Lecture Week-Recess: Optimal Control and Planning

Tan Ying Kiat, Vikash Ranjan
yingkiat(at)u.nus.edu, STUDENT2, etc.

1 Model-Based Reinforcement Learning

The objective of reinforcement learning is

$$\underbrace{p_\theta(\mathbf{s}_1, \mathbf{a}_1, ..., \mathbf{s}_T, \mathbf{a}_T)}_{\pi_\theta(\tau)} = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi_\theta(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \tag{1}$$

$$\theta^* = \arg\max_\theta {}_{\tau \sim p_\theta(\tau)} \left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \right] \tag{2}$$

1.1 Recap of Model-Free Reinforcement Learning

In model-free learning, we assume that the transition probability $p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t)$ is unknown and we can bypass it by sampling.

1.2 What if we knew the transition dynamics?

If we know the transition probability, it can make things easier because it becomes an optimisation problem.

And often we do know the transition dynamics, for example in games, easily modeled systems (such as navigating a car), and simulated environments (such as video games). We can learn the dynamics by system identification and learning.

This week's focus is on how to find out the action to take if we ***know*** the transition probability.

1.2.1 Deterministic

For the deterministic case, we want to find the arg max.

$$\mathbf{a}_1, ..., \mathbf{a}_T = \arg\max_{\mathbf{a}_1, ..., \mathbf{a}_T} \sum_{t=1}^{T} r(\mathbf{s}_t, \mathbf{a}_t) \; s.t. \; \mathbf{a}_{t+1} = f(\mathbf{s}_t, \mathbf{a}_t) \tag{3}$$

1.2.2 Stochastic Open-Loop

For the stochastic open-loop case, we should choose the best expectation, but this may be sub-optimal. As a corollary, if you buy lottery, you may not win even if it is the most optimal expectations.This is because taking action may not get you to an expected state.

$$p_\theta(\mathbf{s}_1, ...\mathbf{s}_T \mid \mathbf{a}_1, ..., \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \tag{4}$$

Exploration in Computer Science Research: Deep Reinforcement Learning (CS 6101, 2019), National University of Singapore.

$$\mathbf{a}_1, ..., \mathbf{a}_T = \arg\max_{\mathbf{a}_1,...,\mathbf{a}_T} E\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t) \mid \mathbf{a}_1, ..., \mathbf{a}_T\right] \tag{5}$$

1.2.3 Stochastic Closed-Loop

For the stochastic closed-loop case, it can be considered as a special case of open-loop, where t=1. *(Or is the open-loop a special case of the closed loop?)*

$$p(\mathbf{s}_1, \mathbf{a}_1, ...\mathbf{s}_T, \mathbf{a}_T) = p(\mathbf{s}_1) \prod_{t=1}^{T} \pi(\mathbf{a}_t \mid \mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t, \mathbf{a}_t) \tag{6}$$

$$\pi = \arg\max_{\pi} E_{\tau \sim p(\tau)}\left[\sum_t r(\mathbf{s}_t, \mathbf{a}_t)\right] \tag{7}$$

As an aside, on-policy/off-policy is a learning problem, while open-loop/closed-loop is a planning problem. They two (policy vs loop) are not correlated, although there are some similarities. Also of note, "global" optimises policy fo the entire existence of the problem, while "local" optimises for that particular timestep.

2 Stochastic Optimization Methods

2.1 "Guess Check" or "Random Shooting Method"

We have to solve optimisation problem in equation (3). The easiest way is not to worry that there are sequence of events and treat it like any other optimisation problem. We can abstract away optimal control/planning:

$$\mathbf{a}_1, ..., \mathbf{a}_T = \arg\max_{\mathbf{a}_1,...,\mathbf{a}_T} \underbrace{J(\mathbf{a}_1, ..., \mathbf{a}_T)}_{don't\ care\ what\ this\ is} \tag{8}$$

$$\mathbf{A} = \arg\max_{\mathbf{A}} J(\mathbf{A}) \tag{9}$$

The simplest method is the "guess and check" or "random shooting method":

1. pick $\mathbf{A}_1, ..., \mathbf{A}_N$ from some distribution.
2. choose $\mathbf{A}_i$ based on $arg\ max_i\ J(\mathbf{A}_i)$

2.2 Cross-Entropy Method (CEM)

The Cross-Entropy Method (CEM) works for continuous-valued inputs:

1. sample $\mathbf{A}_1, ..., \mathbf{A}_N$ from $p(\mathbf{A})$
2. evaluate $J(\mathbf{A}_1), ..., J(\mathbf{A}_N)$
3. pick the *elites* $\mathbf{A}_{i1}, ..., \mathbf{A}_{iM}$ with highest value, where M < N
4. refit $p(\mathbf{A})$ to the elites $\mathbf{A}_{i1}, ..., \mathbf{A}_{iM}$
5. go back to step (1)

The intuition is that if it is around the "good value", then we will likely find the optimal solution near there. This method works better in continuous space but is also dependent on the distribution.

This method will converge towards a single action because the optimisation dimension is not too big. The termination criteria is when the rewards do not change much anymore. The convergence is when the probability changes are geting smaller and smaller over time ("hill climbing aspect") and not because of N shrinking. The convergence is likely towards local, and not global, optimisation.

Other things to note

- Step 2 is parallelizable
- There is a "harsh dimensionality limit" because to cover the dimensions well, we may need an exponential number of samples to get some idea of the local landscape. However, this may not be practical.
- This method may also be used for closed-loop planning, but it could be very expensive.

2.3 Monte Carlo Tree Search (MCTS)

The Monte Carlo Tree Search algorithm is as follows:

generic MCTS sketch

1. find a leaf s_l using TreePolicy(s_1)
2. evaluate the leaf using DefaultPolicy(s_l)
3. update all values in tree between s_1 and s_l

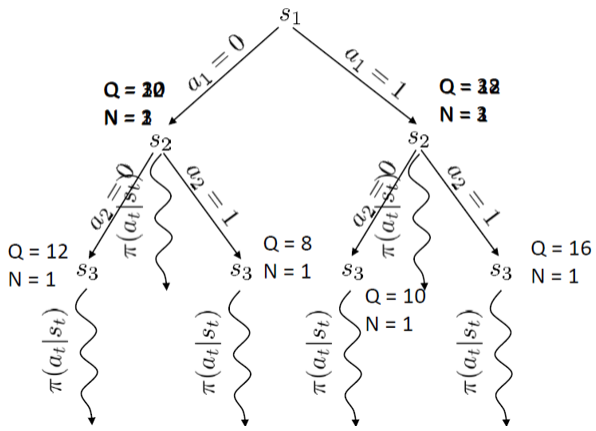
take best action from s_1

UCT TreePolicy(s_t)

if s_t not fully expanded, choose new a_t

else choose child with best Score(s_{t+1})

$$\text{Score}(s_t) = \frac{Q(s_t)}{N(s_t)} + 2C\sqrt{\frac{2\ln N(s_{t-1})}{N(s_t)}}$$



- The default policy is random sampling.
- Trying to balance between exploitation and exploration. C is the hyperparameter to tune this balance between exploitation and exploration.
- The score s_t is used to pick a node for expansion based on the score and the rarity of the node having been visited.
- Typically, the algorithm will run 1 trajectory at a time, where the score is used to decide which child node to visit. Only upon the completion of the trajectory will it return to the first node.

3 Trajectory Optimization: Linear Quadratic Regulation

Optimal control is about operating a dynamic systems at minimum cost. Linear quadratic (LQ) problems are whee the system dynamics are represented by linear differential equations, and the cost (or rewards) is represented by a quadratic function. Hence Linear quadratic regulation (LQR) is used to solve LQ problems.

LQR can be used for a special case of a finite horizon.

Slides 29 to 35 from the presentation deck contains the LQR algorithm. Briefly, the summary of the algorithm is given here:

1. (if necessary) estimate parameters A_t, B_t, Σ_t
2. initialize $\Phi_T := -U_T$ and $\Psi_T := 0$
3. iterate from $t = T$ - 1...0 to update Φ_t and Ψ_t using Φ_{t+1} and Ψ_{t+1} using the discrete Ricatti equations. If there exists a policy that drives the state towards zero, then convergence is guaranteed.

References

Presentation deck used by Lee Yong Ler, Teo Kok Keong, Lin Qian during presentation for CS6101 on 28 Feb 2019.

http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-10.pdf

http://cs229.stanford.edu/notes/cs229-notes12.pdf

http://cs229.stanford.edu/notes/cs229-notes13.pdf

http://rail.eecs.berkeley.edu/deeprlcourse/static/slides/lec-10.pdf

https://en.wikipedia.org/wiki/Linear%E2%80%93quadratic_regulator