Lecture 19

TD Learning

Nondeterministic MDP

what if reward and action functions have probabilistic outcomes ($r(s,a) = r$ with probability $P(r|s,a)$; $\delta(s,a) = s'$ with probability $P(s'|s,a)$)?

value function: expected discounted cumulative reward, accumulated over transitions from current state

$$V^{\pi}(s) = E[\sum_{i=0}^{\infty} \gamma^i r(s_i, a_i)]$$

$$\begin{aligned} Q(s,a) &= E[r(s,a) + \gamma V^*(\delta(s,a))] \\ &= E[r(s,a)] + \gamma \sum_{s'} P(s'|s,a) V^*(s') \end{aligned}$$

$$Q_n(s,a) = \sum_r P(r|s,a) r + \gamma \sum_{s'} P(s'|s,a) \max_{a'} Q(s',a')$$

Q-learning for nondeterministic MDP

need to update learning rule – will not converge in non-deterministic case

instead make revisions more gradually, smoothed with parameter α

$$\hat{Q}_n(s,a) = (1-\alpha_n)\hat{Q}_{n-1}(s,a) + \alpha_n[r + \gamma \max_{a'} \hat{Q}_{n-1}(s',a')]$$

parameter α_n determines smoothness of learning (decayed weighted avg)

$$\alpha_n = \frac{1}{1 + visits_n(s,a)}$$

Temporal Difference learning

Q learning is a form of TD learning – learn by reducing discrepancies between estimates made by agent at different times

can extend lookahead beyond next state (based on observed rewards for n steps):

$$\hat{Q}^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + ... + \gamma^{(n-1)} r_{t+n-1} + \gamma^{(n)} \max_a \hat{Q}(s_{t+n}, a)$$

$TD(\lambda)$ blends Q value estimates from different lookahead steps:

$$\begin{aligned}\hat{Q}^{\lambda}(s_t, a_t) &\equiv (1-\lambda)[\hat{Q}^{(1)}(s_t, a_t) + \lambda \hat{Q}^{(2)}(s_t, a_t) + \lambda^2 \hat{Q}^{(3)}(s_t, a_t) + ...] \\ &= r_t + \gamma[(1-\lambda) \max_a \hat{Q}(s_t, a_t) + \lambda \hat{Q}^{\lambda}(s_{t+1}, a_{t+1})]\end{aligned}$$

$0 \leq \lambda \leq 1$ controls degree of lookahead – more lookahead can produce more efficient training (if following optimal policy, $\hat{Q}^{\lambda=1} \rightarrow$ true Q)

Generalization

hypothesis in Q learning involves table lookup

convergence proofs depend on visiting every state-action pair infinitely often

practical systems incorporate function approximation, for example:

- encode state and action as inputs to neural net
- target output is value of $\widehat{Q}(s, a)$

TD-Gammon

Most famous success story of reinforcement learning – competitive with best human players: 1994 version lost by one point over 40 game match against world class grand-master

Backgammon – opponents have 15 checkers that move in opposite directions, movement governed by roll of two dice

Key idea: too many states to store huge table of values for each state – use backprop network to estimate value function $V(x)$ from state x

Training TD-Gammon

Trained from scratch using TD:

- each board configuration a state (number of black or white pieces at each location)
- random initial assignment of utility values to states
- action generated by selecting next configuration with highest utility
  1. give each possible next state to network to obtain utility
  2. choose next state s_{t+1} with highest utility
- end: scalar reward of 1 if win, 0 if lose
- target for $V_t(s_t)$ is roughly $r_{t+1} + \gamma V_t(s_{t+1})$
- gradient for parameter $\theta_i = \alpha[r(s_t) + \gamma \widehat{V}_t(s_{t+1}) - \widehat{V}_t(s_t)] \frac{\partial \widehat{V}_t(s)}{\partial \theta_i}$

2 versions of program play against each other