Chapter 6 - Temporal-Difference Learning (Aravind S - EE14B013 - RL project RA 3)

TD Prediction

- TD and MC methods learn from close interaction with the environment, but the way they learn and hence their estimates are quite different.
- **Comparison**
  - DP - Full backup; System dynamics required; Bootstrapping;
  - MC - Sample backup; System dynamics not required; No bootstrapping;
  - TD - Sample backup (from MC) and Bootstrapping (from DP); System dynamics not required;

$$V(S_t) \leftarrow V(S_t) + \alpha\Big[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\Big]$$

Advantages of TD

- Better than DP, as model of environment is not needed (reward distribution, transition probabilities).
- Can be implemented in an online and incremental fashion. In MC, you need to wait until the end of episode to learn and hence updates occur only at the end of the episode; whereas in TD, updates can be done incrementally for every transition.
- TD learning for estimating values following a fixed policy π converges to V_π provided the stochastic approximation conditions hold on the step-size parameter α (Eqn 2.7).

Optimality of TD(0)

- Generally if we have a finite set of episodes, the increments for a particular state ‘s’ is accumulated throughout the set and the final total increment is applied to V(s). This is referred as batch updation and this implementation converges as long as α is sufficiently small. Similar batch-updation can be done for MC methods.
- Under batch-updation, TD(0) shows better convergence results than MC methods, but the solutions obtained are different.
- MC tries to minimise rms error on the training data whereas TD(0) finds estimates that would be a maximum-likelihood estimate for an inherent Markov chain for the system. This Markov chain is constructed in the most obvious way possible from the episodes encountered and batch TD(0) converges to this estimate (called certainty-equivalence estimate).
- Batch TD(0) converges faster than batch MC. Maybe because it assumes Markovian nature and since for a Markovian system the learning is easy (due to conditional dependence of future states only on the current state, action). On the other hand, for a non-markovian system, the relationship might be complex and hence obtaining such an estimate needs a longer time to converge.
- Non-batch TD(0) might be faster than constant-α MC as we try to move to a better estimate quickly, but convergence still takes time.

SARSA: On-policy TD Control

- As usual, using TD prediction for action-values with GPI gives a method for control.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)\Big].$$

- The above update is done for every non-terminal state S_t. Action-values for (terminal state, any action) is set to zero.
- SARSA converges to optimal policy and optimal value-functions provided all (s, a) pairs are visited infinitely often, the policy converges to the greedy policy in the limit (eg: Use an ε-greedy policy with a decaying ε) and α obeys Eqn 2.7.
- In case of policies which don’t have a termination, MC methods fail. On the other hand, since SARSA does step-by-step updates, it quickly learns during the episode and changes policies. As a result, poor policies get ruled out.
- It is an on-policy algorithm as the update involves A_{t+1} which depend on the policy followed and hence the estimates correspond to the policy which was used to generate the episodes.

Q-learning: Off-policy TD Control

- Here the optimal action-value function is learnt independent of the behaviour policy. In other words, it is an off-policy method.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\Big[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\Big].$$

- Having a closer look at the equation, we can see that after a large number of updates it will resemble Bellman-optimality equation as the average of samples become the expectations.
- Q-learning converges to the optimal policy and the optimal action-value function under the assumption of stochastic approximation conditions on the step-size parameter α.

Q-learning Vs SARSA

- Both converge to the same optimal policy if the policy used in SARSA approaches greediness in the limit.
- For an exploratory policy in SARSA, the obtained policies are different. SARSA results in safer policies (as future exploratory actions are considered; still optimal among all exploratory policies) whereas Q-learning results in optimal policies.

Expected SARSA

- Minor variant of SARSA/Q-learning. The bootstrap term ($Q(S_{t+1},...)$) is replaced with an expectation (Eqn 6.9).
- **Advantages**
  - Lesser variance than SARSA (as in SARSA A_{t+1} are chosen randomly).
  - Lesser dependence on step-size parameter α as expectation is used directly in the update rule.

Afterstates

- Useful when we know the initial part of the dynamics of the system. Also if a particular configuration can occur in multiple ways, better to use Afterstate value functions.