CME241 Notes

Saakaar Bhatnagar

Model Free Prediction and Control

1 Model Free Prediction

1.1 Monte-Carlo Learning

- Simplest most intuitive algorithm
- Simply averages returns from episodes
- Requires terminating MDPs
- Only works offline, updates at end of each episode
- Cannot learn from incomplete episodes, and only converges by law of large numbers, which makes it unusable in practical situations
- Low Bias (no estimates), high variance(lots of potentially random qty used per update)

Algorithm:

For every episode:

$G_t = \sum_{n=1}^{T} \gamma^{n-1} Rt + n$

N(s)=N(s)+1, depending on first visit/every visit variation

S(s)=S(s)+G_t

V(s)=S(s)/N(s)

1.2 Temporal Difference methods

- Based on the idea of running averages
- Involves bootstrapping
- Can do online updates
- Learn from incomplete episodes, infinite MDPs
- High Bias(estimate from estimate), low Variance(fewer stochastic qty used per update)

Method of running averages:

$$\mu_k = \mu_{k-1} + \frac{x_j - \mu_{k-1}}{k}$$

In a sense, we are averaging the value function :

$$V(s_t) = V(s_t) + \alpha(G_t - V(s_t)$$

The above formulation is for M.C method.

TD(0) formulation:

Update equation:

$$V(s) = V(s) + \alpha(R + t + 1 + \gamma V_{s'} - V_s)$$

1.2.1 AB example from class

Given the current data:

Using the M.C method:

$$V(B) = \frac{\sum G_t(s_t = b)}{N(s_t = b)} = 6/8$$
$$V(A) = 0$$

since the returns of that episode are 0, V(A)=0

Using T.D:

$$V(B) = 6/8$$

Using a convergent value of 6/8 for the evaluation of V(A):

$$V(A) = V(A) + \frac{0 + V(B) - V(A)}{N(A)}$$
$$V(A) = 6/8$$

V(A) is different for both methods.

1.3 TD-Lambda

- Uses n-step returns
- Assigns weights based on λ to each n-step return
- A mid solution of TD(0) and M.C methods
- Forward view method is intuitive but suffers from pitfalls of Monte Carlo
- Backward view method using Elgibility traces is more useful
- Accumulated offline update for Backward view method and update for forward view method is the same, can see by telescoping

- Online equivalence is not there b/w forward and backward method

Formulation:

The update to be made is:

$$V(s) = V(s) + \alpha(G_t^\lambda - V(s))$$

where:

$$G_t^\lambda = (1-\lambda)\sum_{n=1}^{\infty} \lambda^{n-1} G_t^n$$

where

$$G_t^n = \sum_{i=1}^{n} \gamma^{i-1} R_{t+i} + \gamma^n V(s_{t+n})$$

1.3.1 Eligibility Traces

Backward view algorithm is implemented using eligibility traces, defined as:

$$E_0(s) = 0$$

$$E_t = \gamma\lambda E_{t-1} + 1(s_t = s)$$

and the consequent update algorithm:

$$V(s) = V(s) + \alpha\delta_t E_t$$

where $\delta_t = R_{t+1} + \gamma V(s_{t+1} - V(s_t))$, the single step update.
This is done for every state, for every time step, in every episode.

2 Model Free Control

2.1 Exploration versus exploitation

- Problem of knowing how much reward to accumulate, versus how much to explore in hope of better rewards
- For MDPs with known models, this is not a problem and the problem of finding the best policy is simply taking greedy actions
- For unknown MDPs, we must explore
- Hence, in model-free policy iteration we use ϵ- greedy policy updates

Hence, most policy update algoriths are ϵ-greedy:

$$\pi(a|s) = \begin{cases} 1 - \epsilon + \epsilon/m, & a = \max_a Q(s,a) \\ \epsilon/m, & \text{otherwise} \end{cases}$$

We can show that following an ϵ-greedy algorithm leads to value function improvement:

$$v_\pi = q_\pi(s, \pi'(s)) = \sum_{a \in A} \pi'(a|s) q(s, a)$$

Following ϵ-greedy policy:

$$= \epsilon/m \sum_{a \in A} q(s, a) + (1 - \epsilon) max_a q(s, a)$$

Now we can write, since π, π' are both ϵ-greedy:

$$max_a q_\pi(s, a) >= (1 - \epsilon) \sum_{a \in A} \frac{\pi(a|s) - \epsilon/m}{1 - \epsilon} q_\pi(s, a)$$

since $1 > 1 - \epsilon/m$, which is the inequality we get on expanding the RHS above.

On simplifying, we get:

$$q_\pi(s, \pi'(s)) >= q_{pi}(s, \pi(s))$$

2.2 Monte Carlo Policy Iteration

- We can use Monte Carlo for approximating Q value function, then use ϵ-greedy to update the policy
- Monte Carlo will give an approximation of the value function for finite episodes
- We can use the GLIE Monte Carlo algorithm to make sure convergence to optimal value function occurs

GLIE Algorihtm: Given that every state action pair is explored infinitely many times, the ϵ-greedy algorihtm converges to the greedy policy.

We can implement this in M.C by making $\epsilon = 1/k$, where k is the timestep. Hence the algorithm gets greedier and greedier as time progresses. We also have guaranteed convergence to the optimal policy

2.3 SARSA and SARSA-λ

- SARSA is the Q-value function equivalent of TD(0)
- SARSA-Lambda is the Q-value function equivalent of TD(λ)
- They are on-policy algorithms, meaning they use the same policy to sample actions and update the value function
- The major difference in between TD and SARSA algorithms is that the tabular SARSA algorithms are encoded for state-action pairs
- SARSA is also reminiscent of value iteration, in that it makes chooses an action based on policy, and updates.
- SARSA only converges if the rules of GLIE are followed, and Robbins-Monro step size rules are followed.

The basic update equation of SARSA is:

$$Q(s,a) = Q(s,a) + \alpha(R(s,a) + \gamma Q(s',a') - Q(s,a))$$

where a,s',a' are all sampled from the same policy. The update equation from SARSA-λ is

$$Q(s,a) = Q(s,a) + \alpha(q_t^\lambda - Q(s,a)$$

$$q_t^\lambda = (1-\lambda)\sum \lambda^{n-1} q_t^n$$

2.4 Off Policy Learning

- Use a behavioral policy to learn a different policy
- The behavioral policy can be a collection of different policies

The Q-Learning algorithm update equation is:

$$Q(s_t,a) = Q(s_t,a) + \alpha(R_{t+1} + \gamma(Q(s_{t+1},a')) - Q(s_t,a))$$

where $a \in \mu(a|s), a' \in \pi(a|s)$

If the target policy is greedy, we have a simplified expression for the target:

$$\begin{aligned}\text{Target}&=R_{t+1} + \gamma Q(s_{t+1}, argmax_a Q(s_{t+1},a)) \\ &= R_{t+1} + \max_a \gamma Q(s_{t+1},a)\end{aligned}$$

Which makes the update equation the following:

$$Q(s_t,a_t) = Q(s_t,a_t) + \alpha(R_{t+1} + \gamma \max_a(Q(s_{t+1},a)) - Q(s_t,a_t))$$