University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 25: November 27,29

Lecturer: Alexander Schwing & Safa Messaoud Scribe: Wenhan Zhao

1 Introduction

The reinforcement learning techniques we learned so far can be divided into two main groups. The first group of method is that the Markov Decision Process (e.g. The transition probabilities) is known to us, in such cases, we can simply do exhaustive search over the whole policy space, and find the policy with the highest expected future reward. We can also find the best policy using policy iteration, which is quite similar to gradient descend, or using value iteration to find a best value function and decode the best policy from that best value function. When the MDP is unknown to us, we can estimate the transition probabilities using experience replay or use Q-learning algorithms in order to determine the best policy.

Besides such methods, we can also directly optimize the parametric policy, and this is what we called policy gradient based method. This lecture first reviews previous reinforcement learning algorithms and then dive into detailed algorithm of Policy Gradient and its relationship with other reinforcement learning methods, finally introduce some related applications.

2 Recap

2.1 Methods with known MDP

Exhausted Search

Exhausted Search is a simple method which directly iterate through every possible policy $\pi(s)$, and choose the policy π^* which has the largest expected future reward $V^\pi(s)$. $V^\pi(s)$ can be calculated using policy evaluation which requires solving linear system as follows.

$$V^\pi(s) = 0 \; if \; s \in G \tag{1}$$

$$V^\pi(s) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^\pi(s')] \tag{2}$$

However, using exhausted search can be computationally expensive since the total number of policies that need to be computed is very high.

Policy Iteration

Policy iteration is another method which search over policies. However, instead of searching over the entire policy space, we start at a random policy π and then refine it every time after we calculate the expected future reward value, until π converged to a value which is no longer changing.
The basic steps are like follows:

- 1. Initialize a policy π

- 2. Compute the reward value by solving linear function.

$$V^{\pi}(s) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{3}$$

- 3. Get a improved policy using

$$\pi' = \arg\max_{a \in A_s} \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{4}$$

  and then repeat 2 if the improved policy is still changing in this process.

Value Iteration

Value iteration is another method to find the best policy. Unlike methods searching on policies, value iteration focus on searching on the value space, finding the optimal value function V^*. After getting a optimal value function, the best policy can be decoded from the optimal value function as follows:

$$V^*(s) = \max_{a \in A_s} Q(s, a) \tag{5}$$

$$\pi^*(s) = \arg\max_{a \in A_s} Q(s, a) \tag{6}$$

Where

$$Q(s, a) = \sum_{s' \in S} P(s'|s, \pi(s))[R(s, \pi(s), s') + V^{\pi}(s')] \tag{7}$$

2.2 Methods with unknown MDP

If the transition probability is not known to us, we need to estimate the transition probabilities using experience replay. By running a simulator enough rounds to generate large amount of samples (s, a, r, s'), we can get an approximation of the transition probabilities for this system.

However, even if we know the transition probabilities, if there is infinity possible states in the model, we still cannot solve the problem in finite amount of time. Therefore, we introduced Q-Learning to solve this kind of problem.

Q-Learning

The goal of Q-Learning is quite similar to previous methods, which is to let our agent learn a policy telling is what action to take under each state. Q-Learning only uses a batch of (s, a, r, s') got from the experience replay. The algorithm replays the following until Q-value's convergence:

- 1. Get a sample transition tuple (s, a, r, s').
- 2. Calculate a target Q-value.

$$Q_{target} \approx r + \max_{a \in A_{s'}} Q(s', a). \tag{8}$$

- 3. Update the estimated Q-Value.

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha Q(s', a) \tag{9}$$

- 4. Switch s to s'.

At each state, the agent will choose the action that has a largest Q-value, which is a optimal learned policy as a result of Q-Learning.

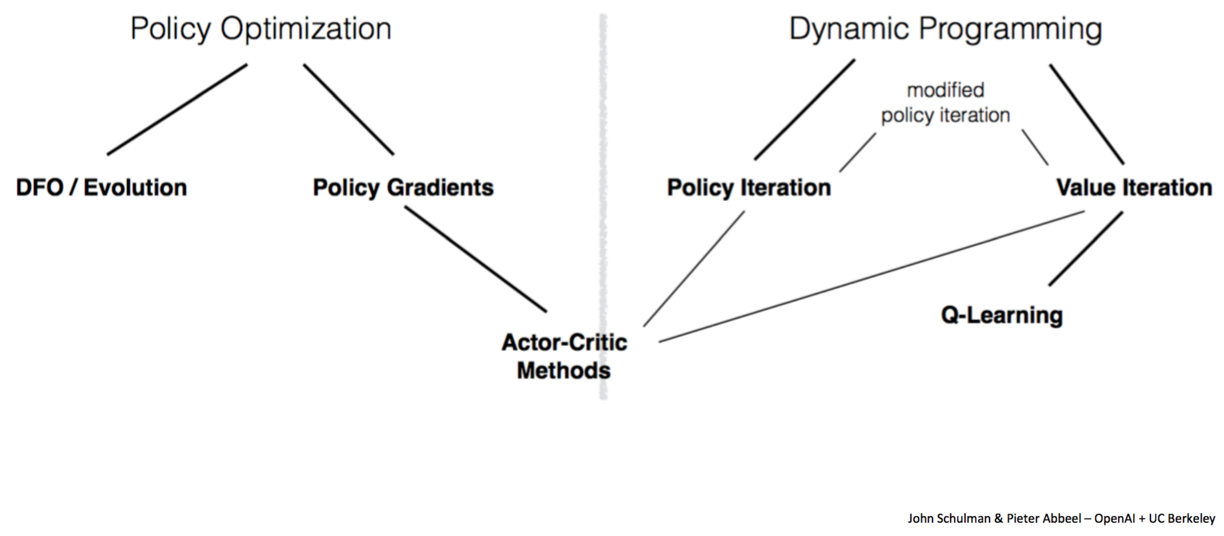

Figure 1: Techniques used in reinforcement learning.

3 Policy Gradient

3.1 Introduction

Previously, our policy is a deterministic function $\pi(s)$, where given a certain state s, the agent have a certain action to perform. However, if we need to directly optimize on the policy, we need to make the policy parametric as $\pi_\theta(a|s)$, representing under a certain parameter(environment) θ, the probability of performing action a in state s. By doing this, we make the policy π a function related to a parameter θ.

Reasons for optimizing π are : The policy function π may be simpler than Q and V; The V-value does not prescribe action, and it's often not used when the transition probabilities are not known; The Q-value will require efficient maximization, so it involves issues in continuous and high-dimensional action spaces.

Figure 1 shows the techniques used in reinforcement learning and their relationship.

3.2 Method for Policy Gradient

In this case, how can we find the best policy when the transition probability is still not known? The idea is we can directly maximize the future reward using gradient descent.

Policy gradient method will take Rollouts as input, which is a sequence of state and action.

$$\tau = (s_0, a_0, s_1, a_1, ...) \tag{10}$$

The expected reward of a particular state-action sequence is obvious, just computing the sum of reward during each state transition and we get the total reward.

$$R(\tau) = \sum_t R(s_t, a_t) \tag{11}$$

However, if we want to calculate the expected future reward just given a start state, we need to involve the policy π_θ in our equation, that's to say, the probability of a particular state-action sequence *tau* occurs depends on the parameter θ, therefore, we have:

$$U(\theta) = E[\sum_t R(s_t, a_t); \pi_\theta] = \sum_\tau P(\tau; \theta) R(\tau) \tag{12}$$

Our goal then is to maximize this U(θ).

Need to notice that $U(\theta)$ is not the same as Q(s,a) in Q-Learning although they both indicates the expected future reward. Q(s,a) depends on the state and actions, it represents the reward given a particular state and action. $U(\theta)$ however, just assumes you start from some start state, and it tells you the overall expectation of future reward.

So how to find the maximum of $U(\theta)$? We will use the gradient descent to do that.

$$\begin{aligned} \bigtriangledown_\theta U(\theta) &= \bigtriangledown_\theta \sum_\tau P(\tau; \theta) R(\tau) \\ &= \sum_\tau \bigtriangledown_\theta P(\tau; \theta) R(\tau) \end{aligned} \tag{13}$$

Since $P(\tau; \theta)$ is something we can't compute, we need to estimate it via sampling. Then we have:

$$\bigtriangledown_\theta U(\theta) = \frac{1}{N} \bigtriangledown_\theta \sum_{\tau \sim \pi_\theta} R(\tau) \tag{14}$$

Where N is the total number of rollouts.

We found that θ disappeared in the cost function, where we are going to compute the gradient, and the computation seems impossible to proceed.

Here we can use a small trick to get out of this:

$$\begin{aligned} \bigtriangledown_\theta U(\theta) &= \sum_\tau \bigtriangledown_\theta P(\tau; \theta) R(\tau) \\ &= \sum_\tau \frac{P(\tau; \theta)}{P(\tau; \theta)} \bigtriangledown_\theta P(\tau; \theta) R(\tau) \\ &= \sum_\tau P(\tau; \theta) \frac{\bigtriangledown_\theta P(\tau; \theta)}{P(\tau; \theta)} R(\tau) \\ &= \sum_\tau P(\tau; \theta) \bigtriangledown_\theta \, log P(\tau; \theta) R(\tau) \end{aligned} \tag{15}$$

Now the approximate gradient with empirical estimate for sample paths under policy π_θ is:

$$\bigtriangledown_\theta U(\theta) \approx \hat{g} = \frac{1}{m} \sum_{i=1}^{m} \bigtriangledown_\theta log P(\tau^{(i)}; \theta) R(\tau^{(i)}) \tag{16}$$

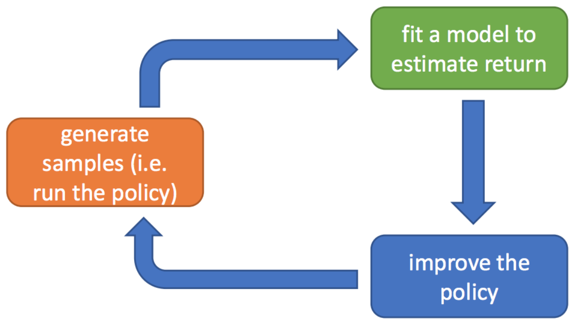
fit a model to estimate return
improve the policy
generate samples (i.e. run the policy)

Figure 2: Basic steps to improve policy using policy gradient [1]

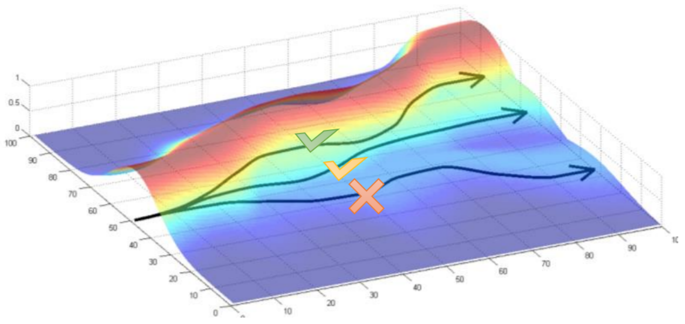

Figure 3: Evaluating the policy gradient [1]

which is a function related to θ. What the algorithm does is [1] :

- 1. Sample state-action sequences τ^i using the current parametric policy $\pi_\theta(a|s)$.
- 2. Compute $\bigtriangledown_\theta U(\theta)$
- 3. Improve θ : $\theta \leftarrow \alpha \bigtriangledown_\theta U(\theta) + \theta$, then repeat step 1.

What we are doing is trying to increase the possibility of rollout that have a positive reward to occur , and decrease the possibility of rollout that have a negative reward to occur. We achieve this adjustment by updating θ every turn. Figure 2 and Figure 3 helps to demonstrate this intuition.

3.3 Important properties

Valid for discontinuous reward function

From equation (16), we see clearly that the gradient is only computed on $logP(\tau^{(i)};\theta)$ since $R(\tau)$ is not related to θ. This makes a very good property : Even when R is something indifferentiable, we can still calculate the gradient of the cost function.

No need for dynamics model

Further simplify equation (16) then we can see $\bigtriangledown_\theta U(\theta)$ is totally not relevant to transition probabilities. Since

$$\begin{aligned}\triangledown_\theta logP(\tau;\theta) &= \triangledown_\theta log[\prod_t P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)] \\ &= \triangledown_\theta[\sum_t logP(s_{t+1}|s_t,a_t) + \sum_t log\pi_\theta(a_t|s_t)]\end{aligned} \tag{17}$$

Note that $\sum_t logP(s_{t+1}|s_t,a_t)$ in (17) is not related to θ and thus becomes zero when calculating gradient on it over θ. Then we have:

$$\begin{aligned}\triangledown_\theta logP(\tau;\theta) &= \triangledown_\theta \sum_t log\pi_\theta(a_t|s_t) \\ &= \sum_t \triangledown_\theta log\pi_\theta(a_t|s_t)\end{aligned} \tag{18}$$

Which is a dynamics model free equation.
So we modified the expression of $\triangledown_\theta U(\theta)$ from equation (16) to

$$\triangledown_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \triangledown_\theta log\pi_\theta(a_t^{(i)}|s_t^{(i)}))R(\tau^{(i)}) \tag{19}$$

3.4 Practically important methods

Two methods are practically important when implementing a real policy gradient algorithm. The first one is reward normalization, and another one is temporal structure.

Baseline

Sometimes all the rollouts will give us a positive feedback or all of them give a negative feedback. This causes the learning process increasing or decreasing the probability for all occurred state-action sequences, result in a very slow convergence.

To solve this problem, we can add or subtract a baseline value b to the reward function when we calculate the gradient of $U(\theta)$. For example, if all $R(\tau^{(i)})$ are positive, we can fix the issue with:

$$\triangledown_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \triangledown_\theta logP(\tau^{(i)})(R(\tau^{(i)}) - b) \tag{20}$$

This will make some reward values become positive and others become negative, result in a fast convergence.
Why is subtraction of baseline b okay ? Consider the following equation :

$$\begin{aligned}E[\triangledown_\theta P(\tau;\theta)b] &= \sum_\tau P(\tau;\theta)\ \triangledown_\theta\ logP(\tau;\theta)b \\ &= \sum_\tau \triangledown_\theta P(\tau;\theta)b \\ &= \triangledown_\theta(\sum_\tau P(\tau;\theta))b \\ &= \triangledown_\theta b \\ &= 0\end{aligned} \tag{21}$$

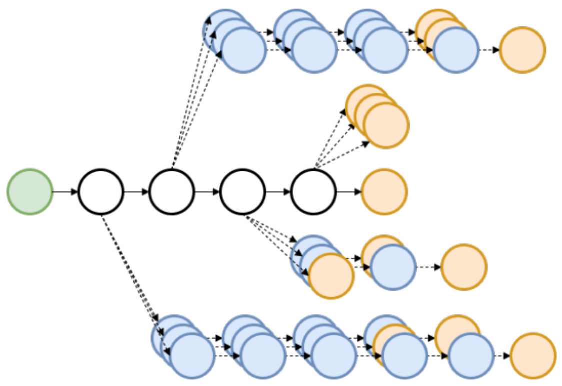

Figure 4: Sampling a caption [2]

It's obvious from (21) the newly introduced baseline will not contribute to the gradient of $U(\theta)$.

The choice of the baseline value b can be varied, one example choice is to use the expectation of $R(\tau)$, which is actually the mean value of all occurred $R(\tau)$:

$$b = E[R(\tau)] = \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)}) \tag{22}$$

Notice the baseline from equation (22) is an adaptive one, every time we utilize a new τ b will change accordingly.

Temporal Structure

Temporal structure has similar idea with the baseline method. Since future reward don't depend on past rewards, we can change the baseline value to be the expected reward after the current turn. So equation (20) can be modified to:

$$\bigtriangledown_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \bigtriangledown_\theta log P(\tau^{(i)})(R(\tau^{(i)}) - b(s_{\hat{t}}^{(i)})) \tag{23}$$

Where

$$b(s_{\hat{t}}) = E[r_t + r_{t+1} + ...] \tag{24}$$

3.5 Related Applications

In [2], researchers optimized their previous method for image captioning using policy gradient. Figure 4 shows the sampling process. The value of each action is estimated as the average rewards received by its K rollout sequences (i.e. K = 3). Solid arrows indicate the sequence of actions being evaluated. The tokens in green and yellow are respectively BOS (beginning of sequence) and EOS (end of sequence) tokens. Sequences in blue are rollout sequences sampled from partial sequences. Note that rollout sequences do not always have the same length, as they are separately sampled from a stochastic policy [2].

The network architecture of the system described in the paper is shown as Figure 5, it consists a encoding CNN layer and several decoding RNN layers, where the trajectory sequences are generated.

Figure 5 describes the model architecture of Show and Tell image captioning system. The tokens in green and yellow are respectively BOS (beginning of sequence) and EOS (end of sequence) tokens.

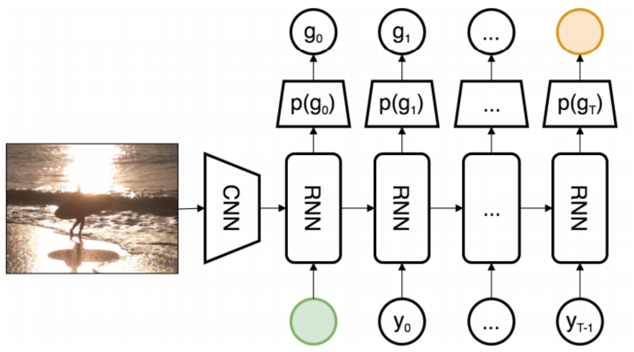

Figure 5: Sampling a caption [2]

4 Quiz Questions

- 1. Why do we need Policy Gradient?

  There are several reasons for it, as mentioned before, directly optimize the policy function π_θ may be simpler than doing optimization of the V-value or Q-value, especially when the MDP (transition probabilities) are not known to us.

  Furthermore, we saw a bunch of good properties when computing the gradient of the expected future reward, one good property is that we don't need dynamic model when polishing our policy since the transition probabilities don't include our parameter θ and thus become zero then eliminated when computing gradient. Another good property is we don't need a differentiable reward function for policy gradient, because the reward function doesn't involves θ either, so it will not be differentiated during computation.

- Techniques to improve vanilla Policy Gradient ?

  As mentioned before, baseline and temporal structures can improve the Policy Gradient algorithm. When all the rollouts will give us a positive feedback or all of them give a negative feedback, the learning process increasing or decreasing the probability for all occurred state-action sequences, this will probably make the system run forever and our policy will not converge . Reward normalization methods like baseline and temporal structure fixed this problem, and let the learning process converge quickly.

References

[1] S. Levine. *CS294-112 Deep Reinforcement Learning, University of California, Berkeley.* 2017 Fall.

[2] N. Y. S. G. K. M. Siqi Liu, Zhenhai Zhu. Improved image captioning via policy gradient optimization of spider. *ICCV*, 2017.