CS401 Lecture Notes - 24th November, 2014

Reinforced learning

This area of machine learning is concerned with software taking actions in an environment or world to maximize reward.

Example: Bacteria, only has limited memory. Gets information about its environments and performs actions now such as moving a limb, secrete mucus to gain reward in the future e.g. Secrete toxins now, sneezes later, infects everyone else, reward.
OR chess, moves early on in game cause huge reward later in game

Using reward and punishment, maximize rewards (or minimise punishment). Actions performed now will cause future reward.

Delayed rewards, this is called credit assignment problem. A technical difficult problem in RL.

Formalisation of credit assignment problem:

$r(t)$ - reward at time: t
$a(t)$ - action at time: t
$s(t)$ - state at time: t

"policy" state->action $a = \pi(s)$

Deterministic policies for now.

We need a notion of future rewards:

$$R(t) = r(t) + r(t+1) + r(t+2) + \ldots + r(t+n)$$

Discounted future rewards, discount factor: alpha α

Amortized reward

$$R(t) = r(t) + \alpha\, r\,(t+1) + \alpha^2 r(\,t+2\,)\ldots$$

$$= \sum_{i=0}^{\infty} \alpha^{\bar{i}} r(t+\bar{i})$$

$$= r(t) = \alpha \sum_{\bar{i}=0}^{\infty} \alpha^{\bar{i}\;1} r(t+\bar{i}^{\,-1}+1)$$
$$= r(t) + \alpha R(t+1)$$

$$0 \le \alpha \le 1$$

Like interest rate, i.e. if money is devalued by 2% α = 0.98

Animals don’t use this as it’s reward system.

Policy
We know the rewards, such as solitare or maze. We know the rules, we know state,action,policy and the reward we’ll get.

$S \, x \, A \; {-}{>} \; S$
$S \, x \, A \; {-}{>} \; reward$
$r(S, \alpha) = reward\ given$
$s(S, \alpha) = next\ state$

Tables/ Matrix

$V^{\pi}(s) =$

$= r(s, \pi(s)) + \alpha \, r(\, s\,(\, s, \pi(s)) + \pi(s(s, \pi(s))) + \ldots\ldots$

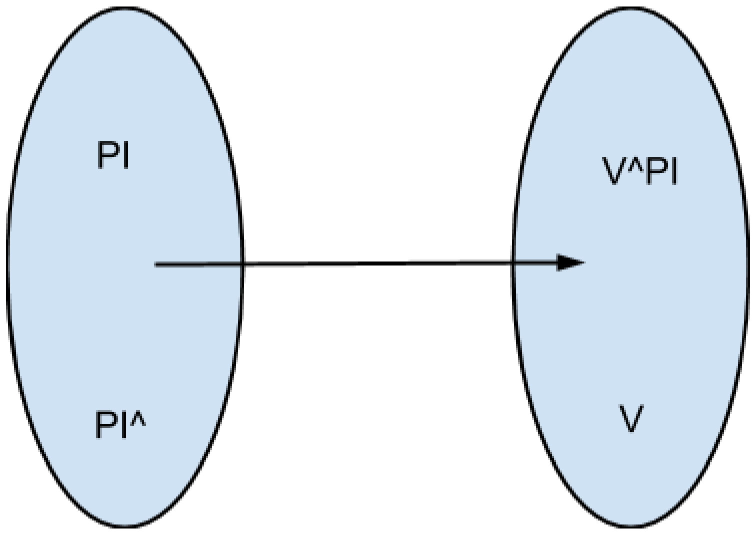
= immediate reward + discounted reward

$r(s, \pi(s)) + \alpha \, V^{\pi}(\pi(s))$

Every state has a fixed set of actions, i.e. a Maze, your actions are to move North, East, South, West

$v = (r(s, \pi(s))_{s} + \alpha(\qquad) \, v$

We can use iteration to solve problems



Policy Space Value Space

Policy to Value function
New policy:
$\overline{\pi}(s)$ = action which maximises immediate reward at state, add amortised reward after

$\overline{\pi}(s) = argmax\ r(s,a) + \alpha V^{\pi}(s, \pi(s))$

We'll take the best action, and then follow policy π

We'll generate a new policy off the old policy. Create new policy, new value function etc.

After a few iterations this should die down and we'll get optimal policy.

Policy Value Iteration

Cons:

- batch
- full knowledge of world model required
- on policy: You consider the move which is best move, can't explore

(this can be used for TSP)

Wouldn't work if trying to learn a world model.

Q Learning:

This is a model free reinforcement learning method. Q-learning can be used to find an optimal action-selection policy. Relaxes requirements.

The algorithm therefore has a function that calculates the Quality of a state-action combination:

$Q \,:\, S\,x\,A \;\Rightarrow R$

$$Q_{t+1}(s_t,a_t) = \underbrace{Q_t(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha_t(s_t,a_t)}_{\text{learning rate}} \times \left[\overbrace{\underbrace{R_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \underbrace{\max_a Q_t(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q_t(s_t,a_t)}_{\text{old value}} \right]$$

If you don't trust your Q function

$Q^{\pi}(s(t),\ a(t)) \ = \ r(t) \ + \ Q\ (s(t+1)\ \ldots\ missing\ parts$

from then on

follow optimal policy

$Q(s(t),\ a(t)) \ = \ r(t) \ + \ \alpha max Q(s(t),\ \alpha)$

Just update Q function as you do stuff

Q learning is online?

- no world model

Even if you take stupid action, still valid

Q leaning

- online
- does not require good moves
- off policy
- something about # of actions

Policy value iteration

- batch

- world model reqired,
- “on policy”
- # of states something

If learning world model you can update Q model for all State, Action, Policy,

If you think they’ll lead to a bad Q

Some way to ensure you try everything (State, Action, Policy)

You can use approximation with SL