Statistical Techniques in Robotics (16-831, S22) Lecture #21 (Wednesday, April 6)

Policy Gradient Method and Actor-Critic

Lecturer: Kris Kitani *Scribes: Jia Shi, Mukun Guo*

1 Review

In Last lecture, we talked about model-free value-based control and moved from discrete space to continuous space. On a high level, in discrete cases, we can store the Q-values for each state-action pair, whereas in continuous space such pairs are infinite. Thus, we use a function approximator to map each state-action pair to a value. In this section, we briefly review some important concepts from last lecture.

1.1 Function Approximator for Control

In previous lectures, we have discussed many model-free value-based control algorithms (i.e. Monte Carlo Control, TD Control, On-policy/Off-policy, etc), but we limited our discussion in a discrete setup, where we can essentially store the value function for every state-action pair in a table. However, there are always cases where the state/action space is continuous. In such setup, we will replace the table with a function approximator. There are many choices for such function approximator. The current most popular one in research communities are deep neural networks. Specifically, previously we store the $Q^\pi(s,a)$ value for every state-action pair. In every time step, we update the value itself using some algorithm. Now we will approximate $Q^\pi(s,a)$ using a function $Q_\theta(s,a)$ that is paramterized by θ, and in ever time step we will update θ so that the function better approximate the value function $Q^\pi(s,a)$.

1.1.1 Control algorithms in Continuous space

Monte Carlo Control algorithm in continuous space is very similar to the discrete space, with two modifications.

1. Instead of updating the Q-value for each state-action pair:

$$Q\left(a^{(t)}, s^{(t)}\right) \leftarrow Q\left(a^{(t)}, s^{(t)}\right) + \alpha\left[G^{(t)} - Q\left(a^{(t)}, s^{(t)}\right)\right]$$

   we will instead update the parameters of the approximator function:

$$\theta = \theta + \alpha\left(G^{(t)} - Q_\theta\left(s^{(t)}, a^{(t)}\right)\right)\frac{\partial}{\partial\theta}Q_\theta\left(s^{(t)}, a^{(t)}\right)$$

   Note that in MC Control, $G^{(t)}$ is simply the cumulative return. Its value is independent of the space being continuous or discrete.

2. We also need to add stochasticity into the policy. Specifically, at the beginning of each episode, with probability ϵ, a random policy will be applied where actions will be randomly selected. With probability $(1-\epsilon)$, we apply a greedy policy where we always select the action with highest value given by our approximator function $Q_\theta(s,a)$, that is, $a^* = \text{argmax}_a\, Q_\theta(s,a)$.

For On-policy TD Control (SARSA), similar modifications were made with some slight change.

1. We apply same parameter update as MC Control
2. Instead select policy at the beginning of each episode, the sampling is done at the beginning of every epoch. The sample strategy is the same as MC Control.
3. Since in TD Control, we use an estimator which uses $Q(s,a)$ to estimate $G^{(t)}$, we need to slightly modify how the estimator works. But this is very simple and straightforward. Originally, our one-step TD uses

$$G^{(t)} \leftarrow r^{(t)} + \gamma Q\left(a^{(t+1)}, s^{(t+1)}\right)$$

to estimate the return. We simply replace the Q-value with the approximator function:

$$G^{(t)} \leftarrow r^{(t)} + \gamma Q_\theta\left(a^{(t+1)}, s^{(t+1)}\right)$$

The Off-policy TD Control in continuous space is same as the On-policy version, except for the policy sampling is done on the behavior policy μ.

1.1.2 Framework for Function Approximation for Control

With the above concrete examples, we notice that the modification from discrete to continuous space can be generalize into 3 steps:

1. Apply an approximator function $Q_\theta(s,a) \approx Q^\pi(s,a)$.
2. Set up optimization problem with objective function: $\hat{\boldsymbol{\theta}} = \arg\min_\theta \mathbb{E}_p\left[\left(Q^\pi(s,a) - Q_\theta(s,a)\right)^2\right]$, find the solution using SGD

$$\nabla_\theta \mathcal{L}_\theta \approx -\left(Q^\pi(s,a) - Q_\theta(s,a)\right)\frac{\partial}{\partial\theta}Q_\theta(s,a)$$

$$\theta = \theta - \alpha\nabla_\theta\mathcal{L}_\theta$$

3. Select a return estimator (Monte Carlo, TD, etc) to estimate $Q^\pi(s,a)$ to run the SGD:

$$Q^\pi(s,a) \approx G^{(t)}$$

2 Summary

2.1 Policy Gradient Method

We have concluded out study on Value-based Reinforcement Learning. We will not move into a new domain, Policy-based RL. In policy gradient methods, we skip estimation of the value function and directly learns a function $\pi_\theta(a|s)$ that approximates the best policy. Recall that our ultimate objective is to maximize the return by playing with the parameters of the policy function. Formally, we could set up the following optimization problem:

$$\hat{\theta} = \arg\max_{\theta} \mathbb{E}_{p_\theta(\zeta)} \left[\sum_{t=0}^{T} r^{(t)} \right]$$

where $p_\theta(\zeta)$ is the distribution over trajectories under MDP:

$$\begin{aligned} p_\theta(\zeta) &= p\left(s^{(0)}\right) \Pi_{t=0}^{T} \pi_\theta \left(a^{(t)} \mid s^{(t)}\right) p\left(s^{(t+1)} \mid s^{(t)}, a^{(t)}\right) \\ \zeta &= \left\{ s^{(0)}, a^{(0)}, s^{(1)}, a^{(1)}, \ldots, s^{(T)}, a^{(T)} \right\} \\ r^{(t)} &\triangleq r\left(s^{(t+1)}, a^{(t)}, s^{(t)}\right) \end{aligned}$$

Note that this is a maximization problem. Hence we will apply gradient ascent to solve it. Let's set

$$J(\theta) = \mathbb{E}_{p_\theta(\zeta)} \left[\sum_{t=0}^{T} r^{(t)} \right]$$

Apply first-order Taylor Series approximation with quadratic regularization

$$\hat{\theta} = \arg\max_{\theta} \left\{ \alpha \left(J\left(\theta'\right) + \left\langle \theta - \theta', \nabla_{\theta'} J\left(\theta'\right) \right\rangle \right) - \frac{1}{2} \left\| \theta - \theta' \right\|^2 \right\}$$

Now let's try to solve this new optimization problem using closed-form:

$$\begin{aligned} \nabla_\theta \left\{ \alpha \left(J\left(\theta'\right) + \left\langle \theta - \theta', \nabla_{\theta'} J\left(\theta'\right) \right\rangle \right) - \frac{1}{2} \left\| \theta - \theta' \right\|^2 \right\} &= 0 \\ \alpha \nabla_{\theta'} J\left(\theta'\right) - \theta + \theta' &= 0 \end{aligned}$$

Rearrange the terms we could get the following gradient update rule:

$$\theta \leftarrow \theta' + \alpha \nabla_{\theta'} J\left(\theta'\right)$$

Now Let's take a look at that's inside the gradient term $\nabla_{\theta'} J\left(\theta'\right)$. Recall that

$$J(\theta) = \mathbb{E}_{p_\theta(\zeta)} \left[\sum_{t=0}^{T} r^{(t)} \right] = \int_{\zeta} p_\theta(\zeta) r(\zeta) d\zeta$$

where $r(\zeta) = \sum_{t=0}^{T} r^{(t)}$. Take its derivative w.r.t. θ and apply some basic derivative rules:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \nabla_\theta \int_\theta p_\theta(\zeta) r(\zeta) d\zeta \\
&= \int_\theta \nabla_\theta p_\theta(\zeta) r(\zeta) d\zeta \\
&= \int_\theta p_\theta(\zeta) \frac{\nabla_\theta p_\theta(\zeta)}{p_\theta(\zeta)} r(\zeta) d\zeta \\
&= \int_\theta p_\theta(\zeta) \nabla_\theta \ln p_\theta(\zeta) r(\zeta) d\zeta
\end{aligned}$$

Recall that a trajectory is simply a combination of state-actions pairs modeled by the MDP, that is,

$$p(\zeta) = p(s^{(0)}) \prod_t \left[\pi_\theta(a^{(t)}|s^{(t)}) \cdot p(s^{(t+1)}|s^{(t)}, a^{(t)}) \right]$$

Since we are in the log space, this becomes:

$$\ln p(\zeta) = \ln p(s^{(0)}) + \sum_t \left[\ln \pi_\theta(a^{(t)}|s^{(t)}) + \ln p(s^{(t+1)}|s^{(t)}, a^{(t)}) \right]$$

Thus, we have:

$$\nabla_\theta \ln p_\theta(\zeta) = \nabla_\theta \left[\ln p\left(s^{(0)}\right) + \sum_{t=1}^{T} \ln \pi_\theta\left(a^{(t)} \mid s^{(t)}\right) + \ln p\left(s^{(t+1)} \mid s^{(t)}, a^{(t)}\right) \right]$$

Note that only the second term (i.e. policy) depend on θ,

$$\nabla_\theta \ln p_\theta(\zeta) = \nabla_\theta \left[\sum_{t=1}^{T} \ln \pi_\theta\left(a^{(t)} \mid s^{(t)}\right) \right]$$

Plug it into the gradient term, we have:

$$\begin{aligned}
\nabla_\theta J(\theta) &= \int_\theta p_\theta(\zeta) \nabla_\theta \left[\sum_{t=1}^{T} \ln \pi_\theta\left(a^{(t)} \mid s^{(t)}\right) \right] r(\zeta) d\zeta \\
&= \mathbb{E}_{p_\theta} \left[\left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta\left(a^{(t)} \mid s^{(t)}\right) \right) r(\zeta) \right]
\end{aligned}$$

Note that above is an theoretical expectation term and in reality is very hard to obtain. Hence we replace the expectation with a simple average over trajectories (i.e. **Monte-Carlo Estimation**):

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{n=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta\left(a^{(t)} \mid s^{(t)}\right) \right) r(\zeta) \right]$$

Now let's try to understand this gradient term part by part. $r(\zeta) = \left(\sum_{t=1}^{T} r^{(t)} \right)$ is nothing more than the cumulative return of an episode (i.e. **Episodic Return**). The higher the return, the larger the gradient term. This essentially indicates how good the current policy is. If it's a good policy, we will have high return and update the policy function with larger gradient value in this direction.

If the policy is bad, $r(\zeta)$ will be small and we make little update. Furthermore, We claim that $\sum_{t=1}^{T} \nabla_\theta \ln \pi_\theta \left(a^{(t)} \mid s^{(t)}\right)$, the **Eligibility Vector**, is an unbiased estimator of the policy function, because $\nabla_\theta \ln \pi_\theta \left(a^{(t)} \mid s^{(t)}\right) = \frac{\nabla_\theta \pi_\theta \left(a^{(t)} | s^{(t)}\right)}{\pi_\theta \left(a^{(t)} | s^{(t)}\right)}$ where the denominator is an inverse weighting. If some actions have lower probability, we will apply a larger weights as an offset and vice versa. This ensures for both lower-probability action and higher-probability action we have appropriate gradient values.

2.2 Monte-Carlo Policy Gradient

Now that we have finished all the pieces of gradient ascent, we can put together our first policy-based reinforcement learning algorithm.

Algorithm 1 function MC-POLICY-GRADIENT (π_θ, α)

```
for e = 1, ..., E do
    {s^(t), a^(t), r^(t)}_{t=0}^T ~ E | π_θ
    G^(0) = Σ_{t=0}^T r^(t)
    for t = 0, ..., T do
        θ ← θ + α (∇_θ log π_θ (a^(t)|s^(t))) (G^(0))
    end for
end for
return π_θ
```

2.3 Episodic Reinforcement Algorithm

Just like other Monte-Carlo algorithms, this policy gradient algorithm also suffers from high variance. Two tricks were proposed to tackle this issue.

2.3.1 Enforce Causality

Since the high variance comes from how return is estimated. We slightly modify the estimation such that the policy's decision at time t cannot affect past rewards. Formally,

$$G^{(0)} = \sum_{t=0}^{T} r^{(n,t)}$$

$$\downarrow$$

$$G^{(0)} = \sum_{t=t'}^{T} r^{(n,t)}$$

Before, whenever we want to update the policy, we considers how this update affects past, now, and future. With this slight tweak, out policy update now only takes into consideration of how current and future rewards change.

2.4 Remove Gradient Bias with a baseline offset

We now that high variance is partly cased by the large numerical value of the return. There is an even easier way to remove the high variance. We could simply subtract the return by some offset b that does not depend on θ. Formally,

$$\nabla_\theta J(\theta) = \int_\theta p_\theta(\zeta) \nabla_\theta \ln p_\theta(\zeta)[r(\zeta) - b] d\zeta = \int_\theta p_\theta(\zeta) \nabla_\theta \ln p_\theta(\zeta) r(\zeta) d\zeta$$

The proof that subtracting b does not affect gradient will be elaborated in Appendix. With this trick, we are ready to state the classic reinforcement algorithm!

Algorithm 2 function Episodic-Reinforce (π_θ, α)

for $e = 1, \ldots, E$ **do**
  $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi_\theta$
  $G^{(0)} = \sum_{t=0}^{T} r^{(t)}$
  for $t = 0, \ldots, T$ **do**
    $\theta \leftarrow \theta + \alpha \left(\nabla_\theta \log \pi_\theta \left(a^{(t)} | s^{(t)}\right)\right) \left(G^{(0)} - b\right)$
  end for
end for
return π_θ

If we put the two tricks together, we get the following policy gradient algorithm With smaller

Algorithm 3 function POLICY-GRADIENT (π_θ, α)

for $e = 1, \ldots, E$ **do**
  $\left\{s^{(t)}, a^{(t)}, r^{(t)}\right\}_{t=0}^{T} \sim \mathcal{E} \mid \pi_\theta$
  for $t = 0, \ldots, T$ **do**
    $G^{(0)} = \sum_{i=t}^{T} r^{(i)}$
    $\theta \leftarrow \theta + \alpha \left(\nabla_\theta \log \pi_\theta \left(a^{(t)} | s^{(t)}\right)\right) \left(G^{(0)}\right)$
  end for
end for
return π_θ

variance, the gradient ascent converges faster and we can achieve good results with fewer sampled trajectories and episodes.

2.5 Pros and Cons of Policy Gradient Methods

Pros:

1. Doesn't require model, learn from pure interaction with environment.
2. Effective in high-dimensional or continuous action space.
3. With specifically designed policy function, we can encode prior knowledge.

4. Find the optimal stochastic policy and naturally explores due to inherent stochasticity.

Cons:

1. Even with proposed tricks, gradient is still high variance and convergence is slow.
2. Due to the nature of gradient ascent, smaller step size will lead to slow convergence. And we will probably converge in local maximum instead of global maximum. If we use a deep neural network at the policy function, we will also suffer from vanishing and exploding gradient.

2.6 Actor-Critic

Actor-Critic is the hybrid of both policy base and value base method by calculating the optimal policy and value function at the same time. In the naming 'Actor-Critic', actor refers to the policy and critic refers to the reward function. It's the most popular RL algorithm.

In the policy gradient method, we need to run through the entire episode to calculate the return, as noted in term $\sum_{t=1}^{T} r^{(t)}$. Actor critic method provide a way to spare this effect by replacing the return with return estimator. Formally,

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}\left[\left(\sum_{t=1}^{T} \nabla_\theta \ln_{\pi_\theta}(a^{(t)}|s^{(t)})\right)\left(\sum_{t=1}^{T} r^{(t)}\right)\right]$$

$$\downarrow$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta}\left[\left(\sum_{t=1}^{T} \nabla_\theta \ln_{\pi_\theta}(a^{(t)}|s^{(t)})\right)\left(G_\phi^{(t)}\right)\right]$$

The $G_\phi^{(t)}$ is the function approximator, we can replace it with other method we have learn as list in Tab.2.6 to form other actor critic function. For instance, Algorithm 4 shows where a Monte Carlo is applied as the function estimate, and Algorithm 5 shows the case where a TD-function is applied, which line 5 is replaced with a one step return approximation of the Q function.

Algorithm 4 Q-MC-ACTOR-CRITIC($\pi_\theta, Q_\phi, \alpha, \beta$)

1: **for** $e = 1, ..., E$ **do**
2: $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi_\theta$
3: **for** $t = 0, ..., T$ **do**
4: $G^{(t)} \leftarrow \sum_{i=t}^{T} r^{(i)}$
5: $\phi \leftarrow \phi + \beta[G^{(t)} - Q_\phi(a^{(t)}, s^{(t)})]\nabla Q_\phi(a^{(t)}, s^{(t)})$
6: $\theta \leftarrow \theta + \alpha\nabla \log \pi_\theta(a^{(t)}|s^{(t)}) \cdot Q_\phi(a^{(t)}, s^{(t)})$
7: **end for**
8: **end for**
9: **return** π_θ

Method	Return estimator G^t
MC	$G^{(t)} = \sum_{i=t}^{T} r^{(i)}$
TD(0)	$G_0^{(t)} = r^t + \gamma V(s^{(t+1)})$
N-step TD	$G^{(t)}(N) = \sum_{i=t}^{t+N-1} \gamma^{i-t} r^{(i)} + \gamma^N V(s^{(t+N)})$
$\lambda - Return$	$G_\lambda^{(t)} = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G^{(t)}(n)$
$TD(\lambda)$	$G_z^{(t)} = z^{(t)}(s) \left[r^{(t)} + \gamma V(s^{(t+1)}) \right]$
IS-MC	$G_{\pi/\mu}^{(t)} = \prod_{i=t}^{T} \frac{\pi(a^{(i)}\|s^{(i)})}{\mu(a^{(i)}\|s^{(i)})} \sum_{j=t}^{T} r^{(j)}$
IS-TD(0)	$G_{\pi/\mu}^{(t)}(1) = \frac{\pi(a^{(i)}\|s^{(i)})}{\mu(a^{(i)}\|s^{(i)})} \left[r^{(t)} + \gamma V(s^{(t+1)}) \right]$

Table 1: Prediction methods to estimate return covered in class

Algorithm 5 Q-TD-ACTOR-CRITIC($\pi_\theta, Q_\phi, \alpha, \beta$)

1: **for** $e = 1, ..., E$ **do**
2: $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi_\theta$
3: **for** $t = 0, ..., T$ **do**
4: $\delta \leftarrow r^{(t)} + Q_\phi(a^{(t+1)}, s^{(t+1)})$
5: $\phi \leftarrow \phi + \beta[\delta - Q_\phi(a^{(t)}, s^{(t)})] \nabla Q_\phi(a^{(t)}, s^{(t)})$
6: $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a^{(t)}|s^{(t)}) \cdot Q_\phi(a^{(t)}, s^{(t)})$
7: **end for**
8: **end for**
9: **return** π_θ

For the baseline offset term B in the gradient function $\nabla_\theta J(\theta)$, the optimal B offset we should use is the value function $V_\phi(s^{(t)})$ for which we can derive the optimal adjusted return, also known as the Advantage function $A(a^{(t)}, s^{(t)})$, which is the difference between the policy gradient Q and the value function, formally,

$$A(a^{(t)}, s^{(t)}) = Q_\phi(a^{(t)}, s^{(t)}) - V_\phi(s^{(t)})$$

Thus we have the full form of policy gradient with advantage function

$$\nabla_\theta J(\theta) = \mathbb{E}_{p_\theta} \left[\left(\sum_{t=1}^{T} \nabla_\theta \ln_{\pi_\theta}(a^{(t)}|s^{(t)}) \right) \left(\sum_{i=t}^{T} A_\phi(a, s) \right) \right]$$

Algorithm 6 give an example of advantage actor critic function with Monte Carlo estimator. In line the MC estimator is obtained, and the parameter of value function, Q value function and the policy is updated with gradient decent algorithm in line 6,7,8 respectively.

Algorithm 6 ADVANTAGE-MC-ACTOR-CRITIC($\pi_\theta, Q_\phi, V_\psi, \alpha, \beta, \kappa$)

for $e = 1, ..., E$ **do**
  $\{s^{(t)}, a^{(t)}, r^{(t)}\}_{t=0}^{T} \sim \mathcal{E}|\pi$
  for $t = 0, ..., T$ **do**
    $G^{(t)} \leftarrow \sum_{i=t}^{T} r^{(i)}$
    $\psi \leftarrow \psi + \kappa[G^{(t)} - V_\phi(s^{(t)})]\nabla_\psi V_\psi(s^{(t)})$
    $\phi \leftarrow \phi + \beta[G^{(t)} - Q_\phi(a^{(t)}, s^{(t)})]\nabla_\phi Q_\phi(a^{(t)}, s^{(t)})$
    $\theta \leftarrow \theta + \alpha[Q_\phi(a^{(t)}, s^{(t)}) - V_\psi(s^{(t)})]\nabla_\theta \log \pi_\theta(a^{(t)}|s^{(t)})$
  end for
end for
return π_θ

References

[1] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

3 Appendix

3.1 Proof of baseline offset not affect gradient

$$
\begin{aligned}
\nabla_\theta J(\theta) &= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)[r(\zeta) - b]d\zeta \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta - \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)bd\zeta \quad \text{evaluate derivative} \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta - \int_\zeta p_\theta(\zeta)\frac{\nabla_\theta p_\theta(\zeta)}{p_\theta(\zeta)}bd\zeta \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta - \int_\zeta \nabla_\theta p_\theta(\zeta)bd\zeta \quad \text{assume } b \text{ not depend on } \theta \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta - b\nabla_\theta \int_\zeta p_\theta(\zeta)d\zeta \quad \text{probability sums up to one} \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta - b\nabla_\theta(1) \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta - 0 \\
&= \int_\zeta p_\theta(\zeta)\nabla_\theta \ln p_\theta(\zeta)r(\zeta)d\zeta
\end{aligned}
$$

which is exactly the version without baseline offset.

3.2 Effect of step size and baseline offset

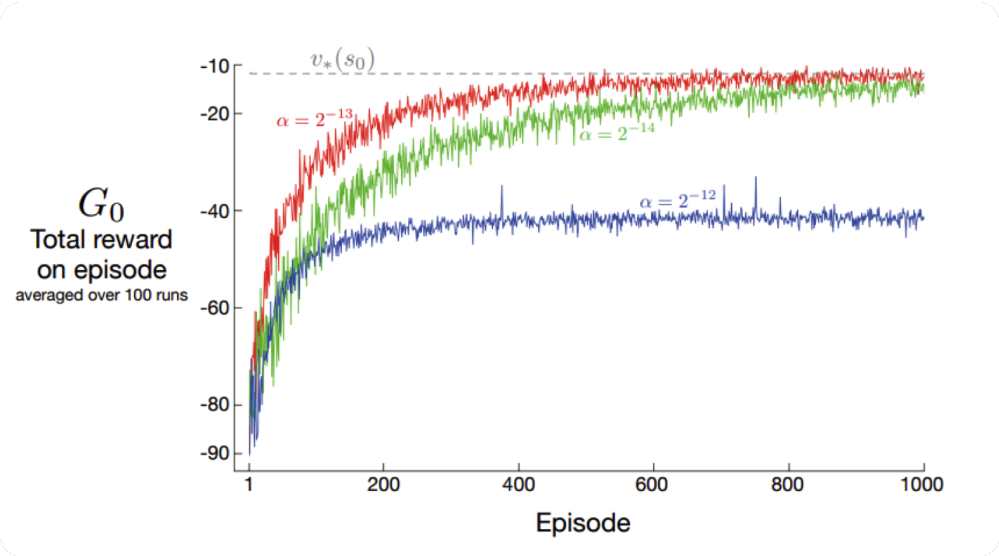

Figure 1: Step size on convergence time, figure credit to [1]. As seen, an appropriate step size help gradient ascent converge at a good maximum. Too large or too small step size both lead to non-optimal result.

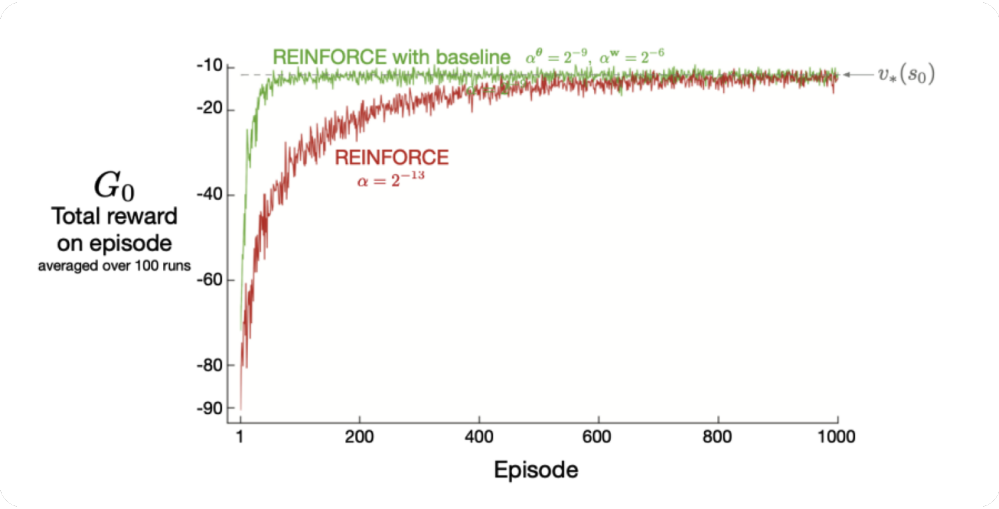

Figure 2: Baseline offset on convergence time, figure credit to [1]. With the baseline offset trick, the gradient ascent converges much faster.