ECE 544NA: Pattern Recognition
Lecture 25: November 27

Lecturer: Alexander Schwing Scribe: Xinrui Zhu

1 Recap

1.1 Markov Decision Processes [1]

- 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

Markov Assumption: Given the present state, the future and the past are independent.

1.2 Policy Evaluation

To evaluate fixed policy π. The slides in class demonstrated different cases

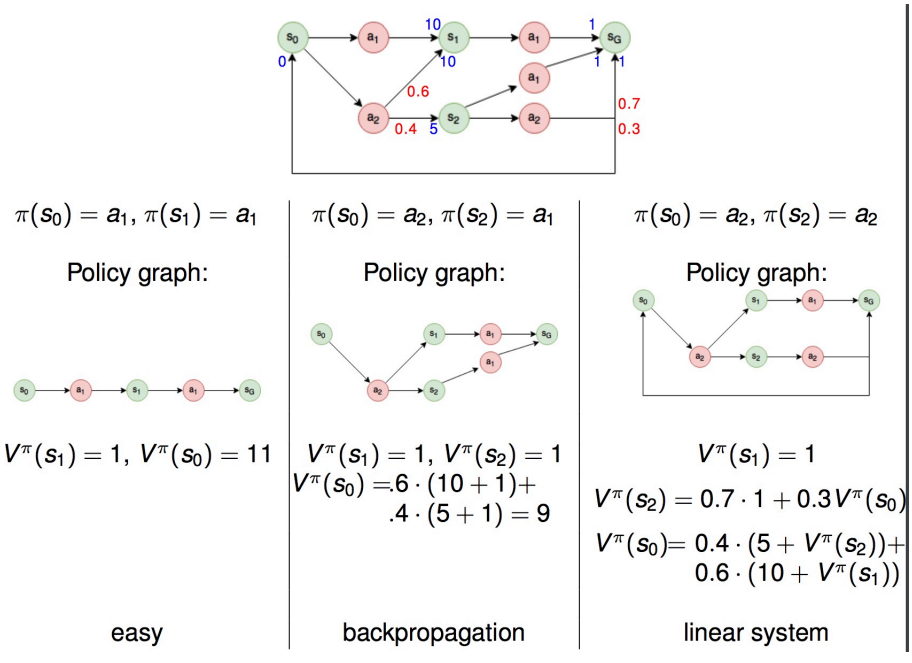

Figure 1: Policy Evaluation Examples

1.3 Exhaustive Search

Compute expected future reward $V^{\pi}(S_0)$ of all $\prod_{s \in S} |A_s|$ policies and then choose policy π^* with largest expected future reward $V^{\pi^*}(S_0)$

To compute expected future reward $V^{\pi}(s)$ for a given policy. We need to solve linear system of equations:

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

Instead of solving the linear equation above, we can use iterative refinement:

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

Still expensive since we need to search over all $\prod_{s \in S} |A_s|$ policies.

1.4 Policy Iteration

Start form some policies and get the optimal one with greedy method(showed below). Don't need to search all the policies.

```
Initialize Policy π
Repeat until policy π does not change
    Solve system of equations:
        V^π(s) = Σ_{s'∈S} P(s'|s,π(s))[R(s,π(s),s') + V^π(s')]
    Extract new policy π using:
        π(s) = argmax_{a∈A_s} Σ_{s'∈S} P(s'|s,a)[R(s,a,s') + V^π(s')]
```

1.5 Value Iteration

Instead of searching over policies, we can also search over values. Find the max value and decode the resulting policy at the end.

Bellman optimality principle:

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

Decoding policy:

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

1.6 Q Learning

In some cases, transition probabilities and rewards are not known. We need to run a simulator to collect experience tuples/samples (s, a, r, s') and estimated the transition probabilities using those samples.

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

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

To account for missing transition probability we keep running average

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

1.7 Deep Q-learning

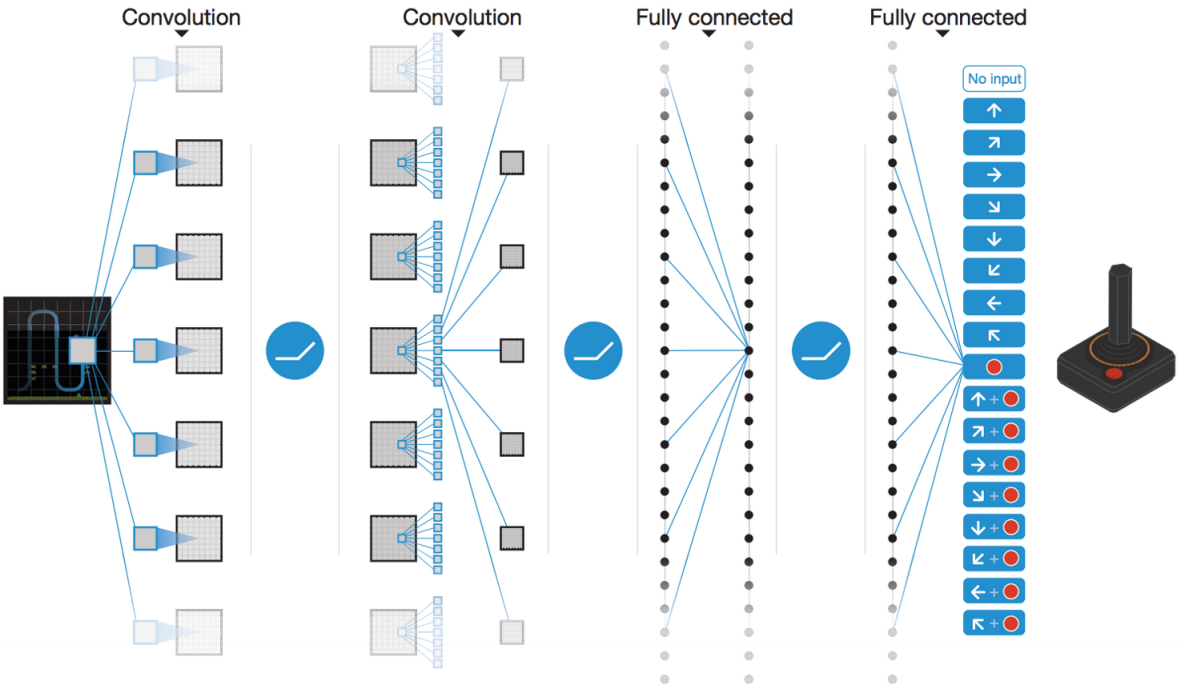

Figure 2: Deep Q-Learning

$Q_\theta(s, a)$ is a neural net which takes images as input and produces a number for each of the possible actions.

Given dataset $D = (s_j, a_j, r_j, s_{j+1})$:
Sample minibatch $B \subseteq D$
Compute target $y_j = r_j + \gamma \max_a Q_\theta - (s_{j+1}, a)$
Use Stochastic (semi-)gradient descent to optimize w.r.t parameters θ

$$\min_\theta \sum_{(s_j,a_j,r_j,s_{j+1}) \in B} (Q_\theta(s_j, a_j) - y_j)^2$$

Perform ϵ-greedy action and augment D

1.8 Summary

- Known MDP
  - To compute V^*, Q^*, π^*: use value/policy iteration
  - To evaluate fixed policy π: use policy evaluation
- Unknown MDP: Model free
  - To compute V^*, Q^*, π^*: use Q-learning
  - To evaluate fixed policy π: use value learning

2 Overview

2.1 What's Policy Gradient

In the last lecture, we worked with value-based reinforcement learning algorithms: Q-learning and Deep Q learning. To choose which action to take given a state, we take the action with the highest Q-value (maximum expected future reward at each state). And the policy was generated directly from the value function using ϵ-greedy

In this lecture, we'll discuss about a policy based reinforcement learning technique called **Policy Gradient**. In policy-based methods, instead of learning a value function that tells us what is the expected sum of rewards given a state and an action, **directly try to optimize our policy function $\pi_\theta(a|s)$ without worrying about a value function**.

2.2 Why Policy Gradient [3]

- **π may be simpler than Q or V and have better convergence properties**

  The problem with value-based methods is that they can have a big oscillation while training. This is because the choice of action may change dramatically for an arbitrarily small change in the estimated action values.

  On the other hand, with policy gradient, we just follow the gradient to find the best parameters. We see a smooth update of our policy at each step.

  Because we follow the gradient to find the best parameters, were guaranteed to converge on a local maximum (worst case) or global maximum (best case).

- **V doesn't prescribe actions: dynamics model + Bellman back-up needed**
- **Q requires efficient maximization: issue in continuous/high-dimensional action spaces**

  The problem with Deep Q-learning is that their predictions assign a score (maximum expected future reward) for each possible action, at each time step, given the current state.

  But what if we have an infinite possibility of actions?

  For instance, with a self driving car, at each state you can have a (near) infinite choice of actions (turning the wheel at 15, 17.2, 19,4, honk). Well need to output a Q-value for each possible action!

On the other hand, in policy-based methods, you just adjust the parameters directly: thanks to that youll start to understand what the maximum will be, rather than computing (estimating) the maximum directly at every step.

- **Can learn stochastic policies**

  We dont need to implement an exploration/exploitation trade off. A stochastic policy allows our agent to explore the state space without always taking the same action. This is because it outputs a probability distribution over actions. As a consequence, it handles the exploration/exploitation trade off without hard coding it.

  We can also get rid of the problem of perceptual aliasing. Perceptual aliasing is when we have two states that seem to be (or actually are) the same, but need different actions.

2.3 Disadvantages

- Typically converge to a local rather than global optimum. Instead of Deep Q-Learning, which always tries to reach the maximum, policy gradients converge slower, step by step. They can take longer to train.
- Evaluating a policy is typically inefficient and high variance.

3 Likelihood Ratio Method(REINFORCE)

The literature on policy gradient methods has yielded a variety of estimation methods over the last years. The most prominent approaches, which have been applied to robotics are finite-difference and likelihood ratio methods, better known as REINFORCE in reinforcement learning. In the lecture, we only discussed about Likelihood Ratio method (which is what I included in this section), but I'll also include some general information about finite-difference method at the end.

3.1 Problem Setup

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

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

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

3.2 Gradient Descent

Let's first take the gradient of both side:

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

Approximate the RHS with empirical estimate for sample paths under policy π_θ:

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

From the equation above, we can find out an important property: **Valid even if R is discountinuous**.

3.3 No need for dynamic model

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

We can notice that in the final result, we don't need to worry about $P(s_{t+1}|s_t, a_t)$ anymore. So, we don't need a dynamic model to estimate it. Plug this result back into the equation in previous section, we can get:

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

4 Practical Improvements

4.1 Baseline

When $R(\tau^{(i)}) > 0$, the variance of the gradient estimator $\hat{g}$ will be very big. In order to reduce the variance, we can subtract a baseline b from $R(\tau^{(i)}) > 0$:

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

It's straightforward to show that subtraction of baseline b won't introduce bias in the gradient

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

There are multiple choices of b, the one provided in the lecture is:

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

4.2 Temporal Structure

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

Sinde the future actions do not depend on past rewards, we can lower the variance by updating the above equation to:

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

And b will also be updated to:

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

5 Vanilla Policy Gradient

Based on the likelihood ratio we discussed above, we can get the vanilla policy gradient algorithm. But I guess it is worth mentioning that since the vanilla policy gradient is sensitive to parametrisations, people developed a parametrisation independent algorithm: natural policy gradient. It will find ascent direction that is closest to vanilla gradient by changing policy by a small fixed amount.

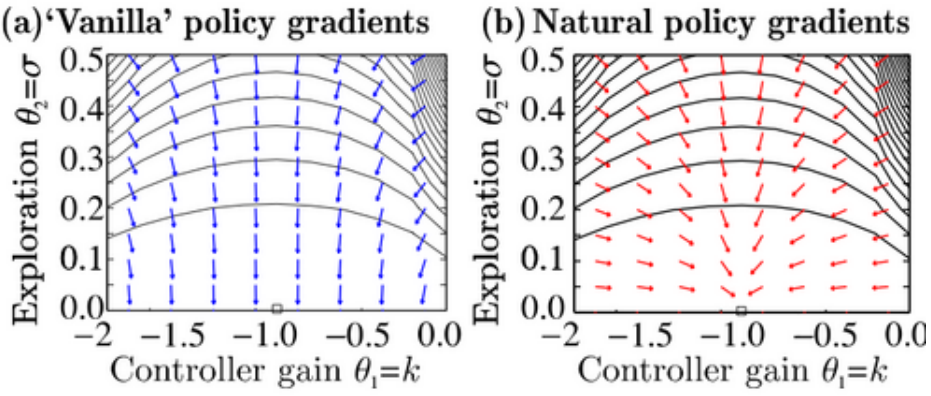

Figure 3: Vanilla policy gradients VS. Natural policy gradients

5.1 Algorithm Scratch

Initial θ, b
For iteration = 1, 2, ...
Collect a set of trajectories $\tau^{(i)}$ by executing policy π_θ
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 $\hat{g}$

5.2 Advantages [2]

Besides the theoretically faster convergence rate, likelihood ratio gradient methods have a variety of advantages in comparison to finite difference methods when applied to robotics. As the generation of policy parameter variations is no longer needed, the complicated control of these variables can no longer endanger the gradient estimation process. Furthermore, in practice, already a single roll-out can suffice for an unbiased gradient estimate viable for a good policy update step, thus reducing the amount of roll-outs needed. Finally, this approach has yielded the most real-world robotics results (Peters Schaal, 2008) and the likelihood ratio gradient is guaranteed to achieve the fastest convergence of the error for a stochastic system.

5.3 Disadvantages [2]

When used with a deterministic policy, likelihood ratio gradients have to maintain a system model. Such a model can be very hard to obtain for continuous states and actions, hence, the simpler finite difference gradients are often superior in this scenario. Similarly, finite difference gradients can still be more useful than likelihood ratio gradients if the system is deterministic and very repetitive. Also, the practical implementation of a likelihood ratio gradient method is much more demanding than the one of a finite difference method.

6 Finite-difference Methods*

Finite-difference methods are among the oldest policy gradient approaches; they originated from the stochastic simulation community and are quite straightforward to understand.

For each dimension $k \in [1, n]$
Estimate kth partial derivative of objective function w.r.t. θ
By perturbing θ by small amount ϵ in kth dimension

$$\frac{\partial U(\theta)}{\partial \theta_k} \approx \frac{U(\theta + \epsilon u_k) - U(\theta)}{\epsilon}$$

where u_k is unit vector with 1 in kth component, 0 elsewhere
Use n evaluations to compute policy gradient in n dimensions

Finite-difference methods require very little skill and can usually be implemented out of the box. They work both with stochastic and deterministic policies without any change. It is highly efficient in simulation with a set of common histories of random numbers and on totally deterministic systems.

However, the perturbation of the parameters is a very difficult problem often with disastrous impact on the learning process when the system goes unstable. In the presence of noise on a real system, the gradient estimate error decreases much slower than for the following methods. Performance depends highly on the chosen policy parametrization.

References

[1] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[2] J. Peters. Policy gradient methods. 2010.

[3] T. Simonini. An introduction to policy gradients with cartpole and doom.