Analysis of Actor-Critic Algorithms And it's Variants

Abhishek R

Indian Institute of Technology Dharwad

June 9, 2020

Outline

What is Reinforcement Learning?

- Branch of machine learning concerned with taking sequences of actions
- It is concerned with the agent interacting with an unknown environment and trying to **maximize cumulative rewards** obtained by taking right actions.

Examples

- Inventory Management
  - Observations: current inventory levels
  - Actions: number of units of each item to purchase
  - Rewards: profit
- Robotics
  - Observations: camera images, joint angles
  - Actions: joint torques
  - Rewards: Open the bottle cap, placing the bottle, lifting the bottle

How do we formulate RL problems? - Markov Decision Process (MDP)

- MDP consists of a tuple of 5 elements: **S**, **A**, $\mathbf{p(s_{t+1}|s_t, a_t)}$, $\mathbf{R(r_{t+1}|s_t, a_t)}$, **Discount factor($\gamma$)**
- All states in MDP has "Markov" property, referring to the fact that the future only depends on the current state, not the history.

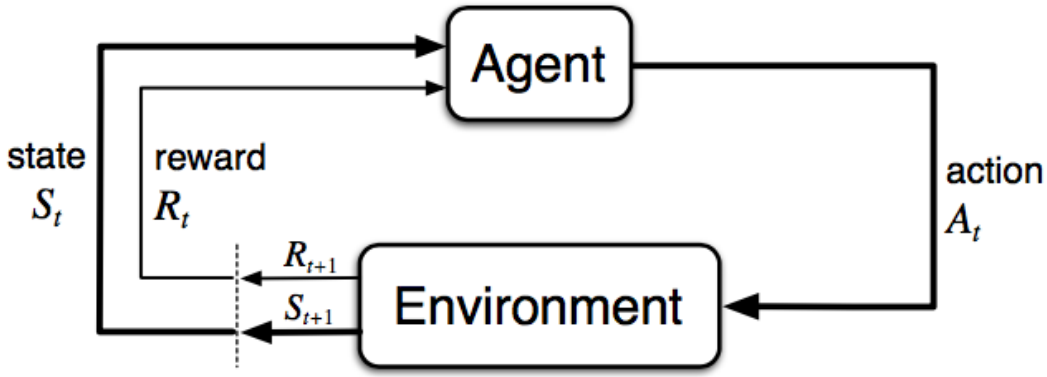


Figure 1: The agent-environment interaction in a Markov decision process. (Image source: Sutton and Barto (2017).)

Policy

- Deterministic policies: a = π(s)
- Stochastic policies: a$\sim$ π(a|s)
- Parameterized policies: π_θ

- **On Policy**: Here we optimize the current policy and use it to determine what actions to explore next.
- **Off Policy**: Here learning allows a second policy, like the ϵ-greedy policy, to explicitly define the exploration it wants. Off-policy learning optimizes the current policy but use another policy to explore actions.

- **State-action Value(Q Value):**
  Is a measure of the overall expected reward assuming the Agent is in state s and performs action a, and then continues playing until the end of the episode following some policy π.

$$Q_\pi(s,a) = E_\pi[\sum_{t=0}^{t=T-1} \gamma^t r_t | S = s, A = a]$$

- **State Value Function:**
  It is the expected reward for an agent starting from state s by following a policy by which the agent picks actions to perform.

$$V_\pi(s) = E_\pi[\sum_{t=0}^{t=T-1} \gamma^t r_t | S = s]$$

Monte-Carlo Methods

- In Monte Carlo (MC) we play an episode of the game starting by some random state (not necessarily the beginning) till the end and we record the states, actions and rewards that we encountered and then compute the V(s) and Q(s,a) for each state we passed through.
- Disadvantage is need to wait until the game terminates to calculate V and Q
- And problem is if the game never terminates.

Solving RL Problems

- **Value based:**
  Here we try to approximate the optimal Value function. Higher the value, the better the action. The most famous algorithm is Q learning and all its enhancements like Deep Q Networks, Double Dueling Q Networks, etc
- **Policy based:** Policy Based algorithms like Policy Gradients and REINFORCE try to find the optimal policy directly.

Each method has their own advantages. While **Policy based** are:

- Better convergence properties
- Effective in high-dimensional or continuous action spaces
- Can learn stochastic policies
- Policy based typically converge to local rather then global optimum.
- Evaluating a policy is typically inefficient and high variance.

On the other hand **Value Based** are:

- More sample efficient
- But they are less stable and suffer from poor convergence
- Less reliable

This is where **Actor-Critic** comes to surface, taking the advantage of both the methods and trying to optimize both policy and value network.

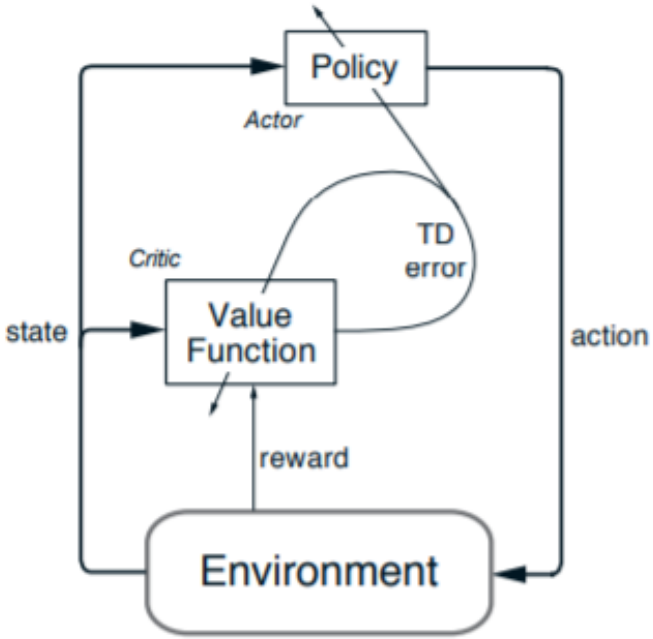


Figure 2: Image source: Sutton and Barto (2017)

Policy Gradient

In Policy Gradient we directly estimate the optimal policy using neural networks.

- Making the good trajectories more probable
- Making the good actions more probable - which we will see in actor-critic methods

The main objective we need to achieve is to maximize expectation of Total Reward for a policy.

$$J(\theta) = E_{\pi_\theta}[R]$$

We use gradient based search for θ by updating θ in following way:

$$\theta \leftarrow \theta + \alpha \nabla J(\theta)$$

Where policy π is parameterized by θ

Gradient of the objective functions

Lets consider a whole trajectory

$\tau = (s_0, a_0, r_0, s_1, a_1, r_1, ..., s_{T-1}, a_{T-1}, r_{T-1}, s_T)$

$$\nabla_\theta E_{\tau \sim \pi(\tau|\theta)}[R(\tau)] = E_\tau[\nabla_\theta \log \pi(\tau|\theta) R(\tau)]$$

We can write $\pi(\tau|\theta)$ as:

$$\pi(\tau|\theta) = \mu(s_0) \prod_{t=0}^{T-1} [\pi(a_t|s_t, \theta) P(s_{t+1}, r_t|s_t, a_t)]$$

$$\log \pi(\tau|\theta) = \log \mu(s_0) + \sum_{t=0}^{T-1} [\log \pi(a_t|s_t, \theta) + \log p(s_{s+t}, r_t|s_t, a_t)]$$

$$\nabla_\theta \log \pi(\tau|\theta) = \nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t, \theta)$$

So, the final expression becomes as:

$$\nabla_\theta E_\tau[R] = E_\tau[R\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t,\theta)]$$

$$\boxed{\nabla_\theta E_\tau[R] = E_\tau[(\sum_{t=0}^{T-1} r_t)\nabla_\theta \sum_{t=0}^{T-1} \log \pi(a_t|s_t,\theta)]} \quad (1)$$

- We can say that for t´ < t

$$E_\tau[r_t \nabla_\theta \log \pi_\theta(a_t|s_t)] = 0$$

Gradients of logprob of future states don't affect the present state.

$$\nabla_\theta E_\tau[R] = E_\tau[r_{t'} \sum_{t=0}^{t'} \nabla_\theta \log \pi(a_t|s_t,\theta)]$$

- Taking sum over all reward terms we get

$$\nabla_\theta E_\tau[R] = E_\tau[(\sum_{t=0}^{T-1} r_{t'}) \sum_{t=0}^{t'} \nabla_\theta \log \pi(a_t|s_t, \theta)]$$

- By rearranging terms, we get each logprob multiplied by sum of future reward term

$$\boxed{\nabla_\theta E_\tau[R] = E_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) \sum_{t'=t}^{T-1} r_{t'}]}$$

- We can replace the future reward sum with future discounted reward sum:

$$\boxed{\nabla_\theta E_\tau[R] = E_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) \sum_{t'=t}^{T-1} \gamma^{t-t'} r_{t'}]} \quad (2)$$

Reinforce Algorithm

Algorithm 1 REINFORCE Algorithm

Algorithm parameters: learning rate α, discount factor γ
Initialize θ arbitrarily
for *each episode* $\{s_0, a_0, r_0, \ldots s_{T-1}, a_{T-1}, r_{T-1}, s_T\} \sim \pi_\theta$ **do**
  for *t=1 to T-1* **do**
    $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) G_t$;
  end
end
return θ

where

$$G_t = \sum_{t'=t}^{T-1} \gamma^{t-t'} r_{t'}$$

is the Future discounted reward

Adding a Baseline

- Above Algorithm(Monte Carlo) has no bias but high variance. Variance hurts deep learning optimization.
- One way we can achieve this by subtracting the R from a baseline as below given below:

$$\nabla_\theta E[R] = E[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta)(\sum_{t'=t}^{T-1} \gamma^{t-t'} r_{t'} - E_{\tau_1,\tau_2..} G(\tau)] \tag{3}$$

- Baseline should be independent of policy parameter to keep the gradient estimate unbiased

- Here the baseline we took was the average of total discounted reward for all the episode till now.
  Total discounted reward for an episode τ is:

$$G(\tau) = \sum_{t=0}^{t=T-1} \gamma^t r_t \tag{4}$$

Q Actor Critic

- From few modification to policy gradient method we bring in Actor Critic in architecture:

$$\boxed{\nabla_\theta E_\tau[R] = E_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t,\theta) Q_w(s_t,a_t)]} \quad (5)$$

- Value/Q function estimate behaves like a "critic" (good v/s bad values) to the "actor" (agent's policy).

Q Actor Critic Algorithm

Algorithm 2 Q Actor Critic

Initialize parameters s,θ,w and learning rate α_θ, α_w; sample a$\sim \pi_\theta(a|s)$.

for *t=0...T-1* **do**

Sample reward $r_t \sim R(s,a)$ and next state $s' \sim P(s'|s,a)$;

Then sample the nest action a$' \sim \pi_\theta(a'|s')$;

Update the policy parameters:

$\theta \leftarrow \theta + \alpha_\theta Q_w(s,a) \nabla_\theta \log \pi_\theta(a|s)$;

Compute the correction (TD error) for action-value at time t:

$\delta_t = r_t + \gamma Q_w(s',a') - Q_w(s,a)$

and use it to update the parameters of Q function:

$w \leftarrow w + \alpha_w \delta_t \nabla Q_w(s,a)$

Move to $a \leftarrow a'$ and $s \leftarrow s'$

end

Advantage Actor Critic(A2C)

- **Advantage function**:

$$A(s_t, a_t) = Q_w(s_t, a_t) - v_v(s_t)$$

- Bellman optimality equation:

$$Q(s_t, a_t) = E[r_{t+1} + \gamma V(s_{t+1})]$$

- So, we can rewrite the advantage as:

$$A(s_t, a_t) = E[r_{t+1} + \gamma V_v(s_{t+1})] - V_v(s_t)$$

- Finally the gradient is

$$\boxed{\nabla_\theta E_\tau[R] = E_\tau[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t|s_t, \theta) A(s_t, a_t)]} \quad (6)$$

Temporal-Difference (TD) Learning

- Temporal-Difference(TD) method is a blend of the **Monte Carlo (MC) method** and the **Dynamic Programming (DP) method**
- the agent learns from sampled experience (Similar to MC) and bootstraps like DP
- The backward view of TD(λ) updates values at each step. So after each step in the episode you make updates to all prior steps.
  The question is how do you weight or assign credit to all prior steps appropriately?
- **Eligibility Traces**, basically keeps a record of the frequency and recency of entering a given state.

TD(λ) Actor Critic

- The eligibility vectors can be updated according to

$$z_{t+1} = \gamma\lambda z_t + \nabla Q(s_{t+1}, a_{t+1})$$

  initialize z_{-1} to zeros.
- Before updating weights we also need to calculate TD error given as:

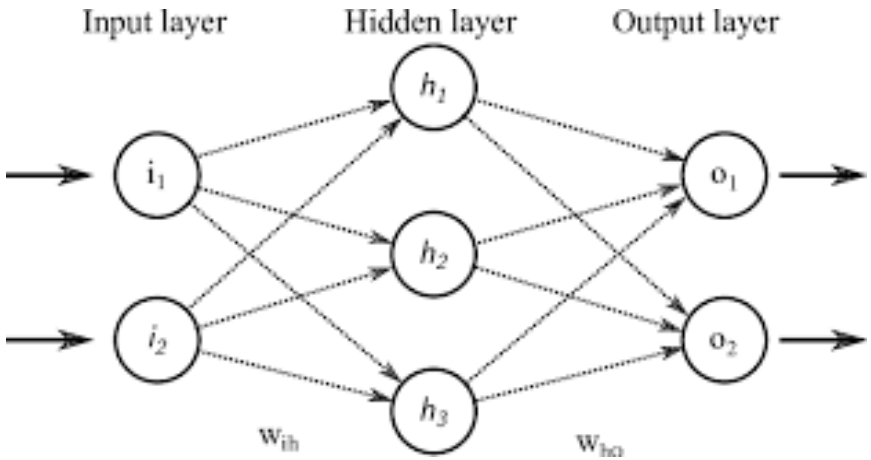
$$d_t = r_t + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

- Then we update the weights as:

$$w_{t+1} = w_t + \alpha d_t z_t$$

Konda Actor Critic

- Here we view the product of Q and the gradient $\log(\pi)$ as, Q projecting onto the space of gradient $\log(\pi)$
- What are the feature space we are using for Q?
- He proposed that features of critic should span subspace of gradient $\log(\pi)$, this will reduce the loss in predicting critic
- Here critic uses TD learning with linear approximation architecture



- Consider ϕ as the output of the layer previous to output layer of actor network. Then we can define the Q function as:

$$Q_w(s,a) = w^T[\phi_{sa} - \sum_b \pi(s,b)\phi_{sb}]$$

- Now Let's consider the temporal difference d_t corresponding to the transition from t to t+1 step:

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

- z_t be the eligibility vector as the same dimension of weights

$$w_{t+1} = w_t + \alpha d_t z_t$$

and the eligibility vectors can be updated according to:

$$z_{t+1} = \gamma\lambda z_t + \phi(s_{t+1}, a_{t+1})$$

initialize z_{-1} to zeros.

Comparison of variants

Variant	$\alpha 1$	par	γ	$\alpha 2$	par	λ	max-μ	max-σ	samp-μ	samp-σ
total	0.01	10	0.99	-	-	-	200.0	0.0	189.49	20.05
without	0.01	10	0.99	-	-	-	200.0	0.0	199.23	8.20
with	0.01	10	0.99	-	-	-	200.0	0.0	198.71	11.71
Qac	0.001	10	1	0.005	114	-	200.0	0.0	162.65	39.58
QepNorm	0.005	10	0.99	0.008	142	-	200.0	0.0	200.00	0.00
Qbaseline	0.005	10	0.99	0.01	114	-	200.0	0.0	198.69	5.65
A2C	0.001	10	0.99	0.005	226	-	200.0	0.0	192.74	14.87
TD(batch 20)	0.01	10	0.99	0.001	386	0.5	200.0	0.0	185.08	34.24
kondaTD(1)	0.005	10	0.90	0.008	8	1	200.0	0.0	200.00	0.00

Table 1: CartPole Final weights table

Variant	$\alpha 1$	par	γ	$\alpha 2$	par	λ	max-μ	max-σ	samp-μ	samp-σ
total	0.01	10	0.99	-	-	-	200.0	0.0	188.28	20.07
without	0.01	10	0.99	-	-	-	200.0	0.0	199.41	8.02
with	0.01	10	0.99	-	-	-	200.0	0.0	198.47	12.59
Qac	0.001	10	1	0.005	114	-	200.0	0.0	158.20	41.47
QepNorm	0.005	10	0.99	0.008	142	-	200.0	0.0	199.20	11.36
Qbaseline	0.005	10	0.99	0.01	114	-	200.0	0.0	198.94	4.53
A2C	0.001	10	0.99	0.005	226	-	200.0	0.0	192.00	15.24
TD(batch 20)	0.01	10	0.99	0.001	386	0.5	200.0	0.0	185.08	34.24
kondaTD(1)	0.005	10	0.90	0.008	8	1	200.0	0.0	185.94	33.39

Table 2: CartPole Optimal Weights table

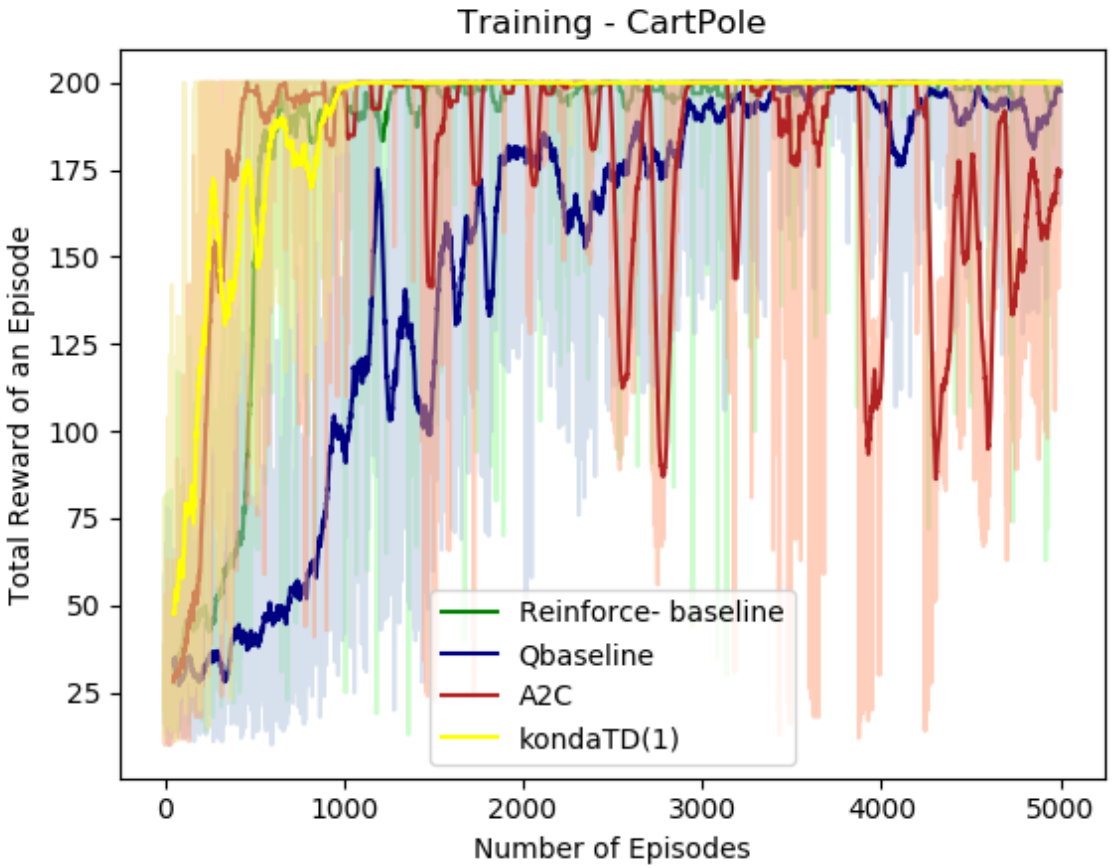


Figure 3: Comparison of few variants on CartPole environment

Variant	$\alpha 1$	par	γ	$\alpha 2$	par	λ	max-μ	max-σ	samp-μ	samp-σ
total	0.005	484	0.99	-	-	-	62.26	118.35	72.30	134.68
without	0.005	484	0.99	-	-	-	131.68	101.10	107.56	93.33
with	0.005	484	0.99	-	-	-	127.24	167.54	129.20	158.34
QepNorm	0.0008	484	0.99	0.006	4996	-	106.60	105.57	109.50	100.71
Qbaseline	0.002	36	0.99	0.008	1668	-	-48.73	23.09	96.86	48.65
A2C	0.01	36	0.99	0.0005	4996	-	191.89	104.33	193.02	101.44
A2C	0.001	836	0.99	0.005	1668	-	231.60	68.50	210.15	75.55
kondaTD(1)	0.0002	836	0.99	0.005	256	1	53.77	95.92	163.96	79.74

Table 3: LunarLander Final Weights Table

Variant	$\alpha 1$	par	γ	$\alpha 2$	par	λ	max-μ	max-σ	samp-μ	samp-σ
total	0.005	484	0.99	-	-	-	-93.98	72.65	25.85	76.26
without	0.005	484	0.99	-	-	-	145.94	56.96	142.75	54.69
with	0.005	484	0.99	-	-	-	193.39	65.02	193.45	72.58
QepNorm	0.0008	484	0.99	0.006	4996	-	14.55	126.57	13.65	116.33
Qbaseline	0.002	36	0.99	0.008	1668	-	-331.38	105.21	-9.28	63.72
A2C	0.01	36	0.99	0.0005	4996	-	201.01	100.52	192.47	101.65
A2C	0.001	836	0.99	0.005	1668	-	193.78	90.90	215.25	73.56
kondaTD(1)	0.0002	836	0.99	0.005	256	1	156.61	72.56	117.46	100.56

Table 4: LunarLander Optimal Weights Table

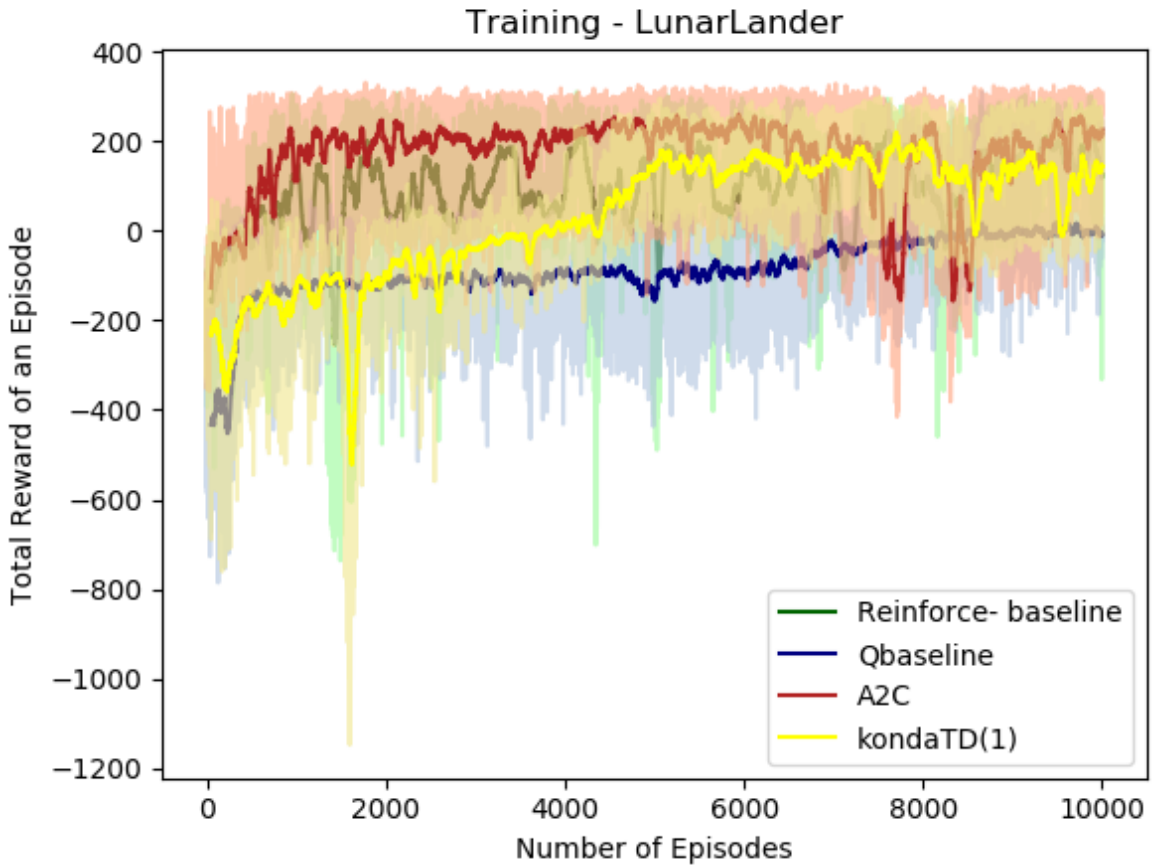


Figure 4: Comparison of few variants on LunarLander environment

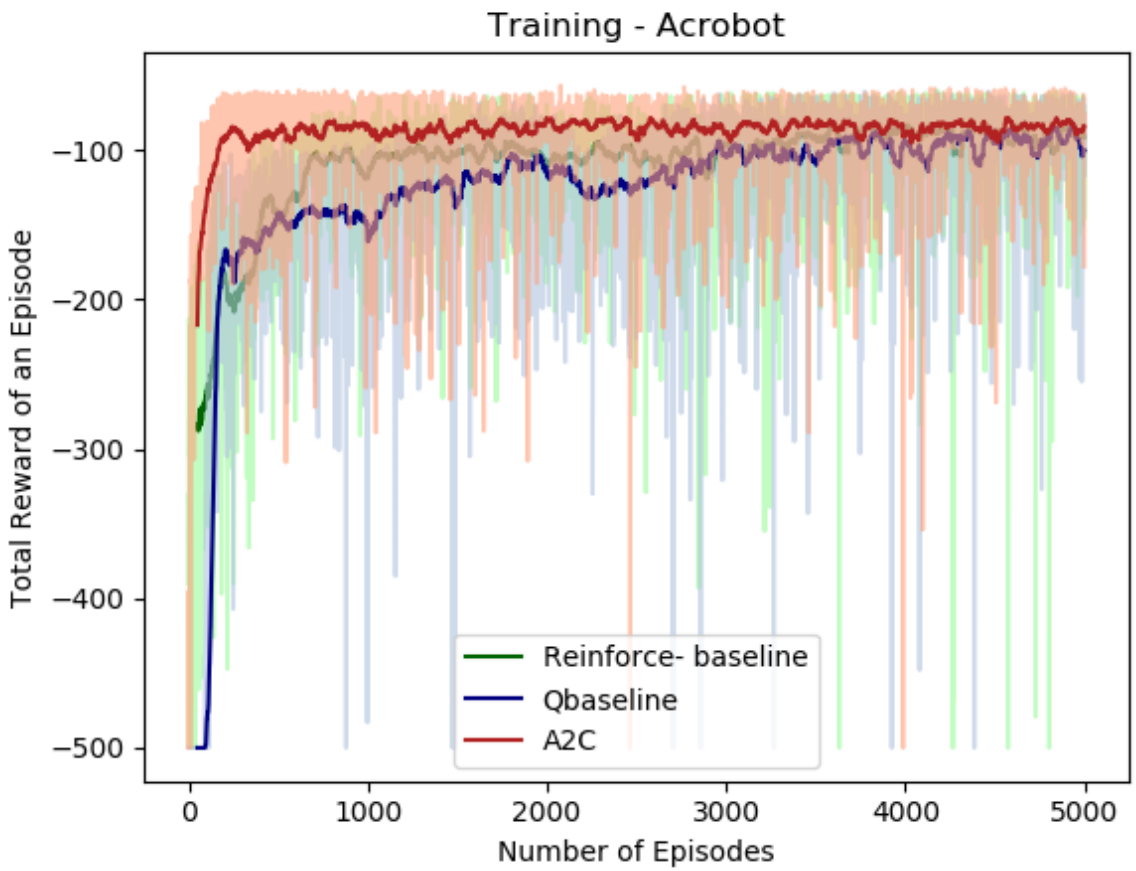


Figure 5: Comparison of few variants on Acrobot environment

Observations

- A2C seems to be faster learner, and also we can say that Value function is better baseline than the average total discounted reward we used in Qbaseline.
- Keep faster learning rate for critic than actor for better learning and convergence
- We can also see the number of parameters used for critic in konda has drastically decreased.
- Requires Large number of episodes to learn the model for an complicated environment

Problems Faced

- Lack of Resources to run compute intensive program, and requirement of high performance GPU
- Choosing the right step-size and network and waiting for longer duration of completion of code, as these methods require lot of episode to learn

Thank You