Project Report

CS687: Reinforcement Learning

Chirag Trasikar Sahil Jindal

1 Introduction

In this project, we implement the following three Reinforcement Learning algorithms: REINFORCE (for extra credit), One-Step Actor Critic and True Online SARSA(λ). We also experiment with the following environments: Cart Pole, Mountain Car and Acrobot (for extra credit). We implemented all algorithms using NumPy and used existing environments provided by OpenAI Gym for this project. The following sections describe the algorithms that we implemented and the results of the experiments that we performed.

Task Distribution: Chirag Trasikar experimented with REINFORCE and One-Step Actor Critic on Cart Pole, Mountain Car and Acrobot environments. Sahil Jindal experimented with True Online SARSA(λ) on Cart Pole, Mountain Car and Acrobot environments. We tuned hyperparameters and interpreted the results together.

2 Algorithms

We referred to the RL book - Reinforcement Learning: An Introduction by Sutton and Barto for the True Online SARSA algorithm and we referred to CS687 slides by Prof. Bruno Castro da Silva for REINFORCE and One-Step Actor Critic algorithms.

2.1 REINFORCE

The pseudo-code for the REINFORCE algorithm is shown in Algorithm 1.

Algorithm 1: REINFORCE.

1) Input: A differentiable parameterized policy, $\pi(\cdot, \cdot; \theta)$
2) Algorithm Parameter: Step size, $\alpha \in (0, 1]$
3) Initialization: Initialize policy parameters θ, eg. to zero.

while `true` **do**
  Generate an episode $S_0, A_0, R_0, S_1, A_1, ...R_T, S_T$ using $\pi(\cdot, \cdot; \theta)$;
  for *each step of the episode,* $t = 0, 1, ..., T-1$ **do**
    $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} R_k$;
    $\theta \leftarrow \theta + \alpha \cdot G_t \frac{\partial \ln(\pi(S_t, S_t; \theta))}{\partial \theta}$

Policy Gradient algorithms try to learn a parameterized policy π^θ which takes as input the state (or features extracted from the state) and returns a probability distribution over actions. They learn the parameters θ by performing gradient ascent in order to maximize the objective: the performance of the policy.

The performance of the policy is defined as follows.

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T} \gamma^t R_t \big| \pi^\theta\Big] \tag{1}$$

The policy parameters theta are updated using the gradient ascent update, since we want to maximize the objective.

$$\theta_{t+1} \rightarrow \theta_t + \alpha \nabla J(\theta_t) \tag{2}$$

Now, according to the **Policy Gradient Theorem**, if $\frac{\partial \pi(s,a;\theta)}{\partial \theta}$ exists for all s, a, θ then for all finite MDPs with bounded rewards:

$$\nabla J(\theta) \propto \mathbb{E}\Big[\sum_{t=0}^{\infty} \gamma^t q^{\pi^\theta}(s, a) \frac{\partial \ln \pi(s, a; \theta)}{\partial \theta}\Big] \tag{3}$$

The REINFORCE algorithm obtains estimates of $\nabla J(\theta)$ by simulating episodes according to the policy π^θ and using the unbiased estimator G_t for q^{π^θ}. G_t is the return obtained after time t.

The algorithm also drops the γ^t term from the expression for $\nabla J(\theta)$ because it causes the gradient updates to be very small, making the learning process extremely slow.

2.2 One-Step Actor Critic

The main problem with the REINFORCE algorithm is that the estimator G_t of q^{π^θ} has very high variance. This causes the performance of the algorithm to be dependent on how good each episode was. If a particular episode was bad, the policy might drastically change for the worse.

To decrease the variance, we can use baselines. Baselines can be used with estimators in order to create a new estimator such that the variance of the new estimator is lower than that of the old estimator but the expected value of the new estimator the same as that of the old estimator. A good baseline is one which is correlated with the estimator. The value function v^{π^θ}, is correlated with the return G_t, hence, we can use the value function v^{π^θ}.

Now, the problem is that we do not know v^{π^θ}. Therefore, One-Step Actor Critic tries to learn v^{π^θ} using a parameterized value function approximator $\hat{v}_w$.

To decrease the variance further we can use another estimator of q^{π^θ}: the TD(0) estimate of the return, $R_t + \gamma v(s')$.

The new estimator now is $R_t + \gamma \hat{v}^\pi_w(s') - \hat{v}^\pi_w(s)$ which is nothing but the TD-error δ_t. Therefore, One-Step Actor Critic learns both value function approximator parameters w and policy parameters θ alternately, ans uses the following update rules:

$$w_{t+1} \leftarrow w_t + \alpha^w \delta \nabla \hat{v}_{w_t}(s) \tag{4}$$

$$\theta_{t+1} \leftarrow \theta_t + \alpha^\theta \delta \nabla \ln(\pi(s, a; \theta)) \tag{5}$$

The pseudo-code for the One-Step Actor Critic algorithm is shown in Algorithm 2.

Algorithm 2: One-Step Actor Critic.

1) Input: A differentiable parameterized policy, $\pi(\cdot, \cdot; \theta)$
2) Input: A differentiable parameterized value function approximator, $\hat{v}_w$
3) Algorithm Parameter: Step sizes, $\alpha^\theta \in (0, 1]$ and $\alpha^w \in (0, 1]$
4) Initialization: Initialize policy parameters θ and value function approximator parameters w, eg. to zero.

```
while true do
    Initialize s ~ d_0;
    for each step of the episode, t = 0, 1, ..., T − 1 do
        a ~ π(s, ·; θ);
        Execute action a, observe reward r and next state s';
        δ ← r + γ v̂^π_w(s') − v̂^π_w(s);
        w ← w + α^w δ ∇ v̂_w(s);
        θ ← +α^θ δ ∇ ln(π(s, a; θ));
        s ← s'
```

2.3 True Online SARSA(λ)

SARSA(λ) is an extension over the normal SARSA algorithm which uses eligibility trace $e(s, a)$. Eligibility e(s,a) can be computed using

$$e(s,a) = \begin{cases} \gamma\lambda e(s,a) + 1 & \text{if } s = s_t \text{ and } a = a_t, \\ \gamma\lambda e(s,a) & \text{otherwise} \end{cases}$$

SARSA(λ) tries to approximate Q^{π^*} using the update rule:

$$Q_{t+1}(s,a) \leftarrow Q_t(s,a) + \alpha\delta e(s,a)$$

where, δ is defined by

$$\delta = r_t + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)$$

In True Online SARSA we calculate Q(s,a,w) as defined:

$$Q(s,a,w) = x(s,a)^T w$$

where, x is the Fourier feature function and w are the weights.

The pseudo-code for the True Online SARSA(λ) algorithm is shown in Algorithm 3.

3 Environments

In this project we experiment with the *CartPole-v0, MountainCar-v0* and *Acrobot-v1* environments provided by the OpenAI Gym. Below, we describe each of these three environments.

Algorithm 3: True Online SARSA(λ).

1) Input: A feature function, $x : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^d$
2) Algorithm Parameter: Step size, $\alpha \in (0, 1]$, trace decay rate $\lambda \in [0, 1]$ and exploration parameter $\epsilon \in [0, 1]$
3) Initialization: Initialize $w \in \mathbb{R}^d$, eg. to zero.

while `true` **do**
  Initialize $S \sim d_0$;
  Choose $A \sim \pi(\cdot|S)$ or ϵ-greedy according to $\hat{q}(S, \cdot; w)$;
  $x \leftarrow x(S, A)$;
  $z \leftarrow \overrightarrow{0}$;
  $Q_{old} \leftarrow 0$;
  for *each step of the episode,* $t = 0, 1, ..., T - 1$ **do**
    Take action A, observe R, S' Choose $A' \sim \pi(\cdot|S')$ or ϵ-greedy according to $\hat{q}(S', \cdot; w)$;
    $x' \leftarrow x(S', A')$;
    $Q \leftarrow w^\top x$;
    $Q' \leftarrow w^\top x'$;
    $\delta \leftarrow R + \gamma Q' - Q$;
    $z \leftarrow \gamma\lambda z + (1 - \alpha\gamma\lambda z^\top x)x$;
    $w \leftarrow w + \alpha^w \delta \nabla \hat{v}_w^(s)$;
    $\theta \leftarrow \theta + \alpha(\delta + Q - Q_{old})z - \alpha(Q - Q_{old})x$;
    $Q \leftarrow Q'$;
    $x \leftarrow x'$;
    $A \leftarrow A'$;

3.1 Cart Pole

In the Cart Pole environment, the agent is supposed to try to balance the pole for as long as possible by moving the cart. For the Cart Pole problem, $\mathcal{S} \in \mathbb{R}^4$. The states are a 4 element long vectors which contain the cart's position, the cart's velocity, the pole's angular position and the pole's angular velocity. The agent can execute only two actions which move the cart either to the left or right, i.e. $\mathcal{A} \in \{0, 1\}$. The state transition model p and reward distribution R are unknown to the agent. The environment returns a reward of $+1$ per timestep till the episode terminates. The episode terminates either when the cart moves out of the screen (a specific range of allowed cart positions), or the pole's angular position is greater than some fixed value on either side. The episode also terminates after 200 timesteps if the pole has been balanced for that long.

3.2 Mountain Car

In the Mountain Car environment, the agent is supposed to reach the peak of the mountain on the right, starting from a valley, by moving the car. The car cannot directly reach the peak and it is supposed to use the velocity gained by oscillating between the mountains, to reach the peak of the right side mountain. $\mathcal{S} \in \mathbb{R}^2$. That is, the states are 2 element long vectors representing the car's position and velocity. There are only 3 actions that the agent can execute: move the car to the left, move the car to the right, do not move the car. Therefore, $\mathcal{A} \in \{0, 1, 2\}$. The state transition model p and reward distribution R are unknown to the agent. The environment returns a reward of -1 per timestep till the episode terminates. The episode terminates when after 200 timesteps if the car does not reach the mountain peak, and the episode also terminates if the car reaches the mountain peak.

3.3 Acrobot (Extra Credit)

In the Acrobot environment, the agent is supposed to move the central joint of the acrobot to make the acrobot's lower arm cross the black line (make the arm reach a particular height). The states are 6 element long vectors representing the angular positions and velocities of various parts of the acrobot, i.e. $\mathcal{S} \in \mathbb{R}^6$. There are three actions that the agent can perform to move the central joint of the acrobot. Therefore, $\mathcal{A} \in \{0, 1, 2\}$. The rewards are -1 for each timestep till the episode ends. The episode terminates in 500 timesteps if the agent cannot make the acrobot's arm reach the line or it terminates immediately after the acrobot reaches the line.

4 Experiments

We generate learning curves for each algorithm for most of the environments. These learning curves show the average episode length versus the episode number. We obtain results by performing 20 trials for REINFORCE and One-Step Actor Ctiric and 10 trials for True-Onlie SARSA(λ). In the CartPole environment the goal is to keep the pole balanced for as long as possible, so the episode lengths should increase as the agent learns to balance the pole. In the Mountain Car and Acrobot environments, the

goal of the agent is to reach the destination as fast as possible. So with the number of episodes, as the agent learns, the episode lengths should keep decreasing. We implemented all the algorithms and required feature construction steps using NumPy only. As mentioned above, we use OpenAI Gym for the environments. The following subsections describe our approach and experimental results.

4.1 Feature Construction: Fourier Basis

We use the Fourier Basis function to construct features for our algorithms. The Fourier Basis function takes as input, the continuous state vector $S \in \mathbb{R}^k$, and the order n. With this input, it produces $(n+1)^k$ features as follows.

We first normalize the state vector S so that all elements within the state vector are between 0 and 1. For normalization, we require the highest and lowest possible values for each element of the state vector. We determine these *high* and *low* values either by running episodes with random actions (for Cart Pole) or by using the *high* and *low* values provided by OpenAI Gym.

After normalizing the state, we generate a $C = (n+1)^k \times k$ matrix. C is a collection of row vectors with k elements such that each element of the k-sized row vectors can vary from 0 to n (the order of Fourier Basis).

Finally we compute the Fourier Basis $\phi(S, n)$ as follows.

$$\phi(S, n) \leftarrow \cos\left[\pi \cdot S \cdot C^\top\right] \tag{6}$$

$S \cdot C^\top$ will be a $(n+1)^k$-sized vector. The cos of this vector indicates individually applying the cos function to all elements of this vector.

4.2 Results

In this section we will state the learning curves and the hyper parameters used.

4.2.1 REINFORCE (Extra Credit)

We use a linear approximator θ to estimate the policy π^θ as shown below. We attempted to use a 2-layer neural network approximator (also implemented in NumPy), however the linear approximator with Fourier Basis functions worked better.

$$\pi(\cdot|S;\theta) \leftarrow \text{softmax}(\sigma \cdot \theta^\top \phi(S, n)) \tag{7}$$

For the Cart Pole environment we use order $n = 4$ Fourier Basis functions. Whereas for Mountain Car and Acrobot environments, we use order $n = 10$ and $n = 4$ respectively. This is because the state vector length is just 2 for Mountain Car, whereas for CartPole and Acrobot is is 4 and 6 respectively. Hence we can afford to use higher order Fourier Basis functions when working on Mountain Car.

With REINFORCE we use a learning rate of $\alpha = 10^{-3}$ without any decay for Cart Pole, $\alpha = 5 \times 10^{-3}$ for Mountain Car and $\alpha = 10^{-3}$ for Acrobot. These are the highest learning rates we found that do not cause the gradients wrt. θ to explode.

We use an exploration factor $\sigma = 1.$ for Cart Pole and Mountain Car environments but for Acrobot we use $\sigma = 0.01$. Lower the value of σ higher the exploration. In the Acrobot problem the state set is larger than Mountain Car and Cart Pole. Hence, we encourage more exploration by lowering the value of σ in the Acrobot problem.

The learning curves for REINFORCE are shown in Figure 1. Clearly, there is a lot of variance in the episode lengths across multiple runs of REINFORCE for the Cart Pole problem. This is because the returns G_t has high variance. For Mountain Car and Acrobot environments, the agent barely even reaches the destination once, and the agent does not learn any thing within the allowed number of iterations. We tried multiple hyper parameter choices, however, none of them worked well for REINFORCE on Maountain Car and Acrobot.

4.2.2 One-Step Actor Critic

We use a linear approximator θ as the actor to estimate the policy π^θ as shown below.

$$\pi(\cdot|S;\theta) \leftarrow \text{softmax}(\theta^\top \phi(S, n)) \tag{8}$$

We use another linear approximator w as the critic to estimate the value of states $\hat{v}_w(S)$.

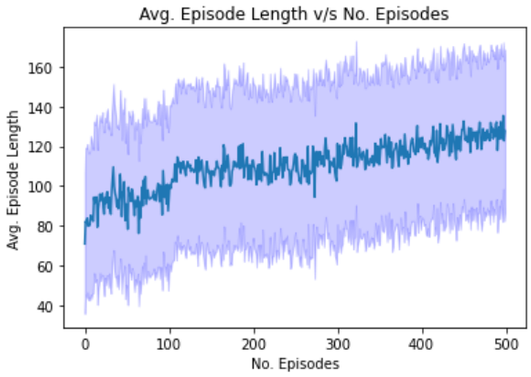
$$\hat{v}_w(S) \leftarrow w^\top \phi(S, n) \tag{9}$$

With our One-Step Actor Critic implementation, we use the same Fourier Basis function orders for each environment as we did for the REINFORCE implementation.

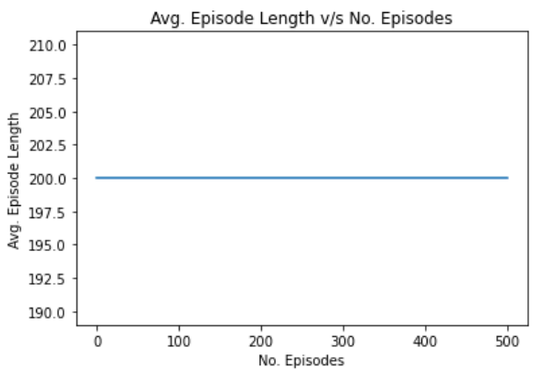
For Cart Pole we use learning rates $\alpha^\theta = 10^{-3}$ and $\alpha^w = 10^{-4}$. For Mountain Car we use $\alpha^\theta = 5 \times 10^{-3}$ and $\alpha^w = 5 \times 10^{-3}$. Finally, for Acrobot we use $\alpha^\theta = 10^{-3}$ and $\alpha^w = 10^{-4}$ again.

For our One-Step Actor critic we use the same exploration factors σ again.

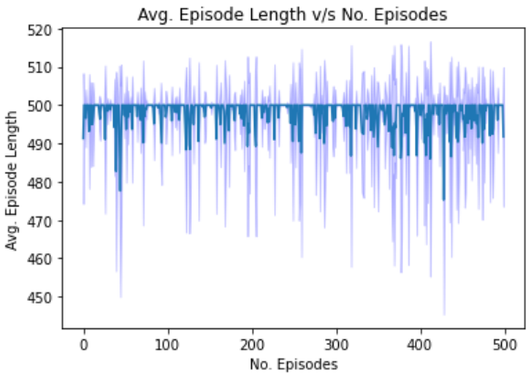
The learning curves for One-Step Actor Critic are shown in Figure 2. These results are much better than that of REINFORCE, in the sense that there is lower variance in the episode lengths as the agent learns over episodes.



(a) Cart Pole



(b) Mountain Car



(c) Acrobot

Figure 1: REINFORCE evaluated on 3 environments (Extra Credit)

4.2.3 True Online SARSA(λ)

We chose action using ϵ greedy according to $\hat{q}(S', \cdot; w)$

For Cart Pole we use learning rates $\alpha = 5 \times 10^{-5}$, $\epsilon = 0.05$, $\lambda = 0.9$, $\gamma = 1$ and $k = 5$. Number of episodes taken were 500. For Mountain Car we use $\alpha = 5 \times 10^{-5}$, $\epsilon = 0.05$, $\lambda = 0.9$, $\gamma = 1$ and $k = 5$. Finally, for Acrobot we use $\alpha = 1 \times 10^{-5}$, $\epsilon = 0.05$, $\lambda = 0.9$, $\gamma = 1$ and $k = 3$.

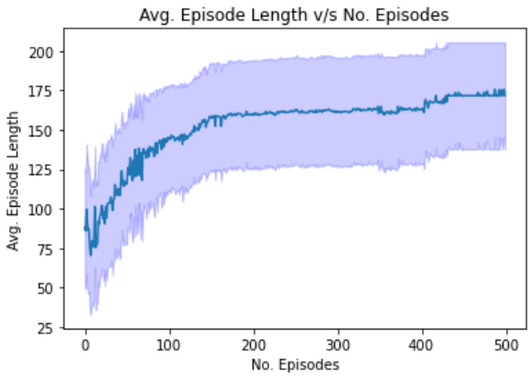
The learning curves for True SARSA(λ) are shown in Figure 3. Cart-Pole and Acrobot results of True Online SARSA(λ) are better than the REINFORCE algorithm.

5 Conclusion

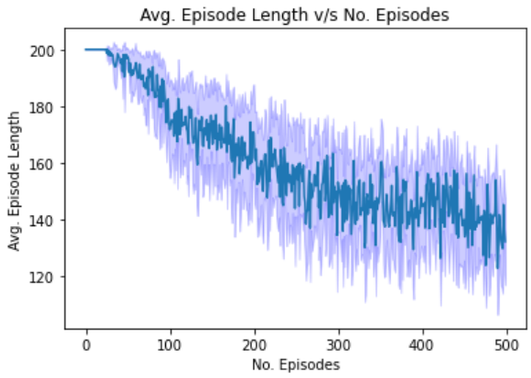
In this project we performed one policy gradient algorithm - REINFORCE, the One-Step Actor Critic Algorithm and the True Online SARSA(λ) algorithm. These were separately performed on three different environments namely Cart Pole, Mountain Car and Acrobot using OpenAI Gym.

Observing the learning curves one can see that the variance of Cart-Pole decreases and the average episode length increases as the number of episodes increases. Compared to the Actor-Critic method, True Online SARSA(λ) performs better for Cart pole. Mountain Car problem seems to perform similarly for Actor-Critic and True Online SARSA(λ) although True Online SARSA(λ) has slightly less variance compared to One-Step Actor Critic. For Acrobot, True Online SARSA(λ) converges faster compared to One-Step Actor Critic algorithm.

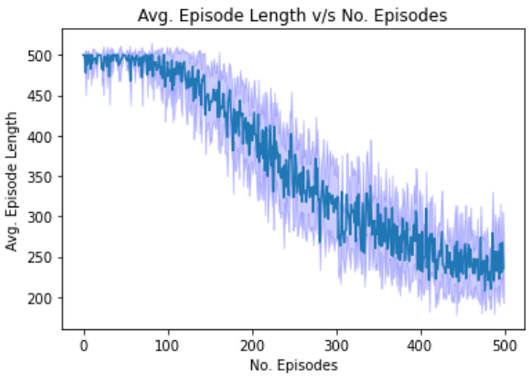
In conclusion, we can see that True Online SARSA(λ) and One-Step Actor Critic performed much better than REINFORCE.



(a) Cart Pole

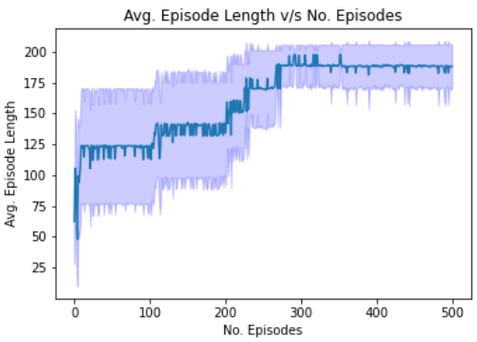


(b) Mountain Car

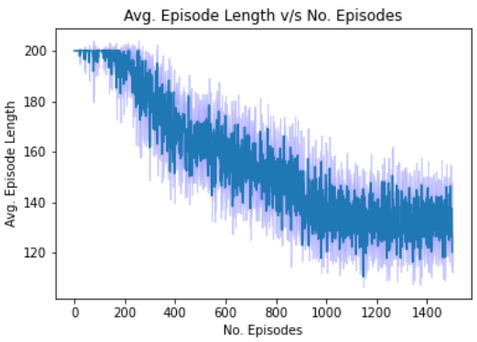


(c) Acrobot (Extra Credit)

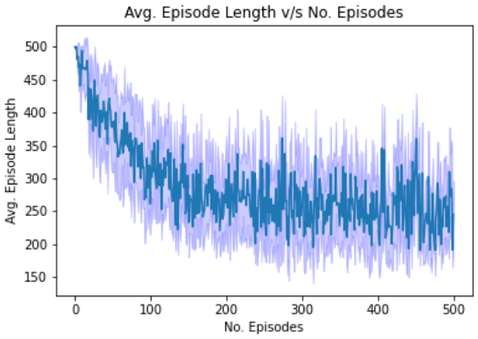
Figure 2: One-Step Actor Critic evaluated on 3 environments



(a) Cart Pole



(b) Mountain Car



(c) Acrobot (Extra Credit)

Figure 3: True Online SARSA(λ) evaluated on 3 environments