Importance Sampling Estimates for Policies with Memory

Christian R. Shelton CSHELTON@AI.MIT.EDU
MIT, AI Lab 545 Technology Square, NE43-741, Cambridge, MA 02139 USA

Abstract

Importance sampling has recently become a popular method for computing off-policy Monte Carlo estimates of returns. It has been known that importance sampling ratios can be computed for POMDPs when the sampled and target policies are both reactive (memoryless). We extend that result to show how they can also be efficiently computed for policies with memory state (finite state controllers) without resorting to the standard trick of pretending the memory is part of the environment. This allows for very data-efficient algorithms. We demonstrate the results on simulated problems.

1. Introduction

In reinforcement learning, the goal is to maximize the expected return by changing the policy. An potentially important step in this process is policy evaluation: calculating the expected return given a policy. If the agent has been executing this policy for a series of trials, then a reasonable estimate of the expected return is the average return over the trails conducted. This is the standard Monte Carlo estimate.

However, if the agent has been executing a different policy than the one we would like to evaluate (*i.e.* we are doing off-policy evaluation) then we must be more clever in how we estimate the return. In particular, we would like to use trajectories collected under one policy (or a set of policies) to evaluate a new (target) policy. This allows for better reuse of data in POMDP RL tasks. Importance sampling has recently become a popular method for dealing this problem (Precup et al., 2000; Meuleau et al., 2001; Shelton, 2001).

1.1 Importance Sampling

Importance sampling is typically presented as a method for reducing the variance of the estimate of an expectation by carefully choosing a sampling distribution (Rubinstein, 1981). For example, the most direct method for evaluating $\int f(x)p(x)\,dx$ is to sample i.i.d. $x_i \sim p(x)$ and use $\frac{1}{n}\sum_i f(x_i)$ as the estimate. However, by choosing a different distribution $q(x)$ which has higher density in the places where $|f(x)|$ is larger, we can get a new estimate which is still unbiased and has lower variance. In particular, we now draw $x_i \sim q(x)$ and use $\frac{1}{n}\sum_i f(x_i)\frac{p(x_i)}{q(x_i)}$ as our estimate. This can be viewed as estimating the expectation of $f(x)\frac{p(x)}{q(x)}$ with respect to $q(x)$ which is like approximating $\int f(x)\frac{p(x)}{q(x)}q(x)\,dx$ with samples drawn from $q(x)$. If $q(x)$ is chosen properly, our new estimate has lower variance. There are more complicated methods, but they are not important for our discussion as they all rely on the same basic ratio calculations. We refer the reader to Hesterberg (1995) for some further discussion of different methods.

For off-policy Monte Carlo evaluation, we will be forced to "turn importance sampling on its head." Instead of choosing $q(x)$ to reduce variance, we will be forced to use $q(x)$ because of how our data was collected.

1.2 Importance Sampling for RL

To instantiate importance sampling in the reinforcement learning framework, we set a bit of notation. Let h represent a trial history[1] of four sequences: the sequence of world states (s_1 through s_T), the sequence of observations (x_1 through x_T), the sequence of actions (a_1 through a_T), and the sequence of rewards (r_1 through r_T). We have assumed a fixed-length trial of T time steps. Note that while we have included the state sequence as part of the history, it is not observable by the agent; it is included for theoretical purposes only and any calculation made using the history must not depend on the state sequence. Note that fixing a policy fixes a probability distribution over histories. The

[1]It might be better to refer to this as a trajectory since we will not limit h to represent only sequences that have been previously observed; it can also stand for sequences that might be observed. However, the symbol t is over used already.

function $R(h)$ (the return, or sum of rewards, of a history) is the equivalent to $f(x)$ in the previous section and $p(h|\pi)$ is the distribution from which samples are drawn if we are executing policy π.

In particular, the agent has been executing policy π' (our sampled trials are drawn from the distribution $p(h|\pi')$) and we wish to estimate the expectation of $R(h)$ with respect to $p(h|\pi)$. Therefore if we let $h^1, h^2, \ldots, h^n$ be the collected histories[2] under policy π', the standard importance sampling estimate is

$$E[R|\pi] \approx \frac{1}{n}\sum_{i=1}^{n} R(h^i)\frac{p(h^i|\pi)}{p(h^i|\pi')} \ .$$

This may look like a problem because h represents the entire history including the state sequence (which was unobserved). Computing $p(h^i|\pi)$ or $p(h^i|\pi')$ is impossible in most cases without knowledge of the underlying POMDP. However, in the next section we examine why this isn't a problem for reactive policies and then extend that result to policies with memory.

2. Computing Importance Sampling Ratios

While not all importance sampling estimators take exactly the form of equation 1.2, all require computing the ratio of the probability of a history under one policy against the probability of the same history under a different policy. Each of Precup et al. (2000); Meuleau et al. (2001); Shelton (2001) present a slightly different estimator but all depend on the same importance sampling ratios. It is these ratios (and not the individual probabilities) which are computable.

2.1 Memoryless Ratios

Let $\pi(x, a)$ be a stochastic policy (the probability of picking action a upon observing x). For the moment we will consider only reactive policies of this form but will extend this in the next section to policies with memory. The key observation is that we can calculate one factor in the probability of a history given a policy.

[2]We have used superscripts to denote trails and subscript to denote time within a trial.

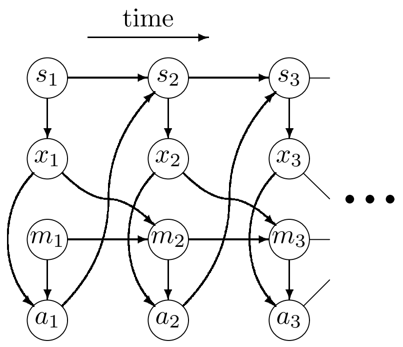

Figure 1. Dependency graph for agent-world interaction with memory model

In particular, that probability has the form

$$\begin{aligned} p(h|\pi) &= p(s_1)\prod_{t=1}^{T} p(x_t|s_t)\pi(x_t,a_t)p(s_{t+1}|s_t,a_t) \\ &= \left[p(s_1)\prod_{t=1}^{T} p(x_t|s_t)p(s_{t+1}|s_t,a_t)\right]\left[\prod_{t=1}^{T}\pi(x_t,a_t)\right] \\ &= W(h)A(h,\pi) \ . \end{aligned}$$

$A(h, \pi)$, the effect of the agent, is computable whereas $W(h)$, the effect of the world, is not because it depends on knowledge of the underlying state sequence. However, $W(h)$ does not depend on π. This implies that the ratios necessary for importance sampling are exactly the ratios that are computable without knowing the state sequence. In particular, if a history h were drawn according to the distribution induced by π and we would like an unbiased estimate of the return of π', then we can use $R(h)\frac{p(h|\pi')}{p(h|\pi)}$ and although neither the numerator nor the denominator of the importance sampling ratio can be computed, the $W(h)$ terms in each probability cancel leaving the ratio of $A(h, \pi')$ to $A(h, \pi)$ which can be calculated. A different statement of the same fact have been shown before in Meuleau et al. (2001).

2.2 Memory Ratios

We would now like to move beyond reactive policies. Consider adding memory in the style of a finite-state controller. At each time step, the agent reads the value of the memory along with the observation and makes a choice about which action to take and the new setting for the memory. The policy now expands to the form $\pi(x, m, a, m') = p(a, m'|x, m)$, the probability of picking action a and new memory state m' given observation x and old memory state m. Now let us factor this distribution, thereby limiting the class of policies realizable by a fixed memory size slightly but making the model simpler. In particular we consider an agent model where the agent's policy has two parts: $\pi_a(x, m, a)$ and $\pi_m(x, m, m')$. The former is the probability of choosing action a given that the observation is x and the internal memory is m. The latter is the probability of changing the internal memory to m' given the observation is x and the internal memory is m. Thus $p(a, m'|x, m) = \pi_a(x, m, a)\pi_m(x, m, m')$. By this factoring of the probability distribution of action-memory choices, we induce the dependency graph shown in figure 1.

If we let M be the sequence $\{m_1, m_2, \ldots, m_T\}$, $p(h|\pi)$ can be written as

$$\begin{aligned}
&\sum_M p(h, M|\pi) \\
&\quad= \sum_M p(s_1)p(m_1) \prod_{t=1}^{T} p(x_t|s_t)\pi_a(x_t, m_t, a_t) \\
&\qquad\qquad \pi_m(x_t, m_t, m_{t+1})p(s_{t+1}|s_t, a_t) \\
&\quad= \left[p(s_1) \prod_{t=1}^{T} p(x_t|s_t)p(s_{t+1}|s_t, a_t)\right] \\
&\qquad \left[\sum_M p(m_1) \prod_{t=1}^{T} \pi_a(x_t, m_t, a_t)\pi_m(x_t, m_t, m_{t+1})\right] \\
&\quad= W(h)A(h, \pi)\ ,
\end{aligned}$$

once again splitting the probability into two parts: one for the world dynamics and one for the agent dynamics. The $A(h, \pi)$ term now involves a sum over all possible memory sequences.

Computing this sum explicitly would take too long. However, $A(h, \pi)$ is exactly the probability of an input-output hidden Markov model (IOHMM), a slight variation of the general HMM. In particular, this new problem has the same structure as figure 1 except that the state sequence and all associated links are removed. Linear time algorithms are well known for computing the probability and its derivative for these models using dynamic programming. A good discussion of such algorithms for the HMM case can be found in Rabiner (1989). Bengio (1999) discusses extensions of HMMs including IOHMMs. For completeness, we will give a quick overview of the results needed for the importance sampling.

For a sequence $z_1, z_2, \ldots, z_n$, let $z_{i,j}$ represent the sequence $z_i, z_{i+1}, \ldots, z_j$ for simplicity of notation. We will define recurrence relations on two quantities. The first is $\alpha_i(m) = p(a_{1,i}, m_i = m|x_{1,T}, \pi_m, \pi_a)$. Its recurrence is

$$\alpha_{i+1}(m) = \pi_a(x_{i+1}, m, a_{i+1}) \sum_{m'} \alpha_i(m')\pi_m(x_i, m', m)$$

which says that the probability of having memory m after $i + 1$ time steps is equal to probability of producing the generated action with that memory bit multiplied by the sum of all the different ways the agent could have transitioned from the previous memory state (m') to the current memory state multiplied by the probability the agent was previous in memory state m'. We are counting up the number of ways of getting to memory m at time $i+1$ while still producing the observed action sequence. To do this we rely on the same set of probabilities for the previous time i.

The second recurrence relation is for $\beta_i(m) = p(a_{i+1,T}, m_i = m|x_{1,T}, \pi_m, \pi_a)$:

$$\beta_i(m) = \sum_{m'} \pi_m(x_i, m, m')\pi_a(x_{i+1}, m', a_{i+1})\beta_{i+1}(m')\ .$$

This has a similar interpretation, but working backwards through the data. The recurrence base cases are $\alpha_1(m) = p(m)\pi_a(x_1, m, a_1)$ and $\beta_T(m) = 1$. Because of the Markov property,

$$p(a_{1,T}, m_i = m|x_{1,T}, \pi_a, \pi_m) = \alpha_i(m)\beta_i(m)\ .$$

Thus the probability of the entire sequence can be found by computing $\sum_m \alpha_i(m)\beta_i(m)$ for any value i ($i = T$ is nice because then we don't have to compute the β sequence).

In some cases, we might also need the derivative of the probability with respect to the parameters. Without repeating the derivation, the result is

$$\begin{aligned}
\frac{\partial A(h, \pi)}{\pi_a(x, m, a)} &= \sum_{i|a_i=a, x_i=x} \frac{\alpha_i(m)\beta_i(m)}{\pi_a(x, m, a)} \\
\frac{\partial A(h, \pi)}{\pi_m(x, m, m')} &= \sum_{i|x_i=x} \alpha_i(m)\beta_{i+1}(m')\pi_a(x_{i+1}, m', a_{i+1})\ .
\end{aligned}$$

We can now use the same estimators and allow for policies with memory. In particular, the estimator has explicit knowledge of the working of the memory. This is in direct contrast to the method of adding the memory to the action and observation spaces and running a standard reinforcement learning algorithm (see Peshkin et al. (1999) as an example of this approach) where the agent must learn the dynamics of its own memory. With this explicit memory model, the learning algorithm understands that the goal is to produce the correct action sequence and uses the memory state to do so by coordinating the actions in different time steps.

3. Experiments

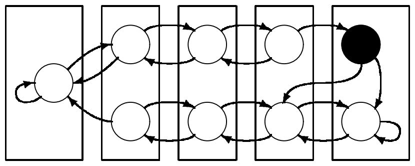

Figure 2. Diagram of the "load-unload" world. This world has nine states. The horizontal axis corresponds to the positioning of a cart. The vertical axis indicates whether the cart is loaded. The agent only observes the position of the cart (five observations denoted by boxes). The cart is loaded when it reaches the left-most state and if it reaches the right-most position while loaded, it is unloaded and the agent receives a single unit of reward. The agent has two actions at each point: move left or move right. Moving left or right off the end leaves the cart unmoved. Each trial begins in the left-most state and lasts 100 time steps.

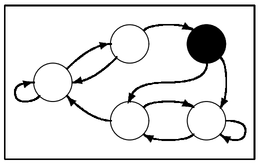

Figure 3. Diagram of the "blind load-unload" world. This world has five states. The horizontal axis corresponds to the positioning of the cart. The vertical axis indicates whether the cart is loaded. The agent is completely blind: no matter what the state of the world, it observes the same thing. Other than these differences, the problem is the same as the one in figure 2. Each trial begins in the left-most state and lasts 100 time steps.

The load-unload problem of figure 2 is a traditional POMDP problem. A cart sits on a line with five discrete positions. When the cart makes it to the left-most state, the cart is filled. When the cart arrives in the right-most state with a full cart, the cart is emptied and the agent receives one unit of reward. The agent can observe the position of the cart, but not the contents (*i.e.* it does not know whether the cart is full or empty). To achieve reasonable performance, the actions must depend on the history. We give the agent one memory bit (two memory states); this results in twenty independent policy parameters. We use the importance sampling greedy-search algorithm of Shelton (2001). Figure 4 compares using the importance sampling estimate in two different ways. On the left is the result when the memory is made to be part of the environment (*i.e.* the action space is now all possible combinations of moving and setting the memory bit and the observation space is all possible combinations of position and memory state). This means that there are 10 observations and 4 actions. On the right is the result when the memory is internal to the algorithm and incorporated as shown in the previous section (same number of observations and actions). We can clearly see that the number of trials required to converge to the optimal solution is far fewer in the case of internal memory.

To consider a more drastic example, we constructed a blind load-unload problem, as shown in figure 3. In this case, we give the agent two memory bits (four memory states). Figure 5 compares the result of internal and external memory for this example. Again, we see a clear gain by explicitly modeling the memory dynamics.

4. Conclusion

Although these results are with one particular importance sampling algorithm, they should work equally well for any method that depends on the importance sampling ratios. With this method, it is possible to effectively learn finite-state controllers. Furthermore, the technique is not limited to modeling memory bits. It can be extended to any part of the agent or environment for which the dynamics are known and the action selection is factorable.

References

Bengio, Y. (1999). Markovian models for sequential data. *Neural Computing Surveys*, *2*, 129–162.

Hesterberg, T. (1995). Weighted average importance sampling and defensive mixture distributions. *Technometrics*, *37*, 185–194.

Meuleau, N., Peshkin, L., & Kim, K.-E. (2001). *Exploration in gradient-based reinforcement learning* (Technical Report AI-MEMO 2001-003). MIT, AI Lab.

Peshkin, L., Meuleau, N., & Kaelbling, L. P. (1999). Learning policies with external memory. *Proceedings of the Sixteenth International Conference on Machine Learning.*

Precup, D., Sutton, R. S., & Singh, S. (2000). Eligi-

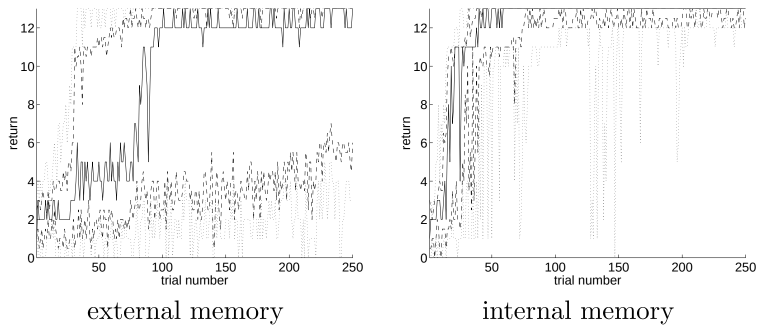

Figure 4. A comparison of placing the memory bits externally as part of the environment to modeling them explicitly as an internal part of the agent for the problem in figure 2. Both graphs were generated from 10 runs of the algorithm. The plotted lines are the (from top to bottom), maximum, third quartile, median, first quartile, and minimum return for each time step across the 10 runs.

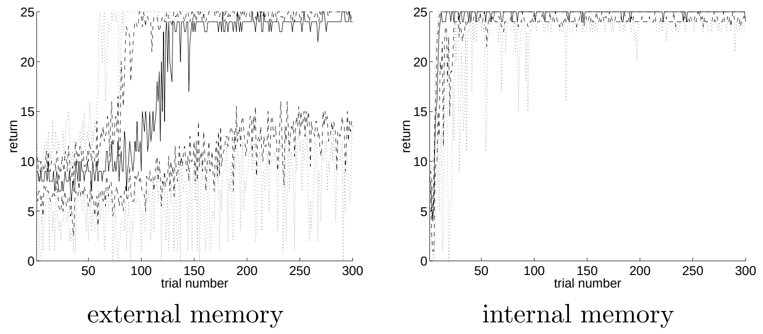

Figure 5. A comparison of placing the memory bits externally as part of the environment to modeling them explicitly as an internal part of the agent for the problem in figure 3. Both graphs were generated from 10 runs of the algorithm. The plotted lines are the (from top to bottom), maximum, third quartile, median, first quartile, and minimum return for each time step across the 10 runs.

bility traces for off-polcy policy evaluation. *Proceedings of the Seventeenth International Conference on Machine Learning*.

Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, *77*, 257–286.

Rubinstein, R. Y. (1981). *Simulation and the monte carlo method*. John Wiley & Sons.

Shelton, C. R. (2001). Policy improvement for POMDPs using normalized importance sampling. *Proceedings of the Seventeenth International Conference on Uncertainty in Artificial Intelligence*. to appear.