MDP Model

- Given by transitions $\Pr(s' | s, a)$
- reward $r(s, a)$
- Criterion: expected total discounted reward (discount factor $\gamma$)

Backup Operators

Bellman Backup

$$[B(V)](s) = \max_a [r(s, a) + \sum_s \Pr(s' | s, a)V(s')]$$

Extracting a policy

$$[Greedy(V)](s) = \arg \max_a [r(s, a) + \sum_s \Pr(s' | s, a)V(s')]$$

Bellman Backup restricted to policy

$$[B^\Pi(V)](s) = r(s, \Pi(s)) + \sum_s \Pr(s' | s, \Pi(s))V(s')$$

Policy Evaluation (calculate $V^\Pi$)

- Solve linear equations $V = B^\Pi(V)$
- Iterative Alg: Repeat $V \leftarrow B^\Pi(V)$
- The solution is $V^\Pi$

Planning / Optimization

- VI: Repeat $V \leftarrow B(V)$
- PI: Repeat $\Pi = greedy(V); V = V^\Pi$
- Another view of PI:
  - Repeat
    - $Q^\pi(s, a) = r(s, a) + \gamma \sum_{s'} \Pr(s' | s, a)V^\pi(s')$
    - $\pi(s) = \arg \max_a Q^\pi(s, a)$
### Learning

- Transition and reward model not given (or problem too large to solve with planning)
- Learn model and plan, or use model free method
- Bandits: are "1 state MDPs"
- MC: Monte Carlo: evaluate $Q(s,a)$ using independent random rollouts
- TD: Temporal Difference: $Q(s,a)$ estimate uses previous value of next state in the rollout

### Exploration policy

- is crucial so that active RL does not get trapped with good estimate of bad policy
- Epsilon-exploration: pick optimal action with prob=$[1-\text{epsilon}]$ and random action with prob=$\text{epsilon}
- Softmax exploration
  $$p(a_t) = \frac{e^{Q(a_t)/T}}{\sum_k e^{Q(a_k)/T}}$$

### On Line Optimization (SARSA)

Repeat:
- [in state $s$] take action $a$; observe $r,s'$
- choose next action $a'$ using policy $P$
- $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$
- $P = \text{epsilon-greedy w.r.t. } Q$
- $s=s'$; $a=a'$

### On Line Optimization (Q learning)

Repeat:
- [in state $s$] take exploration policy action $a$; observe $r,s'$
- $Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_a Q(s',a') - Q(s,a)]$

### RL in practice

- Cannot afford to enumerate states
- In some problems cannot afford to enumerate actions
- Must use generalization.
- The $V()$, $Q()$, $p()$ are explicitly represented as functions of state/action
- (e.g. decision tree; neural network)
- Adapt algorithms to learn these representation

- 1950s: Checkers
- 1960s: poll balancing, tic tac toe
- 1990s: Backgammon
- 2010s: Go
- Using many ideas: function approximation, supervised learning, policy gradients, Monte-Carlo tree search