University of Illinois Fall 2018

ECE 544NA: Pattern Recognition
Lecture 25: November 27

Lecturer: Alexander Schwing Scribe: Tianqi Liu (tliu51)

1 Recap

1.1 MDP in Reinforcement Learning

Reinforcement learning is a technique to solve the decision making problem. It is unsupervised learning and use a reward signal to evaluate the decision. Markov Decision Process (MDP) is used to model the reinforcement learning environment. The MDP is Markov, which means given the current state, the future and the past are independent. A reinforcement learning problem described in MDP typically consists of:

- A set of states S
- A set of actions A_s
- A transition probability $P(s'|s,a)$
- A reward function $R(s)$
- A start and terminate state (Maybe)

The goal is to choose the action according to a policy π that could maximize the expected future reward. If the transition probability is known, the problem can be solved much easier. Using Bellman optimality principle, we got:

$$V^*(s) = \max_{a\in A_s} \sum_{s'\in S} P(s'|s,a)[R(s,a,s') + V^*(s')] \tag{1}$$

$$Q^*(s,a) = \sum_{s'\in S} P(s'|s,a)[R(s,a,s') + \max_{a'\in A_{s'}} Q^*(s',a')] \tag{2}$$

$$\pi^*(s) = \arg\max_{a\in A_S} Q^*(s,a) \tag{3}$$

We have three methods to find the optimal policy π^*:

- Exhaustive search
- Value iteration
- Policy iteration

1.2 Q Learning

The Bellman Optimality Principle can be used to iteratively search for the optimal reward given current state s and action a, as shown above. However, when the transition probability, or even the model is unknown, then we need to estimate transition probabilities using experimental samples. Q learning, a value iteration method, is introduced in such scenarios:

- First, we obtain a transition (s,a,r,a').
- Obtain sample suggest:

$$Q(s,a) \approx y_{(s,a,r,s')} = r + \max_{a' \in A_{s'}} Q(s',a') \tag{4}$$

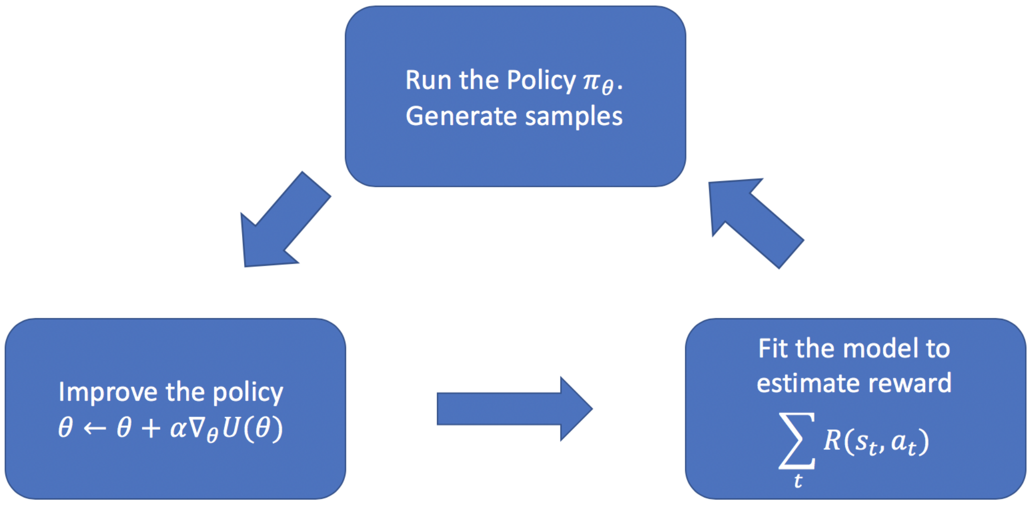
- Keep updating Q. Initialize Q(s,a) randomly in the beginning.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha y_{(s,a,r,s')} \tag{5}$$

2 Policy Gradient

Q learning is a value based method, which means it learns a value function that can estimate the expected sum of rewards Q^* given a state s and an action a. A different approach would be directly optimize the parametric policy $\pi_\theta(a|s)$, a function that maps state to action. Suppose the policy is parametrized by parameter θ. Then the task became searching for optimal parameter θ that maximize the utility function:

$$U(\theta) = \mathbb{E}[\sum_t R(s_t, a_t; \pi_\theta)] \tag{6}$$



Run the Policy π_θ.
Generate samples
Improve the policy
$\theta \leftarrow \theta + \alpha \nabla_\theta U(\theta)$
Fit the model to estimate reward
$\sum_t R(s_t, a_t)$

Figure 1: Policy Gradient Work Flow

The initiative behind is that policy π is easier to get compared to value $V(s)$ or reward $Q(s,a)$. Value $V(s)$ doesn't prescribe actions. Therefore a dynamic model, which is much harder to train, is required. And in the Bellman optimization process, previous results must be backed up.

To calculate reward $Q(s,a)$ requires efficient optimization because Q learning would assign the maximum expected future reward on each possible tuple (s,a) at any time. Thus issues would arise in continuous or high dimensional action space. On the other side, policy gradient are relatively effective in high dimensional or continuous action space.

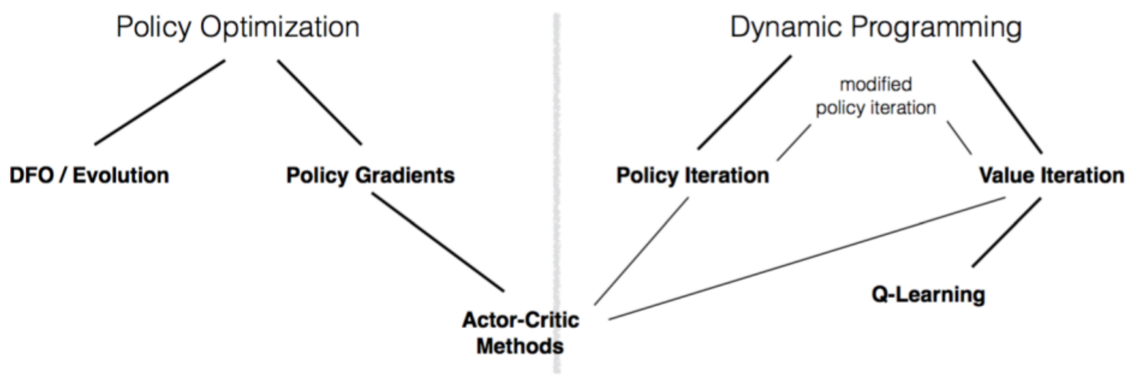


Figure 2: Landscape for Policy Gradient [2]

Another advantage of policy gradient is its convergence property. It has a smooth change at each step. Since we use gradient descent to find the best parametric policy, the converge to a local or global maximum is guaranteed. Meanwhile, the value based methods could potentially oscillate in the training process since the choice of action might significantly changed due to minor changes in estimated action values. During policy optimization, we can smooth out the problem by applying stochastic policy class $\pi(s,a)$, which represent the probability of action a at state s. [3]

Besides all the advantages, policy gradient might goes to local maximum instead of global optimum. Also, policy gradients take longer to converge compared to Q learning.

2.1 Likelihood Ratio Gradient Descent

Likelihood ratio policy gradient is a variant of policy gradients. Define the state-action sequence $\tau = (s_0, a_0, s_1, a_1 ...)$ that covers the time space. The expected reward can be overloaded $R(\tau) = \sum_t R(s_t, a_t)$. Now the goal is to find the optimal parameter θ that maximize the utility function:

$$U(\theta) = \sum_{\tau} P(\tau;\theta)R(\tau) \tag{7}$$

Take the gradient with respect to θ gives:

$$\begin{aligned}\nabla_\theta U(\theta) &= \nabla_\theta \sum_{\tau} P(\tau;\theta)R(\tau) \\ &= \sum_{\tau} \nabla_\theta P(\tau;\theta)R(\tau) \\ &= \sum_{\tau} \frac{P(\tau;\theta)}{P(\tau;\theta)} \nabla_\theta P(\tau;\theta)R(\tau) \\ &= \sum_{\tau} P(\tau;\theta) \frac{\nabla_\theta P(\tau;\theta)}{P(\tau;\theta)} R(\tau)\end{aligned} \tag{8}$$

Apply the log likelihood property:

$$\nabla_\theta \log p(x;\theta) = \frac{\nabla_\theta p(x;\theta)}{p(x;\theta)} \tag{9}$$

Now we got:

$$\nabla_\theta U(\theta) = \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)R(\tau) \tag{10}$$

Since the model is unknown, use empirical estimate for m sample paths under policy π_θ [2]:

$$\nabla_\theta U(\theta) \approx \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^i;\theta)R(\tau^i) \tag{11}$$

Likelihood ratio changes probabilities of different paths rather than change the paths themselves. The gradient tries to increase the probability of paths with positive reward, and decrease the probability of paths with negative rewards. This method is valid even if reward function $R(\tau)$ is discontinuous, unknown or sample space of paths is a discrete set. The reason is that only the derivative $\nabla_\theta \log P(\tau, \theta)$ is needed for the estimator. The distribution $P(\tau, \theta)$ itself is not needed.

Lemma 1 *The log likelihood ratio gradient doesn't require dynamic models.*

Proof:

$$\begin{aligned}\nabla_\theta \log P(\tau;\theta) &= \nabla_\theta \log[\Pi_t P(s_{t+1}|s_t,a_t)\pi_\theta(a_t|s_t)] \\ &= \nabla_\theta[\sum_t \log P(s_{t+1}|s_t,a_t) + \sum_t \log \pi_\theta(a_t|s_t)] \\ &= \nabla_\theta[\sum_t \log \pi_\theta(a_t|s_t)] \\ &= \sum_\theta \nabla_\theta \log \pi_\theta(a_t|s_t)\end{aligned} \tag{12}$$

Hence the gradient of utility function became:

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t))R(\tau^i) \tag{13}$$

When we have a stochastic policy $\pi(a|s)$, we don't have to compute the derivative of $P(s_{t+1}|s_t, a_t)$ or maintain the model. Thus, the computational resources required would decrease significantly, compared with value iteration methods. However, when the policy $\pi(s)$ is deterministic, to compute gradient of log likelihood $\nabla_\theta \log P(\tau;\theta)$, we still need the derivative $\nabla_\theta \log P(s_{t+1}|s_t, a_t)$.

2.2 Baseline

We can subtract a baseline b from the policy gradient to reduce variance without changing expectation.

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)(R(\tau^i) - b) \tag{14}$$

Lemma 2 *Subtractions of baseline b would not change the expectation of gradient.*

Proof:

$$\begin{aligned}\mathbb{E}[\nabla_\theta \log P(\tau;\theta)b] &= \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta)b \\ &= \sum_\tau \nabla_\theta P(\tau;\theta)b \\ &= \nabla_\theta \sum_\tau P(\tau;\theta))b \\ &= 0\end{aligned} \tag{15}$$

The baseline b can be chosen through different ways. One of the choices is:

$$b = \mathbb{E}[R(\tau)] = \frac{1}{m}\sum_{i=1}^{m} R(\tau^i) \tag{16}$$

2.3 Temporal Structure

In Markov process, given the current state, the future and the past shall be independent. Hence, the future action doesn't depend on past reward. The variance can be further reduced by:

$$\begin{aligned}\nabla_\theta U(\theta) \approx \hat{g} &= \frac{1}{m}\sum_{i=1}^{m}(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_t r_{\hat{t}} - b) \\ &= \frac{1}{m}\sum_{i=1}^{m}(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{\hat{t}\geq t} r_{\hat{t}} - b)\end{aligned} \tag{17}$$

Now the baseline b could be set as:

$$b(s_t) = \mathbb{E}[R(s_t, a_t) + R(s_{t+1}, a_{t+1} + ...] \tag{18}$$

2.4 Discount for Variance Reduction

Here γ is the discount factor. We introduce it to eliminate the delay effect between actions and rewards. The intuition behind is that rewards closer to current time are more likely to be effected the current action.

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}(\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t))(\sum_{\hat{t}\geq t} \gamma^(\hat{t} - t)r_{\hat{t}} - b) \tag{19}$$

Now the baseline is:

$$b(s_t) = \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...] \tag{20}$$

2.5 Vanilla Policy Gradient

Combine the techniques mentioned above, we got the Vanilla Policy Gradient Algorithm:

```
Initialize policy parameter $\theta$, baseline $b$ ;
for iteration = 1,2... do
    Collect a set of trajectories $\tau^i$ by executing policy $\pi_\theta$ ;
    Compute reward at each time step and in each trajectory: $R_t^i = \sum_{\hat{t} \geq t} \gamma^(\hat{t} - t) r_{\hat{t}}$;
    Refit the baseline b;
    Update the policy using the policy gradient estimate $\hat{g}$;
end
```

Algorithm 1: Vanilla Policy Gradient Algorithm

3 Application: Image Captioning [1]

Image caption is a task to describe a image using human language, normally a sentence. An encoder-decoder model is used. The encoder is a neural network and the decoder is a recurrent neural network. The green and yellow nodes are the beginning of sequence and end of sequence. g is the image caption output and y is the ground truth. In testing and actual usage, the current input at time t is the previous output g_{t-1}. Policy gradient can be used in optimizing the reward function in the training process of the model and achieved promising results.

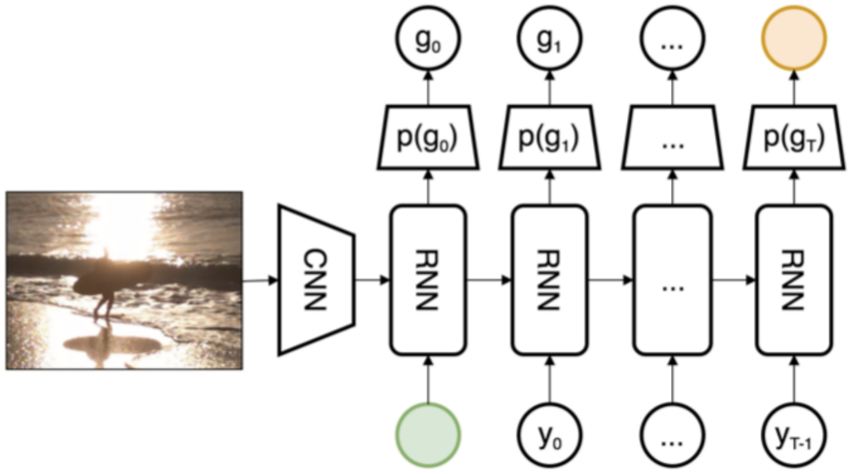


Figure 3: Model Architecture of Image Caption System [1]

At any time t, an action(word) g_t is picked through policy $\pi_{theta}(s_t)$. Here the state s_t is the sequence of all previous words $g_{1:t-1}$. The state transition is deterministic since at any time t, the future state s_{t+1} is the current state s_t concatenate the chosen word g_t. At the end, a reward $R(g_{1:T})$ will be given based on the whole sequence.

The goal training process is to find the optimal policy π_θ such that the reward is maximized. However, the reward can not be seen before the ending time T. So the Monte Carlo roll out is used to estimate the intermediate reward. As the graph below shown, each value is the average of K roll out sequences. The green and yellow nodes are the start and end.

$$\nabla_\theta V_\theta(g_{1:t}) = \sum_{t=1}^{T} \mathbb{E}_{g_{t+1:T}}(R(g_{t+1:T})) \tag{21}$$

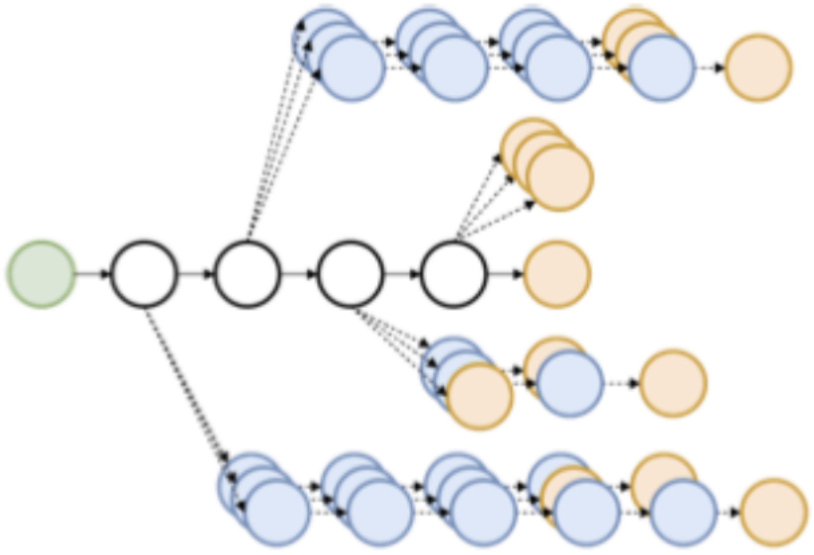


Figure 4: Sample Caption [1]

Here we define Q function:

$$Q_\theta(s_t, g_t) = \mathbb{E}_{g_{t+1:T}}(R(s_t + g_t, g_{t+1:T})) \tag{22}$$

To estimate the Q function, Monte Carlo roll out is used. K samples of sequence $s_t : gt$ are calculated and their average will be the estimation:

$$Q_\theta(s_t, g_t) \approx \frac{1}{K} \sum_{k}^{K} R(s_t + g_t, g_{t+1:T}) \tag{23}$$

Then the gradient of value function can be estimated:

$$\nabla_\theta V_\theta(s_0) = \sum_{t=1}^{T} \sum_{g_t} \mathbb{E}_{g_t}[\nabla_\theta \pi_\theta(g_t|s_t)] \times (Q_\theta(s_t, g_t)) \tag{24}$$

From the previous section we know that:

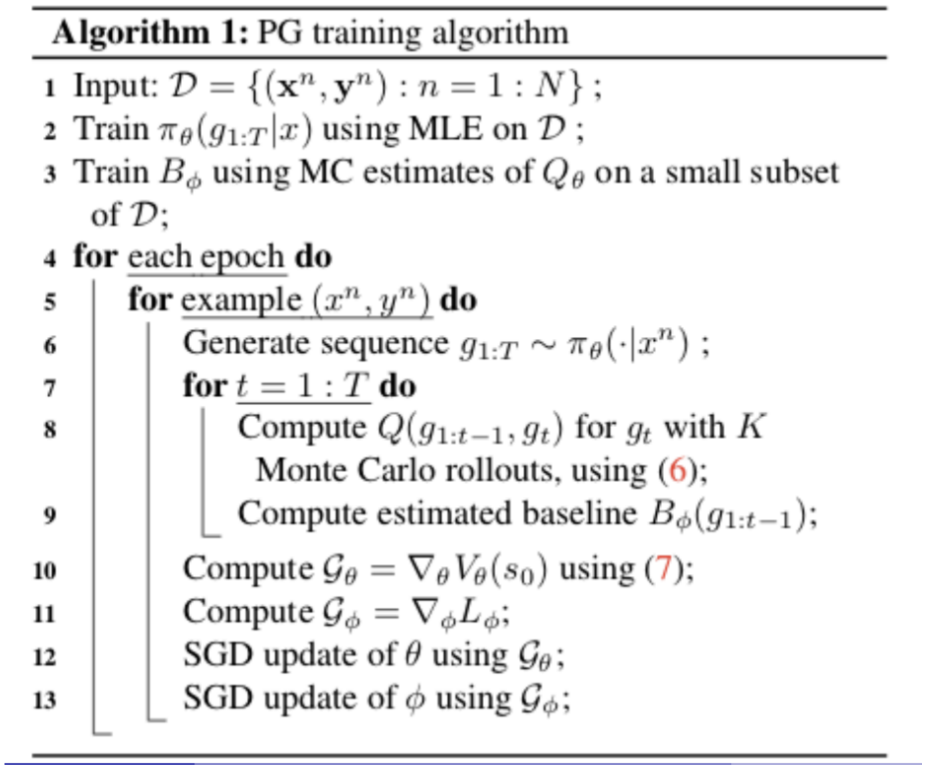
$$\nabla_\theta \pi_\theta(g_t|s_t) = \pi_\theta(g_t|s_t) \nabla_\theta \log \pi_\theta(g_t|s_t) \tag{25}$$

To reduce the variance, a baseline $B_\phi(s_t)$ is introduced. Thus, the estimate of gradient will be:

$$\nabla_\theta V_\theta(s_0) \approx \sum_{t=1}^{T} \sum_{g_t} [\pi_\theta(g_t|s_t) \nabla_\theta \log \pi_\theta(g_t|s_t)] \times (Q_\theta(s_t, g_t) - B_\phi(s_t)) \tag{26}$$

Based on the estimator, our loss function L_θ looks like:

$$L_\phi = \sum_{t} E_{s_t} E_{g_t} (Q_\theta(s_t, g_t) - B_\phi(s_t))^2 \tag{27}$$



Algorithm 1: PG training algorithm

```
1  Input: $\mathcal{D} = \{(\mathbf{x}^n, \mathbf{y}^n) : n = 1 : N\}$ ;
2  Train $\pi_\theta(g_{1:T}|x)$ using MLE on $\mathcal{D}$ ;
3  Train $B_\phi$ using MC estimates of $Q_\theta$ on a small subset
     of $\mathcal{D}$;
4  for each epoch do
5      for example $(x^n, y^n)$ do
6          Generate sequence $g_{1:T} \sim \pi_\theta(\cdot|x^n)$ ;
7          for $t = 1 : T$ do
8              Compute $Q(g_{1:t-1}, g_t)$ for $g_t$ with $K$
                 Monte Carlo rollouts, using (6);
9              Compute estimated baseline $B_\phi(g_{1:t-1})$;
10         Compute $\mathcal{G}_\theta = \nabla_\theta V_\theta(s_0)$ using (7);
11         Compute $\mathcal{G}_\phi = \nabla_\phi L_\phi$;
12         SGD update of $\theta$ using $\mathcal{G}_\theta$;
13         SGD update of $\phi$ using $\mathcal{G}_\phi$;
```

Figure 5: Policy Gradient Training Algorithm for Image Caption [1]

References

[1] S. Liu. *Improved Image Captioning via Policy Gradient optimization of SPIDEr*, 2017.

[2] J. S. Pieter Abbeel. *Deep Reinforcement Learning through Policy Optimization*, 2018. Available at https://people.eecs.berkeley.edu/ pabbeel/nips-tutorial-policy-optimization-Schulman-Abbeel.pdf.

[3] T. Simonini. *An introduction to Policy Gradients with Cartpole and Doom*, 2018. Available at https://medium.freecodecamp.org/an-introduction-to-policy-gradients-with-cartpole-and-doom-495b5ef2207f.