ANALYSIS OF VALUE-BASED ALGORITHMS IN AVERAGE REWARD SETTING

Varun Bhatt
Department of Computing Science
University of Alberta, Canada
vbhatt@ualberta.ca

ABSTRACT

Average reward is the natural objective to optimize in continuing environments. Contemporary reinforcement learning literature mostly focus on optimizing the discounted return using a variety of algorithms. This project studies the average reward equivalent of value-based methods N-step TD, TD (λ), N-step SARSA, and SARSA (λ). On-policy prediction in grid world and random walk environments were used to test if multi-step methods perform better than one-step methods. It was found that initial estimate of average reward affects the performance of multi-step methods, adding another factor to consider when using them. Off-policy prediction in grid world and off-policy control in mountain car environment were used to test the usefulness of adding control variates to off-policy update. Control variates gave significantly better results across a large range of hyperparameters.

1 Average Reward Setting

Environments in reinforcement learning are divided into two categories - episodic and continuing. Continuing environments run forever and the agent needs to maximize the reward that it receives. Since, time is unbounded, sum of rewards or total return is unbounded. A popular way to overcome this is to multiply rewards at later time steps by a discount factor (less than 1) to get discounted return, which is bounded. This has a possibly unwanted consequence of giving higher weightage to immediate rewards compared to later rewards. Average reward setting overcomes this issue by considering differential return, where each reward is subtracted by the average reward. Mathematically, average reward is defined as (Sutton and Barto, 2018, pg. 249)

$$r(\pi) \doteq \lim_{h \to \infty} \frac{1}{h} \sum_{t=1}^{h} \mathbb{E}\left[R_t | S_0, A_{0:t-1} \sim \pi\right] \tag{1}$$

where R_t is the reward at time step t, S_0 is the initial state, A_t is the action taken at time t and π is the policy being followed.

Differential return at time t is defined as

$$G_t \doteq R_{t+1} - r(\pi) + R_{t+2} - r(\pi) + R_{t+3} - r(\pi) + \cdots \tag{2}$$

State and action values (called as differential state and action values) are defined similar to how they are done in discounted return setting, but using differential return instead of discounted return.

An algorithm that finds state or action value needs access to $r(\pi)$, which is not given in advance. Hence, the algorithm needs to estimate $r(\pi)$ as $\bar{R}$ and update it from experience. $\bar{R}$ is updated using TD error that is used for value updates, but with a different step size. The exact algorithms are given in later sections, but the basic idea is to take an algorithm designed for discounted return setting like TD(λ), replace all rewards by differential rewards ($R - \bar{R}$) and remove the discount factor. While this may give an algorithm to predict or maximize differential return, the properties of these algorithms may not reflect those found in discounted return setting.

Previous work on average reward mostly focus on control problems using one-step methods. One of the first works in average reward setting, R-Learning (Schwartz, 1993), is the average reward equivalent of Q-Learning. Tsitsiklis and

Code for the project is available at https://github.com/virtualgod/average_reward_cv

Van Roy (1999) showed that TD (λ) can be modified such that it converges in average reward setting, but didn't have any experimental analysis of the method. Chapter 10 of the textbook by Sutton and Barto (2018) contains the algorithm for one-step and N-step differential SARSA, but no experiments are performed with differential N-step SARSA. The algorithms provided in this project are inspired by one-step and N-step differential SARSA provided in that textbook.

This project aims to study multi-step methods and off-policy learning in average reward setting in more detail by asking three major questions. Is an intermediate n or λ better than one-step algorithms in average reward setting? Is it better to use one-step TD error or multi-step TD error in average reward update? Does adding control variates help in off-policy learning? Sections 2 to 5 tackle the first two questions, while sections 6 to 8 deal with the third one.

2 On-Policy Prediction Using Differential N-step TD and TD(λ)

On-policy state value prediction problem requires finding the differential state values for a given policy. In this project, it is assumed that function approximation is used for estimating the values, i.e, $v_\pi(s) \approx \hat{v}(s, \mathbf{w})$, where $\mathbf{w}$ are weights used in estimation. Temporal difference (TD) learning is a common algorithm to learn state values in discounted reward setting. State values are updated using TD errors, which, in average reward setting is given by

$$\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{3}$$

Weights of value estimate and average reward estimate are updated using TD error as

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{4}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta \delta_t \tag{5}$$

TD error is used instead of simple averaging for $\bar{R}$ update since averaging leads to oscillating around the expected value unless step size is annealed. These equations together give the average reward equivalent of TD(0), called differential TD(0). The complete algorithm is given in Algorithm 1.

Algorithm 1: Differential TD(0)

Input: Differentiable function for value estimation:$\hat{v}(s, \mathbf{w})$; Policy π to evaluate; Step sizes α, β
Initialize $\mathbf{w} \in \mathbb{R}^d$, $\bar{R} \in \mathbb{R}$ arbitrarily
Set S to initial state
Loop forever:
- Take action $A \sim \pi$ and observe R, S'
- $\delta \leftarrow R - \bar{R} + \hat{v}(S', \mathbf{w}) - \hat{v}(S, \mathbf{w})$
- $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla \hat{v}(S, \mathbf{w})$
- $\bar{R} \leftarrow \bar{R} + \beta \delta$
- $S \leftarrow S'$

2.1 Differential N-step TD

N-step methods provide a mechanism to control bootstrapping by using N-step return instead of one-step return. N-step TD errors are given by

$$\delta_t^{(n)} \doteq \sum_{i=t+1}^{t+n} \left(R_i - \bar{R}\right) + \hat{v}(S_{t+n}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t) \tag{6}$$

Differential N-step TD algorithm is similar to its one-step counterpart except that N-step TD error is used in updates. In this project, a two variants of differential N-step TD is considered, one where N-step TD error is used in $\bar{R}$ update, and other where one-step TD error is used in $\bar{R}$ update, to answer the question about the use of multi-step TD error in $\bar{R}$ update. To distinguish between the two variants, the former is called "differential N-step TD (full $\bar{R}$)", while the latter is simply called "differential N-step TD".

2.2 Differential TD(λ)

TD(λ) uses $\lambda \in [0, 1]$ and the corresponding λ-return to control bootstrapping. Computationally, they are implemented using eligibility traces (Chapter 12, Sutton and Barto, 2018). In the average reward setting, eligibility trace vector is updated as

$$\mathbf{z}_t \doteq \lambda \mathbf{z}_{t-1} + \nabla \hat{v}(S_t, \mathbf{w}_t) \tag{7}$$

The updates for value weights and average reward are given by

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha\delta_t\mathbf{z}_t \tag{8}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta\delta_t \tag{9}$$

An additional variant, called "differential TD(λ) (full $\bar{R}$)" is considered, where an additional eligibility trace is maintained for use in $\bar{R}$ update. $\bar{R}$ update is modified as $\bar{R}_{t+1} \doteq \bar{R}_t + \beta\delta_t z_{\bar{R},t}$. That eligibility trace is updated as $z_{\bar{R},t} \doteq \lambda z_{\bar{R},t-1} + 1$, which is equivalent to using λ-return for $\bar{R}$ update too.

3 Environments

A deterministic grid world and random walk are used for prediction problems while mountain car environment is used for control.

3.1 Random Walk

Random walk is a Markov reward process (MRP) with transitions and rewards as shown in Figure 1. A 19 state random walk is used for experiments. The agent starts in S_9, and in each state, the agent is equally likely to transition into one of the two neighbouring states, getting a reward of 0. A transition is present from S_0 and S_{18} back to S_9, giving a reward of -1 or +1 respectively when taken.

3.2 Grid World

The second environment for prediction problems is a deterministic 5x5 grid world environment as shown in Figure 2. The agent starts in the center (denoted by a circle) and can transition to one of the 4 adjacent states by taking one of the 4 actions, with no state change if transitioning into a wall. Each transition gets a reward of 0 except the ones into the states marked in black. Transitioning into either of the states marked in black takes the agent back to the center, giving a +1 reward.

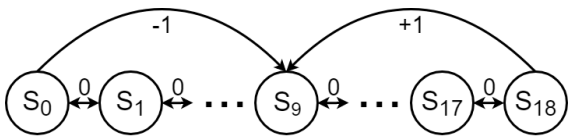


Figure 1: Random walk environment. Arrows denote transitions and numbers on top denote corresponding rewards.

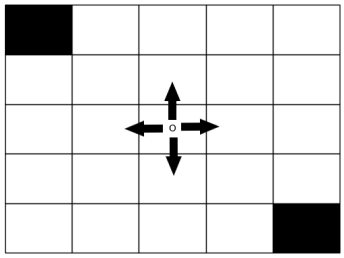


Figure 2: Grid world environment. Reward is 0 at every step expect when transitioning into black cells, which give +1 reward and put the agent back to center.

3.3 Mountain Car

A slightly modified version of mountain car environment (Moore, 1990) is used for control. Observations, actions and dynamics remain the same as before. Reward is 0 at every step. Transitioning into the rightmost position gives a reward of +1 to the agent and puts it back into a random position between $[-0.6, -0.4]$ with zero velocity (the initial state). Note that this makes the environment continuing instead of episodic, with average reward being the reciprocal of average time taken to reach the rightmost position.

4 Pros and Cons of Multi-step Algorithms

In discounted return setting, using some $n > 1$ or $\lambda \in (0, 1)$ is better than either one-step or Monte Carlo methods. The following set of experiments test whether the same holds even in average reward setting.

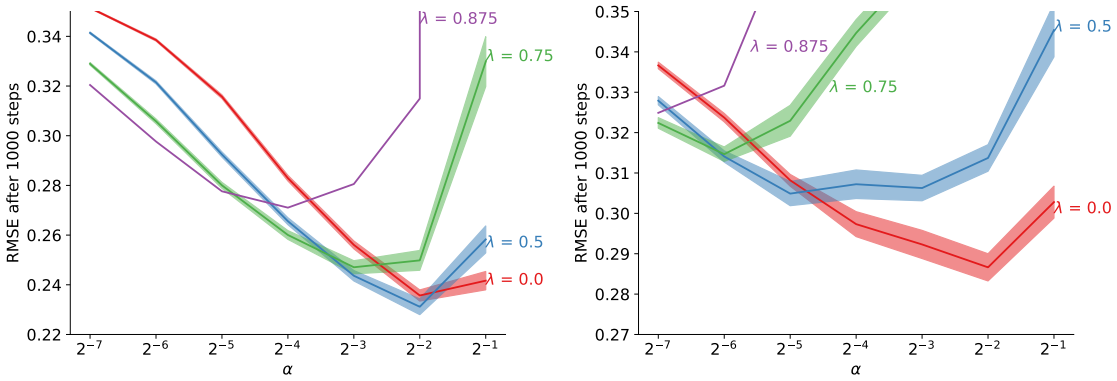
4.1 Experimental Setup

State value prediction in random walk and grid world environments described in Section 3 are used to test the effectiveness of multi-step algorithms. For random walk, objective was to predict the state values of the resulting MRP. For grid world, objective was to find the state values corresponding to a biased policy which goes up with probability 0.5 and takes an equiprobable random action otherwise. True values were found by solving the set of Bellman equations. Since the set of Bellman equations are under-determined, an additional constraint was added to set the value of initial state to zero. The values learnt by the algorithms were subtracted by the learnt value of initial state for consistency in comparison. Root mean square error (RMSE) between the true and predicted state values after 1000 time steps was used as comparison metric. A longer time horizon would give lower RMSE but would obscure any differences in the speed of learning.

To compare one-step and multi-step methods, differential N-step TD with $n \in \{1, 2, 4, 8\}$ and differential TD(λ) with $\lambda \in \{0, 0.5, 0.75, 0.875\}$ were used. Average reward was updated using one-step TD error in all cases. Since initial estimate of $\bar{R}$ affected the performance of algorithms, it was varied along with the step sizes. Initial weights were set to $\mathbf{0}$. For solving random walk problem, initial estimate of $\bar{R}$ was chosen from $\{0, 0.01, 0.1, 1, 10\}$, step size for weight update $\alpha \in \{2^0, 2^{-1}, \ldots, 2^{-5}\}$, step size for average reward update $\beta \in \{2^{-2}, 2^{-3}, \ldots, 2^{-9}\}$. For solving grid world problem, initial $\bar{R} \in \{0.01, 0.1, 1\}$, $\alpha \in \{2^{-1}, 2^{-2}, \ldots, 2^{-7}\}$, $\beta \in \{2^{-2}, 2^{-3}, \ldots, 2^{-9}\}$. 100 runs were performed with different random seeds for each combination of parameters and the resulting mean RMSE and the standard error were reported.

4.2 Results

Results are presented here for differential TD(λ). Results for differential N-step TD were similar and are omitted in the interest of space. Figures 3a and 3b show the RMSE vs α graphs on grid world problem. For each α, RMSE corresponding to the best β was chosen. The two graphs differ in the initial estimate of $\bar{R}$, which was set to 0.1 for Figure 3a and to 1 for Figure 3b. It can be observed that an intermediate λ ($= 0.5$) was better for a large range of parameters when initial estimate of $\bar{R}$ was 0.1, while $\lambda = 0$ (one-step) was better when initial estimate of $\bar{R}$ was 1.



(a) Initial $\bar{R} = 0.1$ (b) Initial $\bar{R} = 1$

Figure 3: RMSE (after 1000 steps) vs α on grid world problem. Good β was chosen for each α. Shaded region is one standard error over 100 runs. $\lambda = 0.5$ was better with initial $\bar{R} = 0.1$, whereas $\lambda = 0$ was better with initial $\bar{R} = 1$.

Figure 4 shows RMSE vs α graph on random walk problem. For each α, RMSE corresponding to the best β and $\bar{R}$ initialization was chosen. There was little difference in results for various initializations of $\bar{R}$ and hence, are not shown separately. In this problem, multi-step methods were observed to be better ($\lambda = 0.5$ or $\lambda = 0.75$ was better depending on α).

4.3 Discussions

The results for random walk problem show that an intermediate λ is better than $\lambda = 0$ or 1. But the results for grid world problem are mixed. When initial estimate of $\bar{R}$ was closer to the true average reward, an intermediate λ performed better. But when the initial estimate was inaccurate, one-step method was the best.

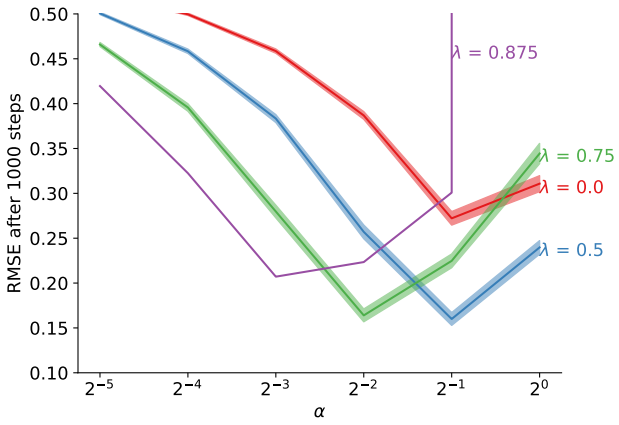


Figure 4: RMSE (after 1000 steps) vs α on random walk problem. Good β and initial $\bar{R}$ were chosen for each α. Shaded region is one standard error over 100 runs. $\lambda = 0.5$ or $\lambda = 0.75$ was better depending on α.

A possible reason for this behaviour can be found by looking at the N-step TD error in equation 6. First, differential N-step methods still do full bootstrapping, with only the bootstrapping time varying with n (unlike in discounted return setting, where the bootstrapped value is multiplied by γ^n and there are terms near the end of episode with no bootstrapping). Second, there is a term $n\bar{R}$ which depends on the estimate of $\bar{R}$. Higher n implies higher reliance on a good estimate of $\bar{R}$. This explains the results seen in grid world environment where an inaccurate initial estimate of $\bar{R}$ led to one-step methods performing better. Finally, variance of the update increases with higher n (which is also seen in discounted return setting). The same reasoning applies to λ methods too, but it is clearer to understand in case of N-step methods.

The results of random walk problem didn't change much even when initial estimate of $\bar{R}$ was changed since the problem has extremely sparse feedback and the disadvantages of inaccurate estimate of $\bar{R}$ was overshadowed by the advantages of being able to propagate values quicker by using a higher n or $\lambda > 0$.

Hence, multi-step methods are still useful in average reward setting but the three points discussed above must be considered when choosing multi-step methods.

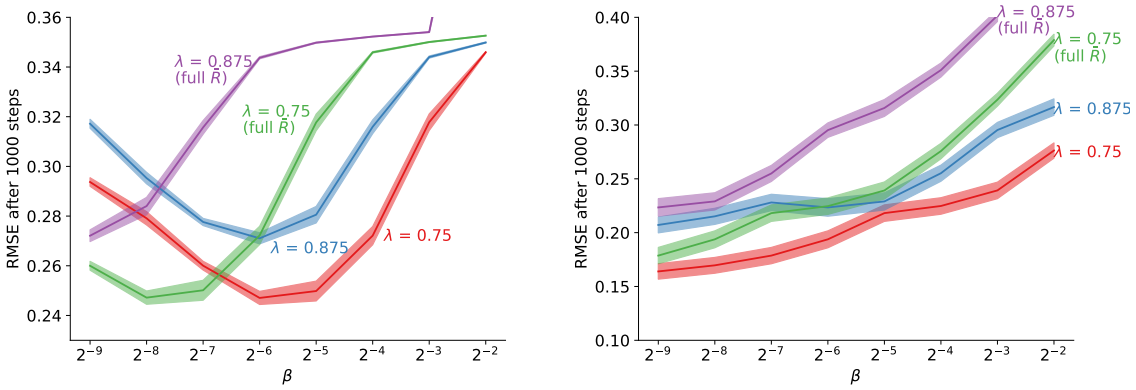
There are a few additional hypotheses that could be tested in future. One, methods such as unbiased step size for updating $\bar{R}$ can give a good estimate of $\bar{R}$ from the beginning itself. Experiments can be performed in future to see if the allows multi-step methods to perform better. Two, it was found during the experiments that setting the initial estimate of $\bar{R}$ exactly equal to the true average reward didn't necessarily give the best performance. Further experiments are required to test if this hypothesis is true.

5 One-step vs Multi-step TD error for Average Reward Update

The next set of experiments test if there is any merit in using multi-step TD error for average reward update. The previous set of experiments described in Section 4.1 were repeated twice, once where algorithms used one-step TD error to update $\bar{R}$ and other where they used multi-step TD error to update $\bar{R}$ (full $\bar{R}$ variants of the algorithms).

5.1 Results

Figures 5a and 5b show RMSE vs β on grid world and random walk problems respectively. For each β, RMSE corresponding to the best α and $\bar{R}$ initialization was chosen. The curves marked as "(full $\bar{R}$)" use multi-step TD error for both weight and average reward update, while the unmarked ones use one-step TD error for average reward update and multi-step TD error only for weight update. Only the curves corresponding to $\lambda = 0.75$ or $\lambda = 0.875$ are shown since the two variants are exactly same for $\lambda = 0$ and the difference is not significant for $\lambda = 0.5$. It can be observed that using one-step TD error for average reward update is better across a large range of parameters in both cases.



(a) Grid world

(b) Random walk

Figure 5: RMSE vs β graphs when average reward is updated using multi-step TD error (full $\bar{R}$) or one-step TD error (unmarked). "(full $\bar{R}$)" is seen to be worse for a large range of parameters in both problems.

5.2 Discussions

The reason to use TD error in average reward update instead of just the rewards is to ensure that the estimate stops changing when TD error is zero as opposed to oscillating around the true value. The advantage of multi-step method in helping propagate the value quicker becomes invalid for the purpose of updating $\bar{R}$. Also, as seen in Section 4.3, using multi-step TD error can accumulate errors in $\bar{R}$ estimate and has higher variance.

Combination of the above factors makes the performance worse when using multi-step TD error for $\bar{R}$ update. Hence, for the remainder of the project, "(full $\bar{R}$)" variants of the algorithms are not considered.

6 Off-Policy Prediction

In off-policy prediction, state values corresponding to a target policy should be found while following a behaviour policy that is possibly different from target policy. To correct for the differences in probabilities of action selection, the observed return must be scaled appropriately before using to update the values. Per-decision importance sampling (Sutton and Barto, 2018, pg. 114) corrects the return by multiplying it with the ratio of probabilities of the action taken in target and behaviour policy. Mathematically, for target policy π and behaviour policy b,

$$\rho_t = \frac{\pi\left(S_t, A_t\right)}{b\left(S_t, A_t\right)} \tag{10}$$

$$G_t = \rho_t \left(R_{t+1} - \bar{R}_t + G_{t+1}\right) \tag{11}$$

ρ is called the importance sampling ratio. The updates for weights and average reward remain the same once the return is corrected by importance sampling. TD error is defined as $\delta_t \doteq G_t - \hat{v}\left(S_t, \mathbf{w}_t\right)$. Differential TD(0) uses one-step TD error in which G_{t+1} is replaced by $\hat{v}\left(S_{t+1}, \mathbf{w}_t\right)$. For differential N-step TD, the recursion is unrolled n times and G_{t+n} is replaced by $\hat{v}\left(S_{t+n}, \mathbf{w}_t\right)$.

For differential TD(λ), the eligibility trace vector for values is updated as $\mathbf{z}_t \doteq \lambda\rho_{t-1}\mathbf{z}_{t-1} + \nabla\hat{v}\left(S_t, \mathbf{w}_t\right)$ to accommodate importance sampling. See Appendix A for the derivation of eligibility traces.

6.1 Control Variates

Suppose the objective is to estimate $\mathbb{E}[X]$ for a variable X. $X^* = X + c(Y - \mathbb{E}[Y])$ is an unbiased estimator of $\mathbb{E}[X]$, where Y is a variable with known expected value (called control variate), and c is a constant. By choosing c appropriately, variance of the estimator can be reduced (Ross, 2013).

For prediction problems, X is the corrected return $\rho_t G_{t+1}$. By choosing $Y = \rho_t\hat{v}\left(S_t, \mathbf{w}_t\right)$, optimal c is found to be equal to -1 (De Asis and Sutton, 2018). Thus, with control variate,

$$G_t = \rho_t \left(R_{t+1} - \bar{R}_t + G_{t+1}\right) + \left(1 - \rho_t\right)\hat{v}\left(S_t, \mathbf{w}_t\right) \tag{12}$$

Control variates can be used in differential TD(0) and differential N-step TD by simply using this G instead of the one defined previously.

For differential TD(λ) with control variates, one-step TD error is redefined as $\delta_t \doteq R_{t+1} - \bar{R}_t + \hat{v}(S_{t+1}, \mathbf{w}_t) - \hat{v}(S_t, \mathbf{w}_t)$ for mathematical convenience. Eligibility trace vector is updated as $\mathbf{z}_t \doteq \rho_t (\lambda \mathbf{z}_{t-1} + \nabla \hat{v}(S_t, \mathbf{w}_t))$. To incorporate control variates in average reward update, it is updated as $\bar{R}_{t+1} \doteq \bar{R}_t + \beta \rho_t \delta_t$.

Variants of TD using control variates only for value weights update are henceforth suffixed "(cv)" and those that use control variates for both weights and average reward are suffixed "(cv $\bar{R}$)".

7 Off-Policy Control Using Differential N-step SARSA and SARSA(λ)

Value-based algorithms for control have two main steps, estimate the action values for the current policy (policy estimation) and update the policy to be greedy (or ϵ-greedy) with respect to the action values (policy improvement). In off-policy case, the behaviour policy can be different from target policy. Generally, behaviour policy balances exploration and exploitation while the target policy is greedy with respect to action values.

Since on-policy control is a subset of off-policy control, it is skipped here for brevity. See the textbook by Sutton and Barto (2018, pg. 249) for the algorithms. Return after per-decision importance sampling is given by

$$G_t = R_{t+1} - \bar{R}_t + \rho_{t+1} G_{t+1} \tag{13}$$

TD error is defined as $\delta_t \doteq G_t - \hat{q}(S_t, A_t, \mathbf{w}_t)$. Differential SARSA(0), which is the average reward equivalent of SARSA(0), uses one-step TD error, where G_{t+1} is replaced by $\hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$. Differential N-step SARSA unrolls the recursion for n steps and replaces G_{t+n} by $\hat{q}(S_{t+n}, A_{t+n}, \mathbf{w}_t)$.

Weights of action value estimate and average reward estimate are updated using TD error as

$$\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \nabla \hat{q}(S_t, A_t \mathbf{w}_t) \tag{14}$$

$$\bar{R}_{t+1} \doteq \bar{R}_t + \beta \delta_t \tag{15}$$

The complete algorithm for SARSA(0) is given in Algorithm 2.

Algorithm 2: Differential SARSA(0)

Input: Differentiable function for action value estimation:$\hat{q}(s, a, \mathbf{w})$; Behaviour policy b and target policy π; Step sizes α, β

Initialize $\mathbf{w} \in \mathbb{R}^d$, $\bar{R} \in \mathbb{R}$ arbitrarily

Set S to initial state and choose action $A \sim b$

Loop forever:

- Take action A and observe R, S'
- Choose next action $A' \sim b$
- $\rho \leftarrow \frac{\pi(S', A')}{b(S', A')}$
- $\delta \leftarrow R - \bar{R} + \rho \hat{q}(S', A', \mathbf{w}) - \hat{q}(S, A, \mathbf{w})$
- $\mathbf{w} \leftarrow \mathbf{w} + \alpha \delta \nabla \hat{q}(S, A, \mathbf{w})$
- $\bar{R} \leftarrow \bar{R} + \beta \delta$
- $S \leftarrow S'$
- $A \leftarrow A'$

Differential SARSA(λ) maintains an eligibility trace vector to update the weights. Eligibility trace vector is updated as $\mathbf{z}_t \doteq \lambda \rho_t \mathbf{z}_{t-1} + \nabla \hat{q}(S_t, A_t, \mathbf{w}_t)$. Weights are updated as $\mathbf{w}_{t+1} \doteq \mathbf{w}_t + \alpha \delta_t \mathbf{z}_t$, where δ_t is the one-step TD error. See Appendix B for the derivation.

7.1 Incorporating Control Variates

Control variates, introduced in Section 6.1, can be applied to action value updates too. In action value case, X is $\rho_{t+1} G_{t+1}$. Using $Y = \rho_{t+1} \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t)$, which has the expected value $\mathbb{E}_\pi[\hat{q}(S_{t+1}, \cdot, \mathbf{w}_t)]$, optimal c is found to be -1 (De Asis and Sutton, 2018). Applying the control variate, the return becomes

$$G_t = R_{t+1} - \bar{R}_t + \rho_{t+1} G_{t+1} + \mathbb{E}_\pi[\hat{q}(S_{t+1}, \cdot, \mathbf{w}_t)] - \rho_{t+1} \hat{q}(S_{t+1}, A_{t+1}, \mathbf{w}_t) \tag{16}$$

TD error is still defined as $\delta_t \doteq G_t - \hat{q}(S_t, A_t, \mathbf{w}_t)$. To add this control variate in differential SARSA(0) and differential N-step SARSA updates, the above return is used after replacing G by $\hat{q}$ at appropriate time step. Differential SARSA(λ) uses the one-step TD error defined here and other parts are exactly same as off-policy case.

8 Advantages of Control Variates

Control variates have been shown to provide better performance in discounted return setting (De Asis and Sutton, 2018). The following experiments test if the results are similar in average reward setting.

8.1 Experimental Setup

Grid world and mountain car environment, described in Section 3, were used to test the effectiveness of control variates in off-policy prediction and control problems respectively. For prediction in grid world, behaviour policy was equiprobable random, while the target policy was biased, going up with probability 0.5 and taking an equiprobable random action otherwise. For control in mountain car, behaviour policy was ϵ-greedy with $\epsilon = 0.3$ and target policy was greedy with respect to action values. For prediction, comparison metric was RMSE between true and predicted value after 1000 steps, as explained in Section 4.1. For control, algorithms were trained for 20000 steps (around 30 episodes if the environment was episodic) and the reciprocal of average reward was used as the comparison metric. Reciprocal of average reward is equivalent to average time taken per episode, which is generally used in episodic case.

Since mountain car has continuous observations, linear function approximation with tile coded features (Sutton and Barto, 2018, pg. 217) was used. Tile coder was hash based[1] with 16 tilings and each tile covering $1/8$ of the feature space width along each dimension.

Differential TD(λ), differential TD(λ) (cv), differential TD(λ) (cv $\bar{R}$) were compared with $\lambda \in \{0, 0.5, 0.75, 0.875\}$. Like in previous experiments, initial estimate of $\bar{R}$ and step sizes were varied, while the initial weights were set to $\mathbf{0}$. For solving prediction problem, initial $\bar{R} \in \{0.01, 0.1, 1\}$, $\alpha \in \{2^{-1}, 2^{-2}, \dots, 2^{-7}\}$, $\beta \in \{2^{-2}, 2^{-3}, \dots, 2^{-9}\}$. Each combination of parameters is run 1000 times with different random seeds and mean RMSE and standard error are reported.

For solving control problem, initial $\bar{R} \in \{0.01, 0.1, 1\}$, $\alpha \in \{2^{-1}, 2^{-2}, \dots, 2^{-7}\}$, $\beta \in \{2^{-1}, 2^{-2}, \dots, 2^{-9}\}$. Each combination of parameters is run 30 times with different random seeds and mean of reciprocal of average reward and standard error are reported.

8.2 Results

Figure 6a shows RMSE vs α graph on grid world prediction problem. For each α, RMSE corresponding to the best β and $\bar{R}$ initialization was chosen. $\lambda = 0.75$ and $\lambda = 0.875$ are not shown since their errors were much higher. The curves marked "(cv)" use control variates introduced in section 6.1 while unmarked ones don't use them. Using control variates is seen to be clearly better. Figure 6b shows the same graph but comparing the algorithms using control variates for both weight and average reward update (marked cv $\bar{R}$) with those using control variates only for weight update (marked cv). Using control variates for average reward update is not statistically significantly better for most parameters, but is slightly better when there is a difference.

Figures 7a and 7b show the reciprocal of average reward vs α on mountain car control problem. For each α, values of β and $\bar{R}$ initialization which gave the highest average reward was chosen. $\lambda = 0.75$ and $\lambda = 0.875$ are not shown since their curves were outside the graph limits for most parameters. The curves marked "(cv)" and "(cv $\bar{R}$)" have same meaning as in previous paragraph. Even for this problem, using control variates for weight update is better than not using them, but adding it to average reward update doesn't give statistically significant benefit.

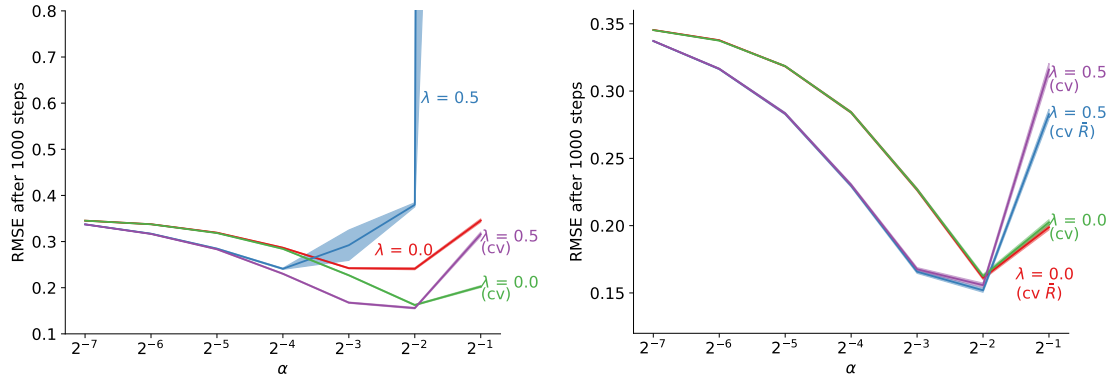
8.3 Discussions

Control variates are consistently better across a large range of parameters both for prediction and control problem. The result is not surprising given that their effectiveness in discounted return setting is due to variance reduction, which is also happening here.

The lack of difference between using and not using control variates for $\bar{R}$ updates is a bit surprising. It may be possible that the problems chosen in the project are not the right ones to highlight the differences, or it may actually not make a significant difference. This is a hypothesis that can be tested in future.

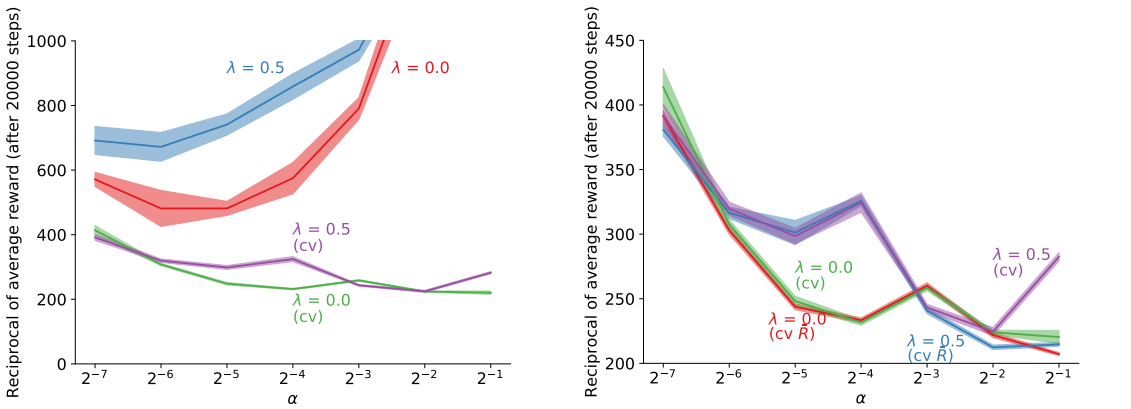
During experiments, it was found that no learning took place in mountain car problem when initial estimate of $\bar{R}$ was 0. Since weights were initialized to $\mathbf{0}$, all values were 0. Each step gave a reward of 0, which meant that the updates were equal to 0 till the agent reached the rightmost position for the first time. With only ϵ-greedy aiding exploration and no optimistic initialization, the agent was never able to reach the rightmost position within 20000 time steps. Setting

[1] Using Tile coding software by Richard S. Sutton, available at `http://incompleteideas.net/tiles/tiles3.html`



(a) Control variates (cv) vs simple off-policy (unmarked) (b) With (cv $\bar{R}$) and without control variates for $\bar{R}$ update

Figure 6: RMSE vs α graphs for grid world prediction problem. Using control variates for weight updates gives a significant improvement in performance. Using control variates for average reward updates too improves the performance slightly, but is not significant.



(a) Control variates (cv) vs simple off-policy (unmarked) (b) With (cv $\bar{R}$) and without control variates for $\bar{R}$ update

Figure 7: Reciprocal of average reward (after 20000 steps) vs α graphs for mountain car control problem. Trends are similar to prediction problem.

$\bar{R} > 0$ made the initial updates negative, leading to better exploration and hence, giving the results shown here. The exact trade-off of using $\bar{R}$ for exploration is not clear at this moment, and might be an idea to study in future.

9 Conclusion

This project provides modification to value-based algorithms to make them work in an average reward setting. The results don't exactly mirror discounted return setting. There are three major takeaways from this project. One, there are additional trade-offs when using mulit-step methods in average reward setting due to always using full bootstrapping and the presence of $\bar{R}$ estimate in updates. Two, updating $\bar{R}$ using one-step TD error is better than using multi-step TD error. Three, control variates are useful in off-policy prediction and control problems, and it might also be useful to use them for average reward updates.

10 Acknowledgements

I would like to thank Prof. Rich Sutton for providing ideas and feedback during the project. I am grateful to Abhishek Naik for his inputs during the discussions about multi-step algorithms.

References

De Asis, K. and Sutton, R. S. (2018). Per-decision multi-step temporal difference learning with control variates. *arXiv preprint arXiv:1807.01830*.

Moore, A. W. (1990). Efficient memory-based learning for robot control.

Ross, S. (2013). Chapter 9 - variance reduction techniques. In *Simulation*, pages 153 – 231. Academic Press, fifth edition.

Schwartz, A. (1993). A reinforcement learning method for maximizing undiscounted rewards. In *Proceedings of the Tenth International Conference on Machine Learning*, pages 298–305.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tsitsiklis, J. N. and Van Roy, B. (1999). Average cost temporal-difference learning. *Automatica*, 35:1799–1808.

Appendix A Eligibility Traces for Off-Policy Prediction

In the derivation, λ is assumed to be a function of t even though the project only uses a constant λ. The differential λ return in off-policy prediction problem is defined as

$$\begin{aligned} G_t^{\lambda s} &\doteq \rho_t \left(R_{t+1} - \bar{R}_t + (1-\lambda_{t+1})\,\hat{v}\left(S_{t+1}, \mathbf{w}_t\right) + \lambda_{t+1} G_{t+1}^{\lambda s}\right) \\ &= \rho_t \left(R_{t+1} - \bar{R}_t + \hat{v}\left(S_{t+1}, \mathbf{w}_t\right)\right) - \rho_t \lambda_{t+1} \hat{v}\left(S_{t+1}, \mathbf{w}_t\right) + \rho_t \lambda_{t+1} G_{t+1}^{\lambda s} \end{aligned}$$

Using $\delta_t^s = \rho_t \left(R_{t+1} - \bar{R}_t + \hat{v}\left(S_{t+1}, \mathbf{w}_t\right)\right) - \hat{v}\left(S_t, \mathbf{w}_t\right)$ and unrolling the recursion,

$$G_t^{\lambda s} \approx \hat{v}\left(S_t, \mathbf{w}_t\right) + \sum_{k=t}^{\infty} \delta_k^s \prod_{i=t+1}^{k} \lambda_i \rho_{i-1}$$

Using the above form of return, and the fact that weight update is $\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha \left(G_t^{\lambda s} - \hat{v}\left(S_t, \mathbf{w}_t\right)\right) \nabla \hat{v}\left(S_t, \mathbf{w}_t\right)$, the sum of updates in forward view is

$$\begin{aligned} \sum_{t=1}^{\infty} \left(w_{t+1} - w_t\right) &= \sum_{t=1}^{\infty} \alpha \nabla \hat{v}\left(S_t, \mathbf{w}_t\right) \sum_{k=t}^{\infty} \delta_k^s \prod_{i=t+1}^{k} \lambda_i \rho_{i-1} \\ &= \sum_{k=1}^{\infty} \alpha \delta_k^s \sum_{t=1}^{k} \nabla \hat{v}\left(S_t, w_t\right) \prod_{i=t+1}^{k} \lambda_i \rho_{i-1} \end{aligned}$$

Since the sum is same in forward and backward view, the eligibility trace at time k is

$$\begin{aligned} z_k &= \sum_{t=1}^{k} \nabla \hat{v}\left(S_t, w_t\right) \prod_{i=t+1}^{k} \lambda_i \rho_{i-1} \\ &= \lambda_k \rho_{k-1} \sum_{t=1}^{k-1} \nabla \hat{v}\left(S_t, w_t\right) \prod_{i=t+1}^{k-1} \lambda_i \rho_{i-1} + \nabla \hat{v}\left(S_k, w_k\right) \\ &= \lambda_k \rho_{k-1} z_{k-1} + \nabla \hat{v}\left(S_k, w_k\right) \end{aligned}$$

A.1 Control Variates

With control variate, λ return becomes

$$G_t^{\lambda s} \doteq \rho_t \left(R_{t+1} - \bar{R}_t + (1-\lambda_{t+1})\,\hat{v}\left(S_{t+1}, \mathbf{w}_t\right) + \lambda_{t+1} G_{t+1}^{\lambda s}\right) + (1-\rho_t)\,\hat{v}\left(S_t, \mathbf{w}_t\right)$$

Using $\delta_t^s = R_{t+1} - \bar{R}_t + \hat{v}\left(S_{t+1}, \mathbf{w}_t\right) - \hat{v}\left(S_t, \mathbf{w}_t\right)$ and unrolling the recursion,

$$G_t^{\lambda s} \approx \hat{v}\left(S_t, \mathbf{w}_t\right) + \rho_t \sum_{k=t}^{\infty} \delta_k^s \prod_{i=t+1}^{k} \lambda_i \rho_i$$

With similar line of reasoning as previous derivation,

$$\sum_{t=1}^{\infty}(w_{t+1}-w_t)=\sum_{k=1}^{\infty}\alpha\delta_k^s\sum_{t=1}^{k}\rho_t\nabla\hat{v}\left(S_t,w_t\right)\prod_{i=t+1}^{k}\lambda_i\rho_i$$

$$\begin{aligned}z_k&=\sum_{t=1}^{k}\rho_t\nabla\hat{v}\left(S_t,w_t\right)\prod_{i=t+1}^{k}\lambda_i\rho_i\\&=\lambda_k\rho_k\sum_{t=1}^{k-1}\nabla\hat{v}\left(S_t,w_t\right)\prod_{i=t+1}^{k-1}\lambda_i\rho_{i-1}+\rho_k\nabla\hat{v}\left(S_k,w_k\right)\\&=\rho_k\left(\lambda_k z_{k-1}+\nabla\hat{v}\left(S_k,w_k\right)\right)\end{aligned}$$

Appendix B Eligibility Traces for Off-Policy Control

The derivation for eligibility traces in off-policy control case follows the same process as done in prediction case but with action values and corresponding control variates. λ return with action values is given by

$$\begin{aligned}G_t^{\lambda a}&\doteq R_{t+1}-\bar{R}_t+\rho_{t+1}\left[\left(1-\lambda_{t+1}\right)\hat{q}\left(S_{t+1},A_{t+1},\mathbf{w}_t\right)+\lambda_{t+1}G_{t+1}^{\lambda a}\right]\\&=R_{t+1}-\bar{R}_t+\rho_{t+1}\hat{q}\left(S_{t+1},A_{t+1},\mathbf{w}_t\right)-\rho_{t+1}\lambda_{t+1}\hat{q}\left(S_{t+1},A_{t+1},\mathbf{w}_t\right)+\rho_{t+1}\lambda_{t+1}G_{t+1}^{\lambda a}\end{aligned}$$

Using $\delta_t^a=R_{t+1}-\bar{R}_t+\rho_{t+1}\hat{q}\left(S_{t+1},A_{t+1}\mathbf{w}_t\right)-\hat{q}\left(S_t,A_t,\mathbf{w}_t\right)$ and unrolling the recursion,

$$G_t^{\lambda a}\approx\hat{q}\left(S_t,A_t,\mathbf{w}_t\right)+\sum_{k=t}^{\infty}\delta_k^a\prod_{i=t+1}^{k}\lambda_i\rho_i$$

Using sum of forward updates and finding eligibility trace,

$$\sum_{t=1}^{\infty}(w_{t+1}-w_t)=\sum_{k=1}^{\infty}\alpha\delta_k^a\sum_{t=1}^{k}\nabla\hat{q}\left(S_t,A_t,w_t\right)\prod_{i=t+1}^{k}\lambda_i\rho_i$$

$$\begin{aligned}z_k&=\sum_{t=1}^{k}\nabla\hat{q}\left(S_t,A_t,w_t\right)\prod_{i=t+1}^{k}\lambda_i\rho_i\\&=\lambda_k\rho_k\sum_{t=1}^{k-1}\nabla\hat{q}\left(S_t,A_t,w_t\right)\prod_{i=t+1}^{k-1}\lambda_i\rho_{i-1}+\nabla\hat{q}\left(S_k,A_k,w_k\right)\\&=\lambda_k\rho_k z_{k-1}+\nabla\hat{q}\left(S_k,A_k,w_k\right)\end{aligned}$$

B.1 Control Variates

With control variate, λ return becomes

$$\begin{aligned}G_t^{\lambda a}&\doteq R_{t+1}-\bar{R}_t+\left(1-\lambda_{t+1}\right)\bar{V}_t(S_{t+1})+\lambda_{t+1}\left[\rho_{t+1}G_{t+1}^{\lambda a}+\bar{V}_t(S_{t+1})-\rho_{t+1}\hat{q}\left(S_{t+1},A_{t+1},\mathbf{w}_t\right)\right]\\&\approx\hat{q}\left(S_t,A_t,\mathbf{w}_t\right)+\sum_{k=t}^{\infty}\delta_k^a\prod_{i=t+1}^{k}\lambda_i\rho_i\end{aligned}$$

with $\delta_t^a=R_{t+1}-\bar{R}_t+\bar{V}_t(S_{t+1})-\hat{q}\left(S_t,A_t,\mathbf{w}_t\right)$ and $\bar{V}_t(S_{t+1})=\mathbb{E}_\pi[\hat{q}\left(S_{t+1},\cdot,\mathbf{w}_t\right)]$

Since the form of return is same with the only change being in δ_t^a, the rest of the derivation is exactly same as before, giving

$$z_k=\lambda_k\rho_k z_{k-1}+\nabla\hat{q}\left(S_k,A_k,w_k\right)$$