ASEN 5264 - 'Cheat Sheet'

Manuel 'Ricky' Puyana & Julian Lema

Feb 2023

1 Probability

1.1 General

$$\sum_{x \in X} P(x|Y) = 1$$

$$P(X|Y)P(Y) = P(X,Y)$$
$$P(X|Y,Z)P(Y|Z) = P(X,Y|Z)$$

$$P(X,Y) = P(X)P(Y) \iff X \perp Y$$

$$P(X,Y|Z) = P(X|Z)P(Y|Z) \iff (X \perp Y)|Z$$

1.2 Bayes Rule

$$P(X|Y) = \frac{P(Y,X)}{P(Y)} = \frac{P(Y|X)P(X)}{P(Y)} = \frac{P(Y|X)P(X)}{\sum_{x \in X} P(Y,x)} = \frac{P(Y|X)P(X)}{\sum_{x \in X} P(Y|x)P(x)}$$

2 MDPs

2.1 Concepts

Markov Assumption: The Markov assumption is that your current state only depends on your previous state and the action you took to get there.

MDP Elements: A Markov decision process consists of a set of states S, a set of actions A, a transition function $T(s'|s,a)$, and a reward function $R(s,a)$.

Stationary MDP: A stationary MDP, shown below, is one where the transition and reward functions do not change with time. We will be dealing mostly with this type of MDP in this class.

Discount Factor: The discount factor is to give a higher value to plans that reach a reward sooner. It also bounds the utility of a state, ensuring it does not reach infinity. We typically denote the discount factor with γ, and it can hold values between 0 and 1. A small discount factor discounts future rewards more heavily. A solution with a small discount factor will be greedy, meaning that it will prefer immediate rewards more

than long-term rewards. The opposite is true for a large discount factor. An alternative to using a discount factor is to use the average reward,

$$\lim_{n\to\infty} \frac{1}{n} \sum_{t=0}^{n-1} r^t$$

Uniqueness of Optimal Policy and Value Functions: The optimal policy isn't necessarily unique, but the optimal value for each state is unique.

Value Function(U or V) vs. State Action Value Function(Q): U^π is the value function defined over states S, whereas Q^π is the state-action value function defined over (s, a),

$$U^\pi(s) = Q^\pi(s, \pi(s))$$

$$Q^\pi(s, \pi(s)) = R(s, a) + \gamma \sum_{s'} T(s'|s, a) U^\pi$$

Note that when the two expressions are combined, we obtain the familiar Bellman equation which recursively defines $U^p i$,

$$U^\pi(s) = R(s, a) + \gamma \sum_{s'} T(s'|s, a) U^\pi$$

2.2 MDP Formulation

All MDPs have the formulation {S, A, T, R} and sometimes {γ, b}.

S:
S is a collection of all of the states in an MDP. Think carefully about what to make the states! In Problem 4 from the 2021 exam, a naive set of states would have been $S : \{Boulder, Denver, Airport\}$; however, a better set of states was $S : \{Boulder, Low\ Traffic\ Denver, High\ Traffic\ Denver, Airport\}$. Note that the states do not all have to be spatially defined.

A:
A is a collection of all of the actions in an MDP. The actions do not all have to be possible from every state; however, the transition matrix and reward matrix can have unintended results this way. It is safer to formulate an MDP where all actions are possible from all states. In Problem 4 from the 2021 exam, it may be tempting to make each of the highways an action; however, the problem is simpler if the actions are listed as $A : \{1, 2\}$ where 1 and 2 are the choice between the two highways available at each state.

T:
T is a transition function that governs the transition from s to s' given action a. The general formulation is $T(s'|s, a)$. The matrix formulation is very useful when the state transition probabilities are known. In this form, T takes on a 3D matrix where the rows are all states s, the columns are all states s', and the 3rd dimension are the actions. This is more easily represented as a 2D matrix for each action. Example from HW2 Q2,

T :

$a = rest$	$left$	mid	$right$
$left$	1	0	0
mid	1	0	0
$right$	1	0	0

,

$a = roll$	$left$	mid	$right$
$left$	0	0.5	0.5
mid	0	0	1
$right$	0	0	1

R:
R is the reward function that generally maps each state-action pair to a reward. It can also map a state-action-state' set to a reward. These are generally expressed as $R(s, a)$ and $R(s, a, s')$ respectively. The

matrix formulation is very useful when rewards are known. In this form, R takes on a set of vectors for each action. Each element of the vector corresponding to action a_i corresponds to the reward for taking action a_i in each state. Example from HW2 Q2,

$$R: \begin{array}{c|c} a = rest & R(s,a) \\ \hline left & 0 \\ mid & 2 \\ right & -1 \end{array}, \begin{array}{c|c} a = roll & R(s,a) \\ \hline left & 0 \\ mid & 0 \\ right & 0 \end{array}$$

2.3 Bellman's Equations

Bellman's Expectation Equation:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma E[V^{\pi}(s')]$$

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s' \in S} T(s'|s, \pi(a)) V^{\pi}(s')$$

Matrix form,

$$\bar{V}^{\pi} = \bar{R}^{\pi} + \gamma \bar{T}^{\pi} \bar{V}^{\pi}$$

This can be used for policy evaluation by,

$$\bar{V}^{\pi} = (I - \gamma \bar{T}^{\pi})^{-1} \bar{R}^{\pi}$$

Bellman's Equation:

$$V^{*}(s) = max_{a \in A}(R(s,a) + \gamma E[V^{*}(s')])$$

$$V^{*}(s) = max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^{*}(s'))$$

Matrix form,

$$\bar{V}* = max_{a \in A}(\bar{R}^{a} + \gamma \bar{T}^{a} \bar{V}^{*})$$

This is used in bellman backup and as a certificate of optimality.

Bellman's Operator:

$$V'(s) = B[V](s)$$

$$V'(s) = max_{a \in A}(R(s,a) + \gamma E[V(s')])$$

$$V'(s) = max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V(s'))$$

Matrix form,

$$\bar{V'} = max_{a \in A}(\bar{R}^{a} + \gamma \bar{T}^{a} \bar{V})$$

This is used in value iteration. It is a contraction mapping. This will converge on V*.

3 Monte Carlo Policy Evaluation and Tree Search

3.1 General

Monte Carlo Policy Evaluation is the process of running a large number of simulations and averaging the accumulated reward. This utilizes random sampling to estimate unknown quantities, as well as using simulated experience that has been squired by running episodes of our MDP under a given policy to estimate the reward function.

The steps for the Monte Carlo Policy Evaluation are as follows

- Generate x amount of episodes under the policy that we are given or want to evaluate
- For episodes, we will calculate the total return from that state (sum of discounted rewards)
- Average our returns across all our simulations to then be able to obtain an estimate of our state value function that is dependent on our given policy

Monte Carlo Tree Search on the other hand takes our current state and time-step, and creates a 'tree' of our problem. This tree has nodes that represent a possible state and the connections represent possible ways to get to each node. Similar to a MCPE, it repeatedly chooses nodes further down to explore based on the policy it is given. From here it obtains estimates of expected outcomes with the actions that are available.

The MCTS has four main steps: selection, expansion, simulation, and backpropagation. In the selection stage, we have to decide whether we want our algorithm to explore possible routes (exploration), or utilize routes that have been found to already produce meaningful rewards (greedy). Once this step is complete, the tree search will add one more layer of nodes and runs a Monte-Carlo simulation until a terminal state is reached. The last step will reverse its findings to the initial step, and update the values of what nodes have been visited along the path. These four steps are repeated until the chosen stop criteria are met.

4 Offline Solutions

4.1 Policy Iteration

- Begin with a policy π
- Evaluate the policy with Bellman's Expectation Equation. $\bar{V}^{\pi} = (I - \gamma \bar{T^{\pi}})^{-1} \bar{R}^{\pi}$
- Improve policy: $\pi'(s) = argmax_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V^{\pi}(s'))$
  or $\bar{\pi}' = argmax_{a \in A}(\bar{R}^a + \gamma \bar{T}^a \bar{V}^{\pi})$
- Converged when $\pi = \pi'$
- Else, set $\pi = \pi'$ and repeat.

4.2 Value Iteration

- Begin with value function V
- Update with Bellman's Operator. $V'(s) = max_{a \in A}(R(s,a) + \gamma \sum_{s' \in S} T(s'|s,a) V(s'))$
  or $\bar{V}' = max_{a \in A}(\bar{R}^a + \gamma \bar{T}^a \bar{V})$
- Converged if $V = V'$ (or within some tolerance using the infinity norm). This shows convergence by the certificate of optimality of Bellman's Equation.
- Else, set $V = V'$ and repeat.

5 Online Solutions

5.1 General

So far we have discussed computing policies offline before any actions have been executed in the real world/problem. The issue of doing this presents itself when we are working with high-dimensional problems in which storage, computing, and execution of our algorithm start to take a toll.

In online tree searching algorithms the size of the state space is much smaller as it only considers and uses states that are reachable from the current state. The algorithm starts with it's initial state and expands the tree one node at the time - similar to the Monte Carlo Tree Search - except that it only expands a singular node at a time and not the entire space. IS THIS CORRECT?

The best way to think about this is that for offline learning you need a large data base to be able to train your model, while an online learning model learns in real-time by trial and error depending on it's application.

Check Pg. 190 in the book as it has a very good example with the 2048 game

5.2

6 Continuous MDPs

6.1 LQR

Given a system with Linear Dynamics and Quadratic Reward, there exists a deterministic solution for value function and policy. An example given in class is keeping a car at the center of a road. This breaks down if there is something in the middle of the road, the reward function would no longer be quadratic.

6.2 Model Predictive Control

There are three types of Model Predictive Control.

- Certainty Equivalent
- Open-Loop
- Hindsight Optimization

Certainty Equivalent This makes the assumption, like in LQR, that the new state is equal to the expectation of the transition from the old state given the old action. This may not hold if the reward function is not quadratic. Essentially we choose the optimal set of actions for one rollout that has no noise.

Open Loop This does many rollouts using a generative transition model and predetermined noise values at each state for each rollout. The model then chooses one set of optimal actions to govern the set of rollouts. This is a very conservative approach and takes into account some of the uncertainty in the model. This is called open loop because the set of actions will be enacted regardless of the individual state of each rollout.

Hindsight Optimization This is like Open Loop in that many rollouts are taken into account; however, the model selections a set of actions for each rollout. This is very optimistic. It is also overconfident as the model knows about future noise when selecting a set of actions. In fact, Hindsight Optimization can even select actions to preempt future noise. Note that at least the first action for all is the same.

6.3 Value Function Approximation

So far our problems have had relatively small state spaces. When we start moving into problems with large state spaces such as:

- Backgammon - 10^{20} states
- Go - 10^{170} states
- Spacecraft - continuous state space

We run into an issue with storage. So far we only have represented the value function by a lookup table where every state s has an entry V(s), or every state-action has a pair in $Q(s, a)$. This causes issues in being able to learn the value of each state individually.

To solve this, we will estimate the value function *function approximation* using a function similar to the following

$$V_\theta = f_\theta(s) \iff NeuralNetworks$$
$$V_\theta = \theta^T \beta(s)) \iff LinearFeature$$

In standard notation, this is shown in the following way.

$$\hat{v}(s, \mathbf{w}) \approx v_\pi(s)$$
$$\hat{q}(s, a, \mathbf{w}) \approx q_\pi(s, a)$$

Where w represents the weights that would be commonly associated with neural networks, for example. This vector of weights will be updated using methods such as Monte Carlo Simulations.

In this type of approximation, the value function is approximated using Neural Networks, Linear functions, or even a decision tree, and then takes the state of the environment as input and produces an estimate of the value function as an output.

The advantage of this is that an agent can learn a good policy in a large state space, or a noisy environment without having to explore every possible state-action combination.

Extra attention should be put into value function approximation programs as it requires careful tuning of the approximation to avoid overfitting or underfitting the data

A Appendix

```
mu = .1;
for 1 = 1:7
end
```

```
mu = .1;
for i in range(1,8)
end
```