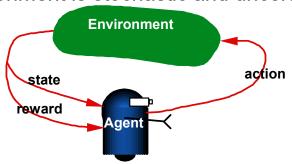
Reinforcement Learning

- Reinforcement Learning Problem
- Temporal Difference Learning
- SARSA
- Function Approximation

Complete Agent

- Temporally situated
- Continual learning and planning
- Object is to affect the environment
- Environment is stochastic and uncertain



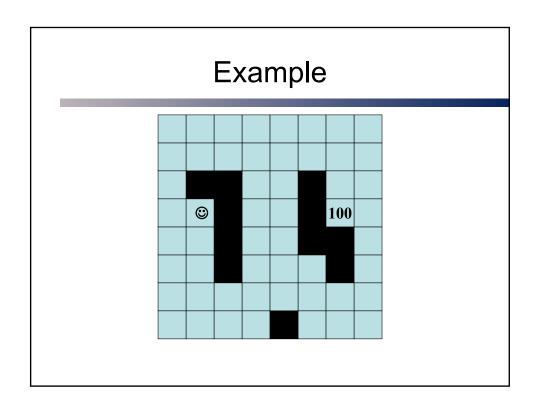
Reinforcement Learning

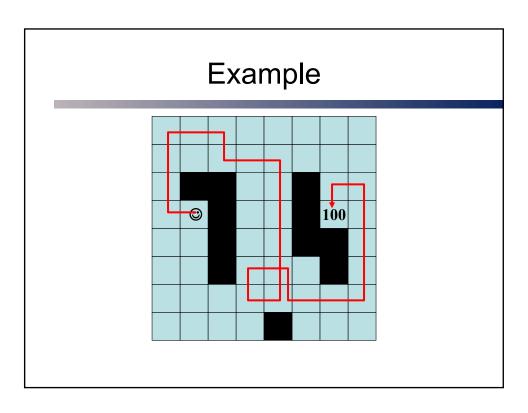
- Goal: learn an optimal strategy for maximizing future reward
 - Agent has little prior knowledge and no immediate feedback
 - Possible need to learn multiple tasks with same sensors/effectors
- Temporal/Action Credit Assignment
 - Rewards and punishments may be delayed sacrifice short term for long term
- Exploration/Exploitation
- Planning and Acting under Uncertainty
 - Can handle deterministic or probabilistic state transitions
 - Possibility that state only partially observable
- Two basic agent designs
 - Agent learns utility function V(s) on states
 - Agent learns action-value Q(s,a) function

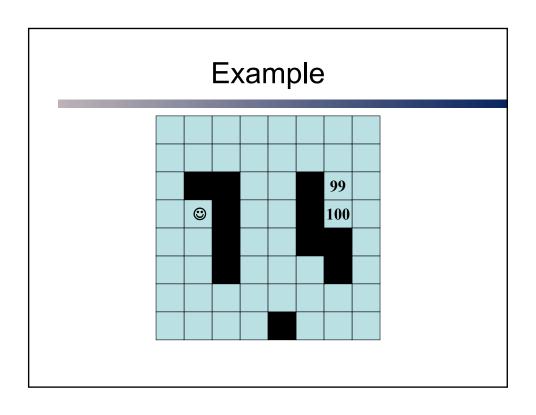
Elements of RL

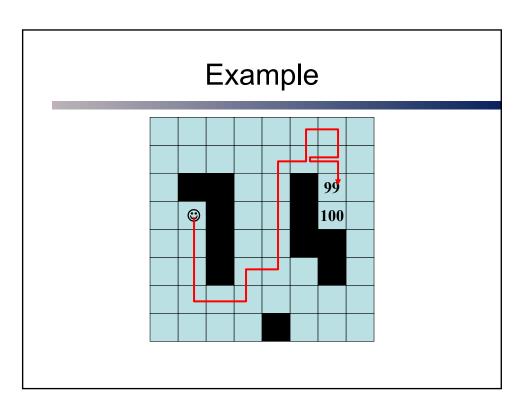


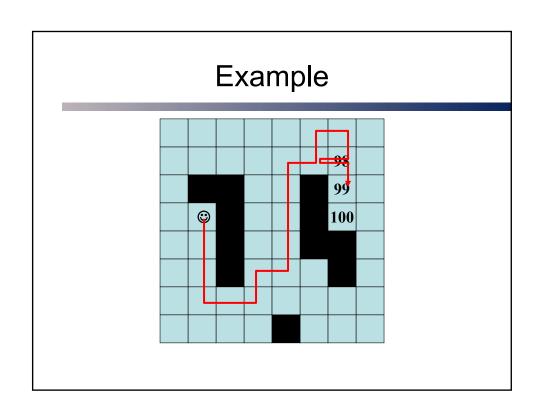
- Policy: what to do
- Reward: what is good
- Value: what is good because it predicts reward
- Model: what follows what

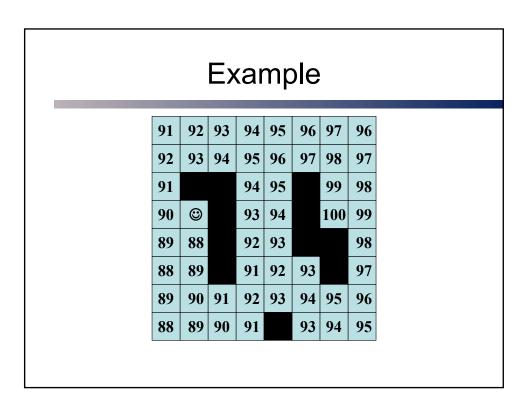








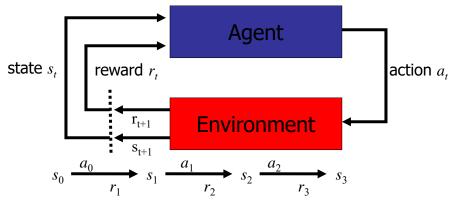




Some Notable RL Applications

- TD-Gammon world's best backgammon program (Tesauro)
- Elevator Control (Crites & Barto)
- Inventory Management 10-15% improvement over industry standard methods (Van Roy, Bertsekas, Lee, and Tsitsiklis)
- Dynamic Channel Assignment high performance assignment of radio channels to mobile telephone calls (Singh and Bertsekas)

Reinforcement Learning Problem



•Goal: Learn to choose actions a_t that maximize future rewards $r_1 + \gamma r_2 + \gamma^2 r_3 + ...$, where $0 < \gamma < 1$ is a discount factor

Approaches for RL

- Model-base Learning
 - These use a model's prediction or distributions of next state and reward in order to calculate optimal actions
 - Policy iteration, value iteration, V(s) based TD learning
- Model-free Learning
 - Relies on samples from the environment to gain experience and learn which actions to take for a state to gain rewards
 - SARSA, Q(s,a) based TD learning, Actor-Critic

Markov Decision Process (MDP)

- S = Finite set of states (|S| = n)
- A = Finite set of actions (|A| = m)
- $Pr(s_p, a, s_{p+1})$ = Transition function (τ)
 - At each time step the agent observes state $s_i \in S$ and chooses action $a \in A$
 - Represented by set of stochastic matrices (non-deterministic)
 - Also includes events occurrences not caused by the agent
- $R(s_p, a, s_{p+1})$ = Reward/cost function
- Markov assumption
 - Current state s_t encapsulates all previous state and action history $h = (s_0, a_0, \dots s_{t-1}, a_{t-1})$
 - Optimal action only depends on the current state and action a_t (and in some cases resultant state s_{t+1})
 - Functions $Pr(s_p a_s s_{t+1})$ and $R(s_p a_t s_{t+1})$ not necessarily known to agent (model based remarks)

Temporal Difference Learning

- Temporal Difference (TD) learning methods
 - Can be used when accurate models of the environment are unavailable – neither state transition function Pr(s,a,s') nor reward function R(s,a,s') are known
 - Can be extended to work with implicit representations of action-value functions
 - Are among the most useful reinforcement learning methods

TD Prediction

- Policy evaluation (the prediction problem): for a given policy π , compute the state-value function V^*
 - The simplest TD method, TD(0)

$$V(s_t) \leftarrow V(s_t) + \alpha [r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

target: an estimate of the return

- TD method is *bootstrapping* by using the existing estimate of the next state $V(s_{t+1})$ for updating $V(s_t)$
- α is the learning rate $(0 \le \alpha \le 1)$

TD(0): Policy Evaluation

Initialize:

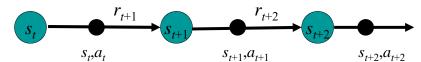
- π = policy to be evaluated
- V(s) = an arbitrary state-value function

Repeat for each episode

- Initialize s
- Repeat for each step of episode
 - $a \leftarrow$ action given by π for s
 - Take action a, observe reward r, and next state s'
 - $V(s) \leftarrow V(s_t) + \alpha [r + \gamma V(s') V(s)]$
 - $s \leftarrow s'$
- Until s is terminal

TD(0): Policy Iteration

• $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$



- The update rule uses a quintuple of events $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$, therefore called SARSA.
- Problem: Unlike in the deterministic case we can not choose a completely greedy policy $\pi(s)=\max_a Q(s,a)$, as due to the unknown transition $\Pr(s_pa,s_{t+1})$ and reward functions and $R(s_pa,s_{t+1})$ we cannot be sure if another action might eventually turn out to be better

ε-greedy policy

- Soft policy: π(s,a) > 0 for all s∈S, a∈A(s)
 Non-zero probability of choosing every possible action
- ε -greedy policy: Most of the time with probability $(1-\varepsilon)$ follow the optimal policy

$$\pi(s) = \max_{a} Q(s,a)$$

but with probability ε pick a random action:

$$\pi(s,a) \geq \varepsilon/|A(s)|$$

 Let ε→0 go to zero as t→∞ for example ε=1/t so that ε-greedy policy converges to the optimal deterministic policy

SARSA Policy Iteration

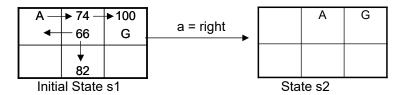
Initialize Q(s,a) arbitrarily:

Repeat for each episode

- Initialize s
- Choose a from using ε-greedy policy derived from Q
- Repeat for each step of episode
 - Take action a, observe reward r, and next state s'
 - $Q(s,a) \leftarrow Q(s,a) + \alpha [r + \gamma Q(s',a') Q(s,a)]$
 - $s \leftarrow s', a \leftarrow a'$
- Until s is terminal

Example

 Consider grid world where goal state (G) yields reward of 100 and other states yield reward of 0.



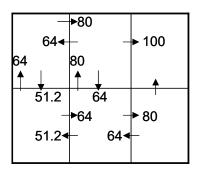
Let learning rate be 0.8.

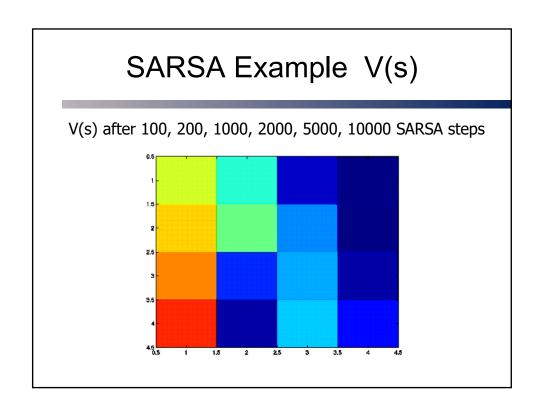
$$Q(s1, right) = 74 + 0.8(0 + max{(Q(s2,left),Q(s2,right), Q(s2,down),Q(s2,up)} - 74)$$

= 74 + 0.8(0 + max{66, 82, 100} - 74)
= 74 + 0.8(26)
= 94.8

Example

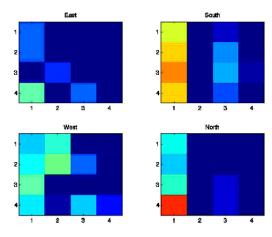
Learned Q(s,a) values





SARSA Example Q(s,a)

Q(s,a) after 100, 200, 1000, 2000, 5000, 10000 SARSA steps



Q-Learning (Off-Policy TD)

- Approximates the optimal value functions
 V*(s) or Q*(s,a) independent of the policy
 being followed.
- The policy determines which state-action pairs are visited and updated

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha [r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Advantages of TD Learning

- TD methods do not require a model of the environment, only experience
- TD methods can be fully incremental
 - You can learn before knowing the final outcome
 - Less memory
 - Less peak computation
 - You can learn without the final outcome
 - From incomplete sequences
- TD converges (under certain assumptions)

Optimality of TD(0)

- Batch Updating: train completely on a finite amount of data, e.g., train repeatedly on 10 episodes until convergence.
 - Compute updates according to TD(0), but only update estimates after each complete pass through the data
- For any finite Markov prediction task, under batch updating, TD(0) converges for sufficiently small α

Extending TD

So far TD uses one-step look-ahead but why not use 2-steps or n-steps look-ahead to update Q(s,a).

2-step look-ahead

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma r_{t+2} + \gamma^2 Q(s_{t+2}, a_{t+2}) - Q(s_t, a_t) \right]$$

N-step look-ahead

$$Q(s_{t}, a_{t}) \leftarrow Q(s_{t}, a_{t}) + \alpha[r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^{n} Q(s_{t+n}, a_{t+n}) - Q(s_{t}, a_{t})]$$

Monte-Carlo method

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{N-1} r_{t+N} - Q(s_t, a_t)]$$
 (compute total reward R^T at the end of the episode)

Temporal Difference Learning

- Drawback of one-step look-ahead:
 - Reward is only propagated back to the successor state (takes long time to finally propagate to the start state)



Initial V:

$$V(s)=0$$
 $V(s)=0$ $V(s)=0$ $V(s)=0$

After first epsiode:

$$V(s)=0$$
 $V(s)=0$ $V(s)=0$ $V(s)=0$ $V(s)=a*100$

After second epsiode:

$$V(s)=0$$
 $V(s)=0$ $V(s)=a*\gamma*100$ $V(s)=a*100+...$

Monte-Carlo Method

Drawback of Monte-Carlo method

- Learning only takes place after an episode terminated
- Performs a random walk until goal state is discovered for the first time as all state-action values seem equal
- It might take long time to find the goal state by random walk
- TD-learning actively explores state space if each action receives a small default penalty

N-Step Return

- Idea: blend TD learning with Monte-Carlo method
- Define:
- $R_t^{(1)} = r_{t+1} + \gamma V_t(s_{t+1})$
- $R_t^{(2)} = r_{t+1} + \gamma r_{t+2} + \gamma^2 V_t(s_{t+2})$
- ...
- $R_t^{(n)} = r_{t+1} + \gamma r_{t+2} + \dots + \gamma^{n-1} r_{t+n} + \gamma^n V_t(s_{t+n})$
- The quantity $R_t^{(n)}$ is called the n-step return at time t.

$TD(\lambda)$

TD(λ): use average of n-step returns for backup

$$\begin{split} R_t^{\lambda} &= (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^n \\ &= (1 - \lambda) \sum_{n=1}^{T-t-1} \lambda^{n-1} R_t^n + \lambda^{T-t-1} R_t \end{split}$$

- *n*-step backup: $V_t(s_t) \leftarrow V_t(s_t) + \alpha [R_t^{(n)} V_t(s_t)]$
 - (if s_T is a terminal state)
- The weight of the *n*-step return decreases by a factor of λ, 0 < λ < 1
 - TD(0): one-step temporal difference method
 - TD(1): Monte-Carlo method

Online (passive) $TD(\lambda)$

Initialize π an MDP, V a table of values, and s, a, r previous state, action, and reward.

Repeat for each episode

- Repeat
 - If s' is new then V(s') = r'
 - If s is not new then do

•
$$V(s) = r + \gamma((1 - \lambda)V(s) + \lambda V(s'))$$

- $s,a,r \leftarrow s' \pi(s'),r'$
- Until s is terminal

Another View (Sutton)

- Initialize V(s) arbitrarily and e(s)=0, for all $s \in S$
- Repeat (for each episode):
 - Initialize s
 - Repeat (for each step of episode):
 - $a \leftarrow$ action given by π for s
 - Take action a, observe reward, r, and next state, s'
 - $\delta \leftarrow r + \gamma V(s') V(s)$
 - $e(s) \leftarrow e(s)+1$
 - For all s:
 - $V(s) \leftarrow V(s) + \alpha \delta e(s)$
 - $e(s) \leftarrow \gamma \lambda e(s)$
 - $s \leftarrow s'$
 - until s is terminal

Q-Learning

Initialize Q a table of state-action pairs, and s,a,r previous state, action and reward.

Repeat for each episode

- Repeat
 - If s' is new then Q(s',a) = r'
 - If s is not null then do
 - $Q(s,a) = r + \gamma \left((1 \lambda) \max_{a} Q(s,a) + \lambda Q(s',a') \right)$
 - $\bullet s,a,r \leftarrow s', \max_{a'} Q(s',a'),r'$
- Until s is terminal

Q-Learning with Exploration

Initialize Q a table of state-action pairs, and s,a,rprevious state, action and reward, and N_{sa} a table of frequencies for state-action pairs.

Repeat for each episode

- Repeat
 - If s' is new then Q(s',a) = r'
 - If s is not null then do
 - Increment N_{sa}(s,a)

$$Q(s,a) = r + N_{sa}(s,a)\gamma(1-\lambda)\max Q(s,a) + \lambda Q(s',a')$$

- $Q(s,a) = r + N_{sa}(s,a)\gamma(1-\lambda)\max_{a}Q(s,a) + \lambda Q(s',a')$ $S,a,r \leftarrow S',\max_{a'}f(Q(s',a'),N_{sa}(s',a')),r'$
- Until s is terminal

Exploratory Function

$$f(u,n) = \begin{cases} R + & \text{if } n < N_e \\ u & \text{otherwise} \end{cases}$$

- Where R⁺ is an optimistic estimate of the best possible reward obtainable in any state
- N_e is a fixed threshold
- Mitchell alternative:

$$P(a_i \mid S) = \frac{k^{Q(s,a_i)}}{\sum_{j} k^{Q(s,a_j)}}$$

Function Approximation

- So far we assumed that the functions Q(s,a) is represented as a table.
- Limited to problems with a small number of states and actions
- For large state spaces a table based representation requires large memory and large data sets to fill them accurately
- Generalization: Use any supervised learning algorithm to estimate Q(s,a) from a limited set of action value pairs
 - Neural Networks (Neuro-Dynamic Programming)
 - Linear Regression
 - Nearest Neighbors

Function Approximation

• Minimize the mean-squared error between training examples $Q_t(s_t, a_t)$ and the true value function $Q^{\pi}(s, a)$

$$\sum_{s \in S} Pr(s) [Q^{\pi}(s, a) - Q_t(s_t, a_t)]^2$$

- Notice that during policy iteration P(s) and $Q^{\pi}(s,a)$ change over time
- Parameterize $Q^{\pi}(s,a)$ by a vector $\boldsymbol{\theta}=(\theta_1,...,\theta_n)$ for example weights in a feed-forward neural network

Stochastic Gradient Descent

 Use gradient descent to adjust the parameter vector θ in the direction that reduces the error for the current example

$$\theta_{t+1} = \theta_t + \alpha [Q^{\pi}(s_t, a_t) - Q_t(s_t, a_t)] \Delta_{\theta_t} Q_t(s_t, a_t)$$

 The target output q_t of the t-th training example is not the true value of QP but some approximation of it.

$$\theta_{t+1} = \theta_t + \alpha [v_t - Q_t(s_t, a_t)] \Delta_{\theta_t} Q_t(s_t, a_t)$$

Still converges to a local optimum if

$$E(v_t) = Q^{\pi}(s_t, a_t)$$
 if $a \rightarrow 0$ for $t \rightarrow \infty$

Example: AlphaZero

Subtleties and Ongoing Research

- Scalability
- Generalize utilities from visited states to other states (inductive learning)
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous actions, states
- Inverse Reinforcement Learning

Summary Reinforcement Learning

- Different Learning Problem: Learn policy for action selection that maximizes future reward
- Solution: Learn value function
- Two approaches: Recursive backups and supervised learning. Q-Learning is an example of the former.
- TD(λ): mixes both of these approaches