upper states almost a gap. This causes a large learning error for the value function, as shown in Figure 9.

4.3. A Difficult Control Problem

We studied the bicycle-riding-balancing task, which is considered as a difficult problem in literature (Lagoudakis & Parr, 2003). The state variable is $(\vartheta, \dot{\vartheta}, \omega, \dot{\omega}, \ddot{\omega}, \psi, d, x_b, y_b, x_f, y_f)$, where θ is the angle of the handler (abusing notation from the weight vector), ω is the vertical angle of the bicycle, d is the

[2] Function approximation may help this problem, but this depends on if the chosen features can generalize appropriately to those regions not covered.

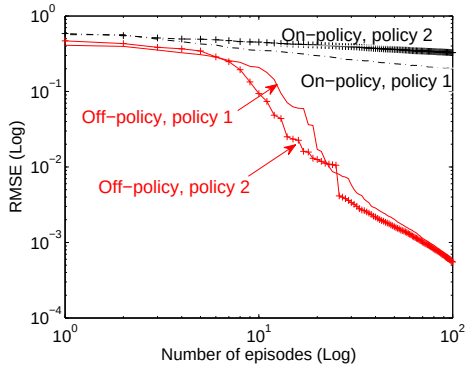

Figure 9. The averaged RMSE of on/off-policy learning on Grid-world.

distance to goal, ψ is the angle of the bicycle to the goal, and $(x_b, y_b)/(x_f, y_f)$ is the back/front tyre position. The actions are the torque applied to the handler, $\tau \in \{-2, 0, 2\}$; and the displacement of the rider, $v \in \{-0.02, 0, 0.02\}$. At least one of τ and v is restricted to 0. This leads to 5 actions in total. If ω is bigger than $\pi/15$, the bicycle falls over and the episode stops. The reward signal is updated according to

$$r_t = \left|\frac{15\omega_{t-1}}{\pi}\right|^2 - \left|\frac{15\omega_t}{\pi}\right|^2 + \frac{d_{t-1} - d_t}{100},$$

where d_t is the distance from the bicycle to the goal at the t time step. The discount factor is 0.80. The state feature is the same as used for a single action in LSPI, comprising 20 basis functions:

$$[1, \omega, \dot{\omega}, \omega^2, \dot{\omega}^2, \omega\dot{\omega}, \theta, \dot{\theta}, \theta^2, \dot{\theta}^2,$$

$$\theta\dot{\theta}, \omega\theta, \omega\theta^2, \omega^2\theta, \psi, \psi^2, \psi\theta, \bar{\psi}, \bar{\psi}^2, \bar{\psi}\theta]^T,$$

where $\bar{\psi} = \pi - \psi$ if $\psi > 0$, otherwise $\bar{\psi} = -\pi - \psi$. The problem is difficult partially because there is a noise added to the displacement action, which follows a uniformly distribution in $[-0.02, 0.02]$.

We collected a data set of 2500 episodes using a uniformly random policy, each comprising 20 steps of samples. Five linear action models were learned using the least-squares algorithm. We used LAM to evaluate the following three policies: (1) policy 1, which takes the five actions with probabilities $0.4, 0, 0.4, 0.1, 0.1$; (2) policy 2 is coined. With probability 0.1, select the action that minimizes the predicted direction to goal (ψ), riding to the goal; with probability 0.9, select the action that minimizes the predicted vertical angle of the bicycle (ω), balancing. The predictions are made according to LAM. For a feature ϕ, we have five feature projections, $\tilde{\phi}^a = F^a\phi$. For each a, $\tilde{\phi}^a(2)$ gives the predicted ω and $\tilde{\phi}^a(15)$ gives the predicted ψ after taking the action; (3) policy 3 is the greedy policy. At a feature ϕ in the samples, we select

$$a^* = \arg\max_a Q(\phi, a) = \arg\max_a \left\{\phi^T f^a + \gamma(F^a\phi)^T\theta\right\}, \quad (1)$$

(where θ is the policy weight vector). Then the projected experience, $(\phi, a^*, \tilde{r} = \phi^T f^{a^*}, \tilde{\phi} = F^{a^*}\phi)$ is fed to LSTD. Policies 1 and 2 can be evaluated using the deterministic or stochastic LAM-LSTD. We added an outer iteration loop for learning policy 3 for convergence (in a few iterations). After the value functions (or parameters θ) of the three policies were learned, we used them for control, selecting actions online according to (1), in which ϕ now takes real time features.

Figure 10 shows the trajectories of the three controllers. Notice that policy 1 is a poor one, according to which the bicycle fell over in less than one hundred of steps most of the time. Surprisingly, action selection according to the value function of policy 1 (i.e., θ_1) in the way of equation (1) can balance the bicycle for at least $72,000$ steps, as shown in the figure. This is because the value function of policy 1 approximates the shape of the balancing policy well, though the policy itself is poor. Figure 11 shows the learned value function of policy 1: $V_1(s) = \phi(s)^T\theta_1$, where s (only the ω is shown) takes the states in 100 episodes of the samples. For most trajectories, V_1 increases as $|\omega|$ reduces. Thus action selection through maximizing V_1 has the effect of balancing the bicycle. That on the few trajectories where V_1 does not increase as $|\omega|$ decreases is because the value function also depends on other factors such as the angle to the goal.

Figure 12 and Figure 13 show the value function of policy 2 versus the sampled back tyre positions of the bicycle on the domain. The left and right plots used the same colors and markers for values of the same states from samples, so there is a correspondence between the two plots. For example, the largest values, on the top of both plots, corresponds to two episodes of states whose x_b is non-negative and y_b close to 0. Figure 12 shows that V_2 generally increases as x_b increases. Figure 13 shows V_2 generally increases as $|y_b|$ decreases. [3] Thus action selection through maximizing V_2 has the effect of pushing the bicycle along the positive direction in the x-axis. The balancing aspect of policy 2 is similar to policy 1, and thus not shown. The shape of V_3 is similar to V_2, and thus omitted as well.

Off-policy learning on complex control problems is exclusively important because on-policy learning is of-

[3] Again there are exceptions for some episodes because of the dependence on other factors.

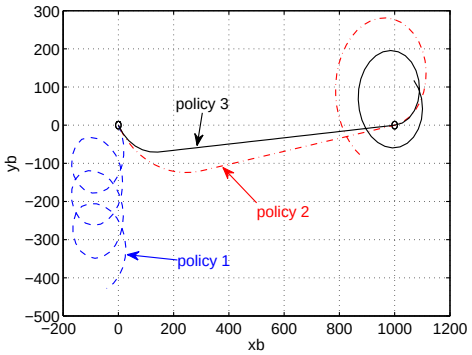

Figure 10. The Bicycle domain: trajectories of acting through maximizing the value functions of the three policies. The value functions were learned using LAM-LSTD off-policy learning.

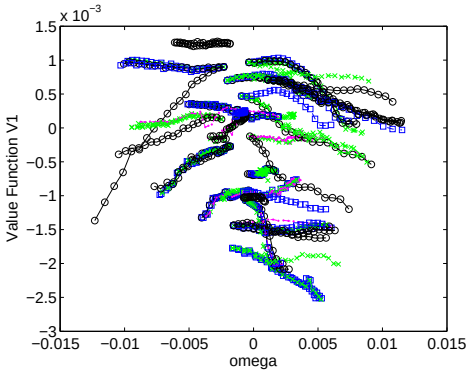

Figure 11. The Bicycle domain: the value function of policy 1 versus ω, shown on 100 episodes of samples. Notice maximizing the value function has the effect of reducing ω most of the time.

ten difficult. For instance, in this example behaving according to policy 1 mostly fell under 100 episodes. Short episodes of samples under target policies can be very common since the policies can fail long before the goal of an agent is reached. In cases where important rewards are given upon reaching the goal, which are the most common in reinforcement learning, on-policy learning of many policies could not collect sufficient good samples. [4] Moreover, on-policy evaluation requires as many sets of samples as the policies, for which off-policy learning can use only one set of samples—off-policy learning is just more data efficient.

[4]This does not create a problem for this example since important rewards are given not upon reaching the goal but upon driving along the direction to it.

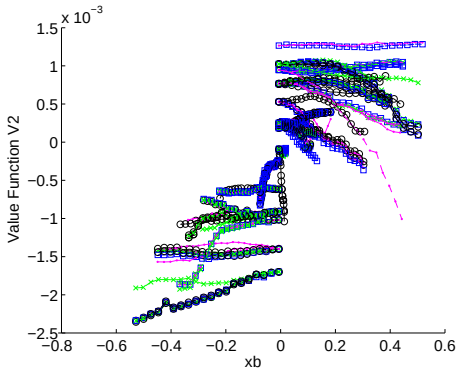

Figure 12. The Bicycle: the learned value function for policy 2 versus x_b.

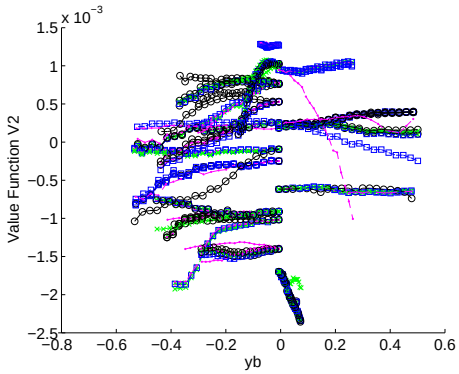

Figure 13. The Bicycle: the learned value function for policy 2 versus y_b. Action selection through maximizing V_2 has the effect of pushing to the positive direction in the x-axis.

5. Discussion and Conclusion

Off-policy learning is an interesting topic, which has been pursued since the early days of RL. The goal of off-policy learning is very charming: using a single stream of data collected from an arbitrary policy to evaluate any other policy. Researchers have proposed importance sampling (Precup et al., 2001) and gradient descent methods (Sutton et al., 2009) to this goal. These methods are generally model-free.

We proposed a model-based method for efficient off-policy learning. Given a data set of samples, we first learn a set of linear action models. The linear action models are then used to project the experience under a target policy. Off-policy learning algorithms such as LSTD and GTD can then be applied to evaluating the projected experience. We proposed two off-policy learning algorithms with LAM, based on two ways of projecting experience. Empirical results of evaluating various policies show that our algorithms performed

very well.

Our results suggest that off-policy learning is a promising way of improving the efficiency of using samples. As long as collection of data is allowed on a problem, off-policy learning can replace on-policy learning in evaluating various policies. For RL problems where interacting with the environment is time consuming or money expensive, our method provides a very cheap solution, a *one-collection-for-all* solution: one time of data collection from interaction with the environment can provide accurate evaluation of as many policies as a RL researcher is interested in. We noticed that the linear Gaussian MDP model (Bowling et al., 2008) is also action-dependent but policy-independent. The linear MDP model is learned from samples, and then used to explicitly construct the policy models for approximate policy iteration using sigma-points methods. As noted by Bowling et. al. (2008), this method can get rid of samples after the model is learned and is computationally faster than LSPI which memorizes and sweeps the samples at each iteration. This observation also holds for an extension of our method to approximate policy iteration (see (Yao, 2010)). The major difference of our method is that we do not construct the policy models explicitly, but use the LAM to project samples for different policies, which is more efficient in both computation and memory.

Though the model-based property is the uniqueness of our approach to off-policy learning, we didn't go into comparing with the model-free approach. The model-based approach is generally known to be more data efficient, but more complex in per-time-step complexity (Moore & Atkeson, 1993; Kaebling et al., 1996; Sutton & Barto, 1998; Sutton et al., 2008). For example, LSTD produces more accurate predictions than TD, but its per-time-step computational complexity is higher than TD (Bradtke & Barto, 1996; Boyan, 2002; Xu et al., 2002). These conclusions still hold for the comparisons between our model-based algorithms and model-free algorithms of off-policy learning. Our solution is more data efficient, but is $O(n^2)$ per time step in computation, for which gradient TD is $O(n)$. Furthermore, our model-based approach to off-policy learning does not exclude the use of model-free off-policy learning algorithms. For example, GTD algorithms can be used to evaluate the projected experience by LAM, giving several LAM-GTD algorithms.

Acknowledgement

I gratefully give thanks to Csaba Szepesvári, Rich Sutton, Joseph Modayil, Yasin Abbasi-Yadkori, István Szita, Amir massoud Farahmand, Mike Bowling, Lihong Li, and Eric Hansen for many helpful discussions. I also specially thank Michail Lagoudakis for sending me the bicycle simulator.

References

Bowling, Michael, Geramifard, Alborz, and Wingate, David. Sigma point policy iteration. In *AAMAS*, pp. 379–386, 2008.

Boyan, J. A. Technical update: Least-squares temporal difference learning. *Machine Learning*, 49:233–246, 2002.

Bradtke, S. and Barto, A. G. Linear least-squares algorithms for temporal difference learning. *Machine Learning*, 22:33–57, 1996.

Frank, J., Mannor, S., and Precup, D. Reinforcement learning in the presence of rare events. In *ICML*, 2008.

Kaebling, L.P., Littman, M.L., and Moore, A.W. Reinforcement learning: A survey. *JAIR*, 4:237–285, 1996.

Koller, D. and Parr, R. Policy iteration for factored MDPs. In *UAI*, 2000.

Lagoudakis, M. and Parr, R. Least-squares policy iteration. *JMLR*, 4:1107–1149, 2003.

Moore, A. W. and Atkeson, C. G. Prioritized sweeping: Reinforcement learning with less data and less time. *Machine Learning*, 13(1):103–130, 1993.

Parr, R., Li, L., Taylor, G., Painter-Wakefiled, C., and Littman, M. L. An analysis of linear models, linear value-function approximation, and feature selection for reinforcement learning. In *ICML*, 2008.

Precup, Doina, Sutton, Richard S., and Dasgupta, Sanjoy. Off-policy temporal-difference learning with function approximation. In *ICML*, 2001.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

Sutton, R. S., Szepesvári, Cs., Geramifard, A., and Bowling, M. Dyna-style planning with linear function approximation and prioritized sweeping. In *UAI*, 2008.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvari, Cs., and Wiewiora, E. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *ICML*, 2009.

Watkins, C. J. C. H. *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

Xu, X., He, H., and Hu, D. Efficient reinforcement learning using recursive least-squares methods. *JAIR*, 16:259–292, 2002.

Yao, H. Approximate policy iteration with linear action models. Technical Report TR 10-07, Department of Computing Science, University of Alberta, 2010.