University of Illinois Spring 2020

CS446/ECE449: Machine Learning
Lecture 30: April 30

Lecturer: Alex Schwing Scribe: Yuxuan Zhang (yuxuanz8)

1 Recap

Reinforcement Learning (RL) is an area of machine learning concerned with how software agents take actions in an environment in order to maximize the rewards. Reinforcement learning is one of the three paradigms, alongside supervised learning and unsupervised learning. The environment is usually stated in the form of Markov Decision Processes (MDP) so that many algorithms can utilize its property and dynamic programming techniques.

Here is a summary for what we have learned in Reinforcement Learning.

Known MDP

- To compute V^*, Q^*, π^*: use policy/value iteration or exhaustive
- To evaluate fixed policy π: use policy evaluation

Unknown MDP

- Estimate transition probabilities using experience replay
- Q-learning

1.1 Markov Decision Process

Definition of MDP is stated as follows.

- A set of states $s \in S$.
- A set of actions $a \in A_s$.
- A transition probability $P(s'|s, a)$.
- A reward function $R(s, a, s')$ (sometimes just $R(s)$ or $R(s')$).
- A start and maybe a terminal state.

$$P(S_{t+1} = s'|S_t = s_t, A_t = a_t, S_{t-1} = s_{t-1}, A_{t-1} = ..., S_0 = s_0) = P(S_{t+1} = s'|S_t = s_t, A_t = a_t)$$

Markov indicates that the current state only depends on previous state and action. Given a description of a MDP, we want to perform actions according to a policy π^* so as to maximize the expected future reward. The policy is encoded as a mapping from States S to Actions A_s in the following.

$$\pi(s) : S \to A_s$$

There are three ways to find the best policy π^*, exhaustive search, policy iteration and value iteration. We will discuss each of them briefly afterwards.

1.2 Exhaustive Search

Exhaustive search is also called brute force. For each possible policy, compute expected future reward $V^\pi(s_0)$ and choose policy with the largest expected future reward at last. The problem of this approach is that the total number of policies, $\prod_{s\in S}|A_s|$, is too large if the environment has enormous states and actions. The way to compute expected future reward $V^\pi(s)$ for a given policy is to solve a linear system of equations.

$$V^\pi(s) = 0 \quad if\ s \in G$$
$$V^\pi(s) = \sum_{s'\in S} P(s'|s,\pi(s))[R(s,\pi(s),s') + V^\pi(s')]$$

However, solving a linear system of equations is expansive. Instead of it, we can use iterative refinement:

$$V^\pi_{i+1}(s) \to \sum_{s'\in S} P(s'|s,\pi(s))[R(s,\pi(s),s') + V^\pi_i(s')]$$

1.3 Policy Iteration:

1. Initialize policy π
2. Repeat until policy π does not change
   - Solve system of equations (e.g., iteratively)
     $$V^\pi(s) = \sum_{s'\in S} P(s'|s,\pi(s))[R(s,\pi(s),s') + V^\pi(s')]$$
   - Extract new policy π using
     $$\pi(s) = arg\max_{a\in A_s} \sum_{s'\in S} P(s'|s,a)[R(s,a,s') + V^\pi(s')]$$

1.4 Value Iteration

We can change search space from over policies to over values. After getting a optimal value function, the best policy can be decoded from the optimal value function as follows.

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

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

1.5 Q-learning

The algorithms and methods we talked in the previous sections are usually for the known transition probabilities and known rewards. What if model and those things are unknown. We can turn to simulation to collect samples and approximate transition probabilities using Q-learning algorithm. Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state. We describe the algorithm as:

1. Obtain a sample transition (s, a, r, s')
2. Obtained sample suggests:

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

3. To account for missing transition probability we keep running average

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

The figure 1 shows an example for the application of the Q-learning. Left is the environment and right is the table of probabilities.

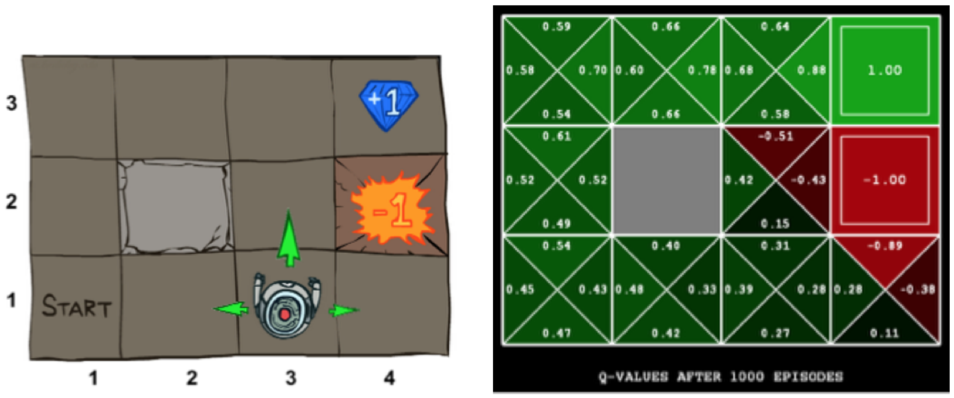

Figure 1: Q-learning example in a tablaur example

The objective for this lecture is to introduce more about Reinforcement Learning Techniques, Policy Gradient, and understand its relation to other methods.

2 Policy Gradient

The goal of reinforcement learning is to find an optimal strategy for the agent to obtain optimal rewards. The policy gradient methods target at modeling and optimizing the policy directly. The policy is usually modeled with a parameterized function respect to θ. The value of the reward (objective) function depends on this policy and then various algorithms can be applied to optimize θ for the best reward.

2.1 Variant: Likelihood ratio policy gradient

- Rollout, state-action sequence: $\tau = (s_0, a_0, s_1, a_1, ...)$
- Expected reward: $R(\tau) = \sum_t R(s_t, a_t)$

$$U(\theta) = E\left[\sum_t R(s_t, a_t); \pi_\theta\right] = \sum_\tau P(\tau; \theta) R(\tau)$$

The goal is to maximize the expected reward function of θ.

$$\max_\theta U(\theta) = \max_\theta \sum_\tau P(\tau; \theta) R(\tau)$$

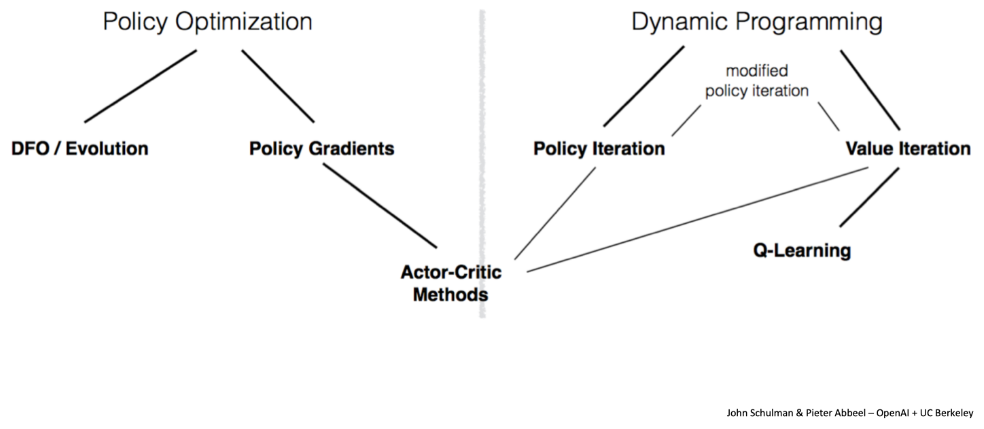

Figure 2: Relationships of methods in Reinforcement Learning

Gradient descent:

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

We can approximate the gradients with empirical estimate for sample paths under policy π_θ:

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

But we need to keep in mind that this approximation is impossible without trick. This empirical estimate is valid even if R is discontinuous. It is similar to maximum likelihood. We can increase probability of path τ with positive R and decrease probability of path τ with negative R. There is another advantage of this formula is that it does not need for dynamics model.

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

Consequently:

$$\Rightarrow \nabla_\theta \log P(\tau;\theta) = \sum_t \nabla_\theta \log \pi_\theta(a_t|s_t) \tag{2}$$

From equation 1, we can get

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_t \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\right) R(\tau^{(i)}) \tag{3}$$

2.2 Baseline

The variance from samples is high, so we add a baseline to the equation 3 to solve the issue .

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m} \nabla_\theta \log P(\tau^{(i)};\theta)\left(R(\tau^{(i)} - b)\right) \tag{4}$$

The subtraction of baseline b is fine because

$$
\begin{aligned}
E[\nabla_\theta \log P(\tau;\theta) b] &= \sum_\tau P(\tau;\theta)\nabla_\theta \log P(\tau;\theta) b && (5)\\
&= \sum_\tau \nabla_\theta P(\tau;\theta) b && (6)\\
&= \nabla_\theta \left(\sum_\tau P(\tau;\theta)\right) b && (7)\\
&= 0 && (8)\\
& && (9)
\end{aligned}
$$

Choices of b can be: e.g., (others are available at Greensmith et al. (2004) [1])

$$b = E[R(\tau)] = \frac{1}{m}\sum_{i=1}^{m} R(\tau^{(i)}) \tag{10}$$

2.3 Temporal Structure

Temporal structure of policy gradient is defined as,

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\left(\sum_t \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\right)\left(\sum_t R(s_t^{(i)}, a_t^{(i)}) - b\right) \tag{11}$$

Because future actions do not depend on past rewards, we can get lower variance through

$$\nabla_\theta U(\theta) \approx \hat{g} = \frac{1}{m}\sum_{i=1}^{m}\sum_t \nabla_\theta \log \pi_\theta(a_t^{(i)}|s_t^{(i)})\left(\sum_{\hat{t}\geq t} R(s_{\hat{t}}^{(i)}, a_{\hat{t}}^{(i)}) - b_{\hat{t}}^{(i)}\right) \tag{12}$$

One good choice for b is:

$$b(s_t) = E[r_t + r_{t+1} + ...] \tag{13}$$

3 Algorithm

Reinforce learning aka vanilla Policy Gradient.

- Initial θ, b
- For iteration = 1,2, ...
  - Collect a set of trajectories $\tau^{(i)}$ by executing policy $\pi_t heta$
  - Compute reward and bias
  $$R_t^{(i)} = \sum_{\hat{t}\geq t} \gamma^{\hat{t}-t} r_{\hat{t}}$$
  - Re-fit the baseline b
  - Update the policy using the policy gradient estimate $\hat{g}$

4 Application - Image Captioning

Image Captioning is the process of generating textual description of an image. It uses both Natural Language Processing and Computer Vision to generate the captions. Some methods solved this problem by training a statistical model on a dataset of (image, caption) pairs which is trained to maximized the log likelihood.

The flow of the algorithm proposed by Liu et al. is presented in the figure 3 [2]. Convolutional Neural Network (CNN) works as the encoder, extracting features to the decoder. y_i is the input text in the Recurrent Neural Network (RNN) model and g_i is the output. The green circle in the graph is the beginning of sequence (BOS) and the yellow is the end of sequence (EOS).

Figure 4 demonstrates the process of generating a sample. The value of each action is estimated as the average rewards received by its K rollout sequences. Solid arrows indicate the sequence of actions being evaluated. 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].

At time step t, we pick a discrete action, which corresponds to choosing a word $g_t \in V$, using a stochastic policy or generator $\pi_\theta(g_t|s_t, x)$, where $s_t = g_{1:t-1}$ is the sequence of words chosen so far,

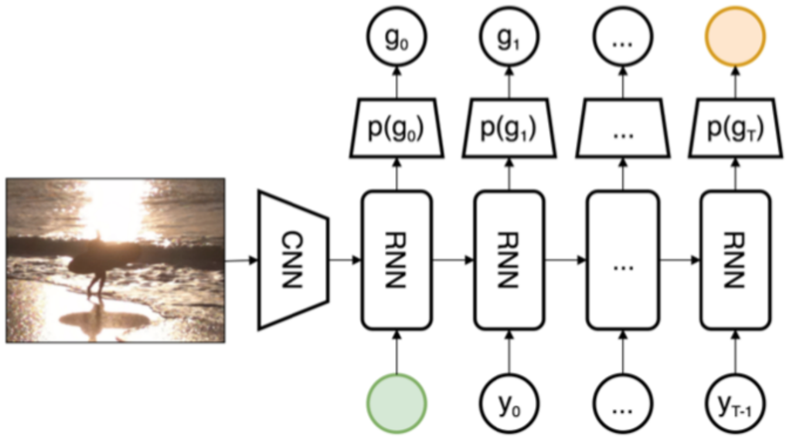

Figure 3: Model Architecture of Image Captioning

x is the image and θ are the parameters of the model. Note that in their case, the state transition function is deterministic: we simply append the word chosen at time t to get $s_{t+1} = st; g_t$. When we reach the end of the sequence, we get a reward of $R(g_{1:T}|x^n, y^n)$.

$$Q(g_{1:t-1}, g_t) \approx \frac{1}{K} \sum_{k=1}^{K} R(g_{1:t-1}; g_t; g^k_{t+1:T}) \tag{14}$$

We estimate how good a particular word choice g_t is by averaging over all complete sequences sampled according to the current policy, conditioned on the partial sequence $g_{1:t-1}$ sampled from the current policy so far. Similarly, we will denote baseline as $B_\phi(g_{1:t-1})$.

$$\nabla_\theta V_\theta(s_0) \approx \sum_{t=1}^{T} \sum_{g_t} [\pi_\theta(g_t|s_t) \nabla_\theta \log \pi_\theta(g_t|s_t) \times (Q_\theta(s_t, g_t) - B_\phi(g_{1:t-1})] \tag{15}$$

The overall algorithm is summarized in Algorithm 1. Note that the Monte Carlo rollouts only require a forward pass through the RNN, which is much more efficient than the forward-backward pass needed for the CNN. Additionally the rollouts can be also be done in parallel for multiple sentences [2].

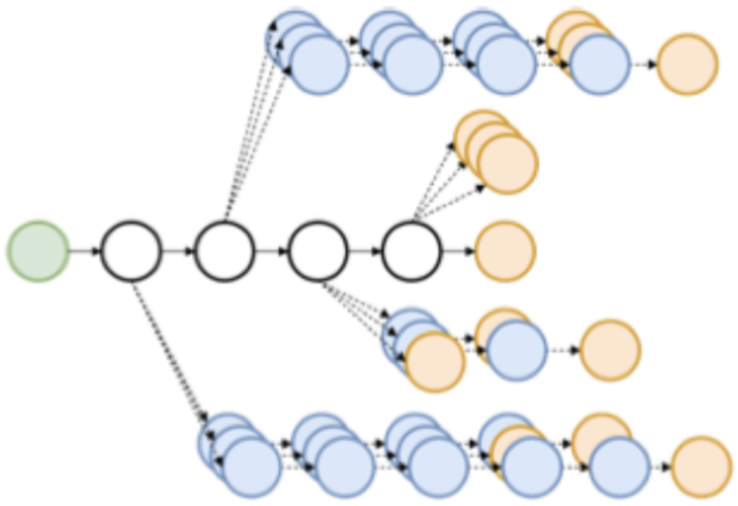

Figure 4: Sampling a caption

Algorithm 1: PG training algorithm

```
Input: $D = (x^n, y^n) : n = 1 : N$;
Train $\pi_\theta(g_{1:T}|x)$ using MLE on $D$;
Train $B_\phi$ using MC estimates of $Q_\theta$ on a small subset of $D$;
for each epoch do
    for example $(x^n, y^n)$ do
        Generate sequence $g_{1:T} \sim \pi_\theta(\cdot|x^n)$ ;
        for $t$=1:T do
            Compute $Q(g_{1:T}, g_t)$ for $g_t$ with $K$ Monte Carlo rollouts, using (14) ;
            Compute estimated baseline $B_\phi(g_{1:T})$ ;
        end
        Compute $\mathcal{G}_\theta = \nabla_\theta V_\theta(s_0)$ using (15);
        Compute $\mathcal{G}_\theta = \nabla_\phi L_\phi$ ;
        SGD update of $\theta$ using $\mathcal{G}_\theta$ ;
        SGD update of $\phi$ using $\mathcal{G}_\phi$ ;
    end
end
```

5 Future Course Recommendation

- Applied Machine Learning: CS 498
- Artificial Intelligence: ECE 440/CS 440
- MDPs, Reinforcement Learning: ECE 586
- Learning Algorithms: ECE 598 PV

- Machine Learning in Silicon: ECE 598
- Computational Inference and Learning: ECE 566
- Statistical Learning Theory: ECE 543/CS 598
- Computer Vision: ECE 549
- Digital Imaging: ECE 558

References

[1] E. Greensmith, P. L. Bartlett, and J. Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Machine Learning*, 5:1471–1530, 2004.

[2] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy. Improved image captioning via policy gradient optimization of spider. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.