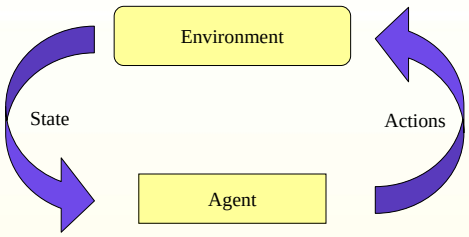
Artificial Intelligence

Reinforcement Learning

MONTANA STATE UNIVERSITY

Agent-Based Systems



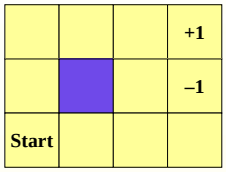
MONTANA STATE UNIVERSITY

Sample Problems

- A simple navigation problem.
- The race-track problem.
- The evasive maneuvers problem.

MONTANA STATE UNIVERSITY

Simple Navigation



Goal: Travel from Start to square with maximum payoff in minimum number of steps.

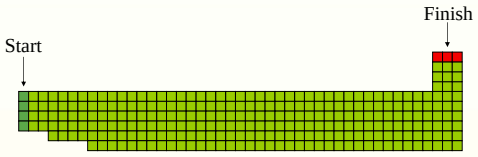
MONTANA STATE UNIVERSITY

Two Versions

- Deterministic Navigation
  - Actions: up, down, left, right
  - All actions yield expected behavior.
  - Striking a wall or obstacle yields no change in position.
- Nondeterministic Navigation
  - Initially same as deterministic.
  - Expected transition has probability 0.8
  - Left of expected 0.1, right of expected 0.1

MONTANA STATE UNIVERSITY

Race Track Problem



MONTANA STATE UNIVERSITY

Race Track Problem

- Acceleration is either horizontal or vertical.
- With probability p, actual accelerations are zero, independent of intended acceleration.
- Dynamics given by the following.

With probability p

$$x_{t+1} = x_t + \dot{x}_t$$
$$y_{t+1} = y_t + \dot{y}_t$$
$$\dot{x}_{t+1} = \dot{x}_t$$
$$\dot{y}_{t+1} = \dot{y}_t$$

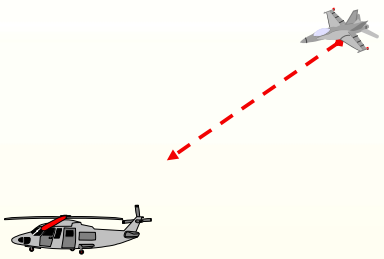
With probability $1-p$

$$x_{t+1} = x_t + \dot{x}_t + a_t^x$$
$$y_{t+1} = y_t + \dot{y}_t + a_t^y$$
$$\dot{x}_{t+1} = \dot{x}_t + a_t^x$$
$$\dot{y}_{t+1} = \dot{y}_t + a_t^y$$

MONTANA STATE UNIVERSITY

Mountains Minds

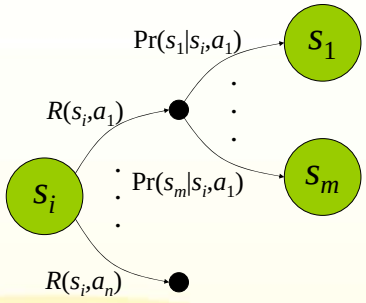
Evasive Maneuvers



MONTANA STATE UNIVERSITY

Mountains Minds

Markov Decision Process



MONTANA STATE UNIVERSITY

Mountains Minds

Markov Decision Process

- ***Def*:** $M = \langle S, A, T, R, \gamma \rangle$ is a *Markov decision process* (MDP) where
  - S is a finite set of states of the environment
  - A is a finite set of actions
  - $T: S \times A \rightarrow \pi(S)$ is a state transition function where $T(s,a,s^*)$ is the probability of ending state s^* given agent takes action a in state s.
  - R: $S \times A \rightarrow \Re$ is a reward function where $R(s,a)$ is the expected reward for taking action a in state s.
  - $\gamma \in [0,1]$ is a discount factor.

Markov Decision Process

- ***Def*:** A *policy* is a specification of how an agent will act in all states.
  - A policy can be stationary or nonstationary.
- ***Def*:** a *stationary* policy π: $S \rightarrow A$ specifies for each state an action to be taken, independent of time step.
- ***Def*:** a *nonstationary* policy is a sequence of state-action mappings, indexed by time.
  - Given $\delta = \langle \pi_t, \ldots, \pi_{t+\tau} \rangle$, π_i is used to choose an action in step i as a function of that state.

Evaluating an MDP

- Given an MDP and a policy, we can evaluate the expected, long-run value an agent could expect to gain from following it.
- Let $V^{\pi}(s)$ be the expected, discounted future reward for starting in state s and executing stationary policy π indefinitely.

Evaluating an MDP

- Then the infinite horizon value of policy π from state *s* is defined as

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{\forall s^* \in S} T(s, \pi(s), s^*) V^{\pi}(s^*)$$

- This defines a set of simultaneous equations, one for each state, from which the value of π can be found through Gaussian elimination or similar method.

MONTANA STATE UNIVERSITY

Mountains Minds

Evaluating an MDP

- Given a value function, *V*, we can derive the policy from that function, assuming a *greedy* policy as follows:

$$\pi_V(s) = \arg\max_{\forall a \in A} \left[R(s, a) + \gamma \sum_{\forall s^* \in S} T(s, a, s^*) V(s^*) \right]$$

MONTANA STATE UNIVERSITY

Mountains Minds

Evaluating an MDP

- In general, given an initial state *s*, we want to execute the policy π that maximizes $V^{\pi}(s)$.
- There exists a stationary policy in a discounted, infinite-horizon MDP that is optimal for every starting state.
- The value function for this policy is defined by.

$$V'(s) = \max_{\forall a \in A} \left[R(s, a) + \gamma \sum_{\forall s^* \in S} T(s, a, s^*) V'(s^*) \right]$$

- Any greedy policy with respect to this function is optimal.

MONTANA STATE UNIVERSITY

Mountains Minds

Evaluating an MDP

- Note these equations are not linear (due to the max term); therefore, Gaussian elimination will not work.
- Two common algorithms for solving MDPs are
  - Value Iteration
  - Policy Iteration
- An approach based on linear programming is also possible.

MONTANA STATE UNIVERSITY
Mountains Minds
16

Dynamic Programming

- A mathematical optimization technique for making a sequence of interrelated decisions.
- Generally must be tailored to every problem solved.
- Tailoring focuses on defining value functions (like what we have seen).

MONTANA STATE UNIVERSITY
Mountains Minds
17

Characteristics of DP

- Problem divided into stages with a “policy” decision at each stage.
- Each stage has several states associated with it.
- Following policy results in state transition.
- DP finds an optimal policy.

MONTANA STATE UNIVERSITY
Mountains Minds
18

Characteristics of DP

- ***Principle of Optimality*** says, given the current stage, an optimal policy for the remaining stages is independent of the policy adopted for previous stages.
- Typical approach starts by finding an optimal policy for last stage.
- Set up recursive relationship for previous optimal policies—*Bellman equations*.
- Applying recursive relationship provides update procedure to move backward through stages.

MONTANA STATE UNIVERSITY

Mountains Minds

Value Iteration

- Compute a sequence V_t using an auxiliary function $Q_t(s,a)$ which is the t-step value of starting in state s, taking action a, and continuing with the optimal (t–1)-step nonstationary policy.
- Value iteration terminates when the maximum difference between two successive value functions is less than some pre-determined ε, called the *Bellman error magnitude*.

MONTANA STATE UNIVERSITY

Mountains Minds

Value Iteration

function ValueIteration(MDP, ε)
  for each $s \in S$, $V_o := 0$
  $t \leftarrow 0$
  loop
    $t \leftarrow t + 1$
    for each $s \in S$
      for each $a \in A$
        $Q_t(s,a) \leftarrow R(s,a) + \gamma \Sigma_{s^*} T(s,a,s^*) V_{t-1}(s^*)$
      $\pi_t(s) \leftarrow \text{argmax}_a Q_t(s,a)$
      $V_t(s) \leftarrow Q_t(s,\pi_t(s))$
  until $\max_s |V_t(s) - V_{t-1}(s)| < \varepsilon$
  return π_t

MONTANA STATE UNIVERSITY

Mountains Minds

Policy Iteration

- Let π_o be the greedy policy for the initial value function V_o.
- Let $V_1 = V^{\pi}$ be the value function for π_0.
- If we alternate between finding the optimal policy for a particular value function and deriving a new value function for that policy, we converge on the optimal policy for the MDP.

MONTANA STATE UNIVERSITY

Mountains Minds

Policy Iteration

```
function PolicyIteration(MDP)
    for each s ∈ S, V_o := 0
    t ← 0
    loop
        t ← t + 1
        for each s ∈ S
            for each a ∈ A
                Q_t(s,a) ← R(s,a) + γ Σ_s* T(s,a,s*) V_{t-1}(s*)
            π_t(s) := argmax_a Q_t(s,a)
        V_t ← EvalMDP(π_t,MDP)
    until V_t(s) = V_{t-1}(s) for all s
    return π_t
```

MONTANA STATE UNIVERSITY

Mountains Minds

Policy Iteration

- The function EvalMDP solves the system of linear equations given by

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{\forall s^* \in S} T(s, \pi(s), s^*) V^{\pi}(s^*)$$

MONTANA STATE UNIVERSITY

Mountains Minds

Example

Actions

G	1	2	3
4	5	6	7
8	9	10	11
12	13	14	G

$\gamma = 0.7$

$R(s,a) = -1$ for all transitions

Assume initial policy is to move randomly. If the agent bumps into a wall, there is no change in state.

MONTANA STATE UNIVERSITY

25

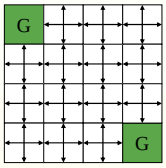
Mountains Minds

Iteration $k = 0$

V_k for current policy

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

Greedy policy w.r.t V_k



MONTANA STATE UNIVERSITY

26

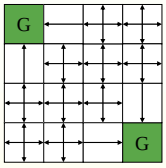
Mountains Minds

Iteration $k = 1$

V_k for current policy

0.0	–1.0	–1.0	–1.0
–1.0	–1.0	–1.0	–1.0
–1.0	–1.0	–1.0	–1.0
–1.0	–1.0	–1.0	0.0

Greedy policy w.r.t V_k



MONTANA STATE UNIVERSITY

27

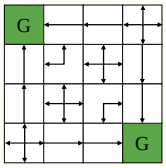
Mountains Minds

Iteration $k = 2$

V_k for current policy

0.0	–1.0	–1.7	–1.7
–1.0	–1.7	–1.7	–1.7
–1.7	–1.7	–1.7	–1.0
–1.7	–1.7	–1.0	0.0

Greedy policy w.r.t V_k



MONTANA STATE UNIVERSITY

28

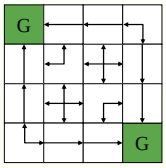
Mountains Minds

Iteration $k = 3$

V_k for current policy

0.0	–1.0	–1.7	–2.2
–1.0	–1.7	–2.2	–1.7
–1.7	–2.2	–1.7	–1.0
–2.2	–1.7	–1.0	0.0

Greedy policy w.r.t V_k



MONTANA STATE UNIVERSITY

29

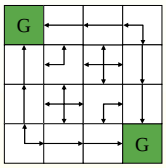
Mountains Minds

Iteration $k = \infty$

V_k for current policy

0.0	–1.0	–1.7	–2.2
–1.0	–1.7	–2.2	–1.7
–1.7	–2.2	–1.7	–1.0
–2.2	–1.7	–1.0	0.0

Greedy policy w.r.t V_k



MONTANA STATE UNIVERSITY

30

Mountains Minds

Synchronous DP

- The previous algorithm for Value Iteration corresponds to synchronous dynamic programming.
- The value function is updated for each state at each step of the algorithm.
- The algorithm is synchronous because only values from step *t* are used in step *t*+1.

$$V_{t+1}(i) = \max_{\forall a \in A} \left[R(i,a) + \gamma \sum_{\forall j \in S} T(i,a,j) V_t(j) \right]$$

MONTANA STATE UNIVERSITY
Mountains Minds
31

Complexity of Synchronous DP

- Given *n* states with *m* = the largest number of admissible actions for a state, then each iteration requires
  - $O(mn^2)$ operations for nondeterministic MDP
  - $O(mn)$ operations for deterministic MDP
- Each iteration is called "backing up" since we effectively back up through the stages one stage at a time with each iteration.

MONTANA STATE UNIVERSITY
Mountains Minds
32

Convergence

- Synchronous DP converges to *V** in *undiscounted* stochastic MDPs (i.e., when $\gamma = 1$) under the following conditions:
  - The initial cost of every goal state is zero.
  - There exists at least one proper policy.
  - All policies that are not proper incur infinite cost for at least one state.
- Given the complexity and convergence restrictions, alternative DP approaches were developed.

MONTANA STATE UNIVERSITY
Mountains Minds
33

Gauss-Seidel DP

- Similar to synchronous DP
- Backups occur in an ordered "sweep" of the states.
- For a particular state, use most recent value of successor states in update.
- Thus some states will use previous iterations, some will use current.

MONTANA STATE UNIVERSITY
Mountains Minds

Gauss-Seidel DP

- Assume the states are numbered according to some total order.

$$V_{t+1}(i) = \max_{\forall a \in A} \left[R(i,a) + \gamma \sum_{\forall j \in S} T(i,a,j) V(j) \right]$$

where

$$V(j) = \begin{cases} V_{t+1}(j) & \text{if } j < i \\ V_t(j) & \text{otherwise} \end{cases}$$

MONTANA STATE UNIVERSITY
Mountains Minds

Gauss-Seidel DP

- Gauss-Seidel DP converges under the same conditions as synchronous DP, except it also converges for the discounted case (i.e., $\gamma < 1$).

MONTANA STATE UNIVERSITY
Mountains Minds

Asynchronous DP

- Similar to Gauss-Seidel, except the states are not updated in complete, successive sweeps.
- At each stage, select a *subset* of states at random to be updated.
- A new subset is selected at each stage.
- Simplest case, the size of the subset is 1.

MONTANA STATE UNIVERSITY
Mountains Minds
37

Asynchronous DP

- Update procedure

$$V_{t+1}(i) = \begin{cases} \max_{\forall a \in A} \left[R(i,a) + \gamma \sum_{\forall j \in S} T(i,a,j) V(j) \right] & i \in S_t^{sub} \\ V_t(i) & \text{otherwise} \end{cases}$$

where

$$V(j) = \begin{cases} V_{t+1}(j) & \text{if } j < i \\ V_t(j) & \text{otherwise} \end{cases}$$

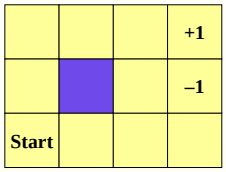
MONTANA STATE UNIVERSITY
Mountains Minds
38

Asynchronous DP

- Discounted asynchronous dynamic programming converges to *V** "*in the limit*."
- This means that the cost/reward of each state is backed up infinitely often.

MONTANA STATE UNIVERSITY
Mountains Minds
39

Nondeterministic Navigation



Model $M_{ij}^{a} \equiv P(j \mid i, a) =$ probability that doing a in i leads to j.

Each state has a reward $R(i) = \begin{cases} -0.04 \text{ for non-terminal states} \\ \pm 1 \text{ for terminal states} \end{cases}$

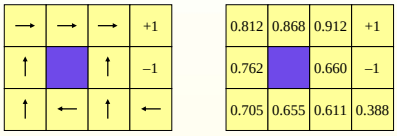
MONTANA STATE UNIVERSITY

40

Mountains Minds

Solving the MDP

- In search problems, aim is to find an optimal *sequence*.
- In MDPs, aim is to find an optimal *policy* (i.e., the best action for every possible state).
- Optimal policy and state values for example:



MONTANA STATE UNIVERSITY

41

Mountains Minds

Utility

- In *sequential decision problems*, preferences are expressed between sequences of states.
- Usually use an additive utility function.
  - $U(s_1,\ldots,s_n) = R(s_1) + \ldots + R(s_n)$
- Utility of a state (a.k.a. its value) is defined to be $U(s_i)$ = expected sum of rewards until termination, assuming optimal actions.
- Given the utility of states, choosing the best action is just MEU (maximum expected utility): choose action such that the expected utility of the immediate successor is highest.

MONTANA STATE UNIVERSITY

42

Mountains Minds

Reinforcement Learning

- A learning technique in which
  - An agent needs to make a sequence of decisions.
  - The agent interacts with the environment to sense state and make those decisions.
  - The agent learns a policy of optimal actions based on trial-and-error search of the search space.
  - The only information provided to the agent is a measure of "goodness" (sometimes delayed) of the action or sequence of actions.

MONTANA STATE UNIVERSITY
Mountains Minds
43

Not Supervised Learning

- RL does not present a set of input/output pairs.
- The agent chooses actions and receives reinforcements, not correct actions.
- The environments are usually non-deterministic.
- Many models based on offline learning, but online performance is important.
- System must explore space of state/action pairs.

MONTANA STATE UNIVERSITY
Mountains Minds
44

When RL = SL

- RL becomes Supervised Learning when
  - There are two possible actions, *and*
  - The reinforcement signal is Boolean, *and*
  - The world is deterministic.
- When action *a* is chosen in situation *s*,
  - If $r = 1$, then $f(s) = a$
  - If $r = 0$, then $f(s) = \neg a$

MONTANA STATE UNIVERSITY
Mountains Minds
45

Models of Optimal Behavior

- RL attempts to maximize one of:
  - Finite horizon reward

$$E\left(\sum_{t=0}^{k} r_t\right)$$

  - Infinite horizon discounted reward

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

  - Average reward

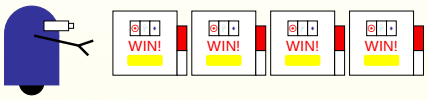
$$\lim_{k\to\infty} E\left(\frac{1}{k}\sum_{t=0}^{k} r_t\right)$$

MONTANA STATE UNIVERSITY

Mountains Minds

K-Armed Bandits

- The world has only one state.
- Actions pay off independently on each trial.
- The world is stationary.
- Question is how to trade exploration and exploitation.

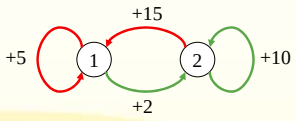


MONTANA STATE UNIVERSITY

Mountains Minds

Delayed Reinforcement

- Actions determine immediate reward.
- Actions change situation.
- Can be treated as a network of bandit problems.
- Choose actions to maximize some measure of long run performance



MONTANA STATE UNIVERSITY

Mountains Minds

Real-Time DP

- Up until now, *all* of the algorithms have been run "off-line."
- Thus, the algorithms require knowledge of the underlying model (i.e., MDP).
- All state transitions and transition probabilities must be known.

MONTANA STATE UNIVERSITY
Mountains Minds
49

Real-Time DP

- RTDP modifies asynchronous DP to run "on-line."
- Since RTDP is on-line, the underlying model need not be known *a priori*.
- Approach
  - Agent always follows a policy that is greedy w.r.t. most recent estimate of V^*.
  - Between execution of successive states, cost/reward of state s_t (and others) is backed up.

MONTANA STATE UNIVERSITY
Mountains Minds
50

Real-Time DP

- ***Def*:** Trial-based RTDP is RTDP where trials are initiated such that every state will, with probability one, be a start state infinitely often in an infinite number of trials.
- Given this definition, three theorems can be proven. (Note: proofs will not be given.)

MONTANA STATE UNIVERSITY
Mountains Minds
51

RTDP Performance

- ***Theorem 1*:** For any discounted Markov decision process, and any initial evaluation function, trial-based RTDP converges with probability one.
  - **Note:** This is another convergence result that is "in the limit."

RTDP Performance

- ***Theorem 2*:** In undiscounted, non-deterministic MDPs, trial-based RTDP converges with probability one under the following conditions:
  - The initial cost of every goal state is zero.
  - There is at least one proper policy.
  - All policies that are not proper incur infinite cost for at least one state.

RTDP Performance

- ***Theorem 3*:** In undiscounted, nondeterministic MDPs, trial-based RTDP with the initial state of each trial restricted to a set of start states, converges with probability one to V^* on the set of relevant states, and the agent's policy converges to an optimal policy on the relevant states under the following conditions:
  - The initial cost of every goal state is zero.
  - There is at least one proper policy.
  - All immediate costs incurred by transitions from non-goal states are positive.
  - The initial costs of all states are non-overestimating.

Prioritized Sweeping

- Problem: RTDP (and other model-based methods) update all states or states at random, even if they are not "interesting."
- Solution: What if, rather than updating randomly (or systematically), we focus on states expected to be "most useful?"
- This leads to idea known as *prioritized sweeping*.

MONTANA STATE UNIVERSITY

Mountains Minds

Prioritized Sweeping (Approach)

- Suppose we are in state *s*.
- Suppose we choose action *a*.
- We then observe state *s'* and get reward *r*.
- We can now add this information to the model we are learning and update the value function

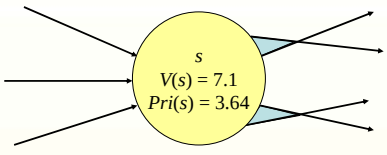
$$V(s) \leftarrow \max_a \left(R(s,a) + \gamma \sum_{s'} T(s,a,s') V(s') \right)$$

- Then we do a little more computation to focus updates.

MONTANA STATE UNIVERSITY

Mountains Minds

Extra Information in Model



- Each state has a *priority*.
- Each state remembers its *predecessors*.

MONTANA STATE UNIVERSITY

Mountains Minds

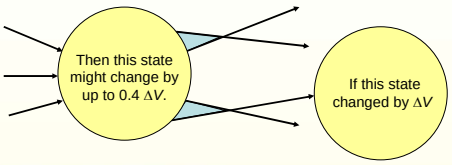
Using Priority Measures

- On each step, perform backups on k states that currently have the highest priority (e.g., $k = 50$).
- For each state
  - Remember the current value of the state: $V_{old} \leftarrow V(s)$.
  - Update the state's value using the Bellman equation.
  - Set the state's priority back to 0.
  - Compute the value change: $\Delta V \leftarrow |V_{old} - V(s)|$.
  - Use ΔV to possibly change the priorities of some of s's predecessors.

MONTANA STATE UNIVERSITY
Mountains Minds
58

Computing the Priorities

- If we've done a backup on state s and it has changed by amount ΔV, then the immediate predecessors of s should be informed of this event.



MONTANA STATE UNIVERSITY
Mountains Minds
59

Global Behavior

- When a real-world transition is "surprising," lots of computation is directed to propagate this new information.
- When a real-word transition is "boring," the computation continues in the most deserving part of the space.

MONTANA STATE UNIVERSITY
Mountains Minds
60

Model-Free RL

- Offline DP requires knowledge of underlying model
  - Transitions
  - Transition probabilities
- RTDP need not have underlying model, but typically does.
- The approach to model-free RTDP is called *Q*-learning.

MONTANA STATE UNIVERSITY
Mountains Minds
61

Q-Learning

- Agent maintains current estimate of optimal *Q* values for every admissible state-action pair.
- *Q* values are estimated based on experience interacting with environment.
- For an encounter, agent behaves according to a strategy greedy in $Q(s,a)$.
- Sequence of states with actions taken is stored.

MONTANA STATE UNIVERSITY
Mountains Minds
62

Q-Update Rule

- Let $Q_t(s,a)$ be the *Q* value at time *t* when action *a* is performed in state *s*.
- Let $\alpha_t(s,a) \in [0,1]$ be the learning rate for state-action pair $\langle s,a \rangle$ at time *t*.
- Let $\gamma \in [0,1]$ be a discount rate.
- Let *s** indicate the "successor" state.

$$Q_{t+1}(s,a) = [1-\alpha_t(s,a)]Q_t(s,a) + \alpha_t(s,a)[R(s,a) + \gamma Q_t(s^*, \pi(s^*))]$$

MONTANA STATE UNIVERSITY
Mountains Minds
63

Learning Rate

- Observe that an individual learning rate is associated with each state-action pair.
- Note further that the learning rates are not constant.
- For convergence, the following is required.

$$\sum_{t=1}^{\infty} \alpha_t(s,a) = \infty \qquad \sum_{t=1}^{\infty} \alpha_t(s,a)^2 < \infty \quad \Rightarrow \quad \alpha_t(s,a) = \frac{\alpha_0(s,a)\tau}{\tau + n_t(s,a)}$$

MONTANA STATE UNIVERSITY
Mountains Minds
64

Q-Learning Algorithm

```
APPLY-Q-POLICY(Q,s) returns a;
        curr-Q := –∞;
        for all a ∈ A
                if Q(s,a) > curr-Q then
                        curr-Q := Q(s,a);
                        curr-a := a;
        return curr-a;
```

MONTANA STATE UNIVERSITY
Mountains Minds
65

Q-Learning Algorithm

```
BUILD-SEQUENCE(s,Q) returns Seq;
        Seq = [s];
        while LAST(Seq) ∉ Terminal-State do
                a := APPLY-Q-POLICY(Q,LAST(Seq));
                APPEND(Seq,a);
                s := RESULT(APPLY(a,LAST(Seq));
                APPEND(Seq,s);
        APPEND(Seq,⊥);
        return Seq;
```

MONTANA STATE UNIVERSITY
Mountains Minds
66

Q-Learning Algorithm

```
UPDATE-Q-VALUES(Seq,Q) returns Q;
    ⟨s,a⟩ = FIRST-PAIR(Seq);
    while a ≠ ⊥ do
        ⟨s',a'⟩ = NEXT-PAIR(Seq);
        if a' ≠ ⊥ then
            Q(s,a) := (1 – α)Q(s,a) +
                α(R(s,a) + γ Q(⟨s',a'⟩));
        else
            Q(s,a) := (1 – α)Q(s,a) + αR(s,a);
        ⟨s,a⟩ = ⟨s',a'⟩
    return Q;
```


Q-Learning Convergence

- Watkins proved that, assuming each state-action pair is visited and updated infinitely often (i.e., in the limit), $Q(s,a)$ will converge to the optimum Q values with probability 1.
- To ensure convergence, each action must be considered in each state. Therefore, we must permit other than the Greedy policy to be applied (occasionally).

Exploration Vs. Exploitation

- Assign a probability distribution to the admissible actions.
- Typically, use a Boltzmann distribution

$$\Pr(a \mid s) = \frac{e^{Q_t(s,a)/T}}{\sum_{a' \in A} e^{Q_t(s,a')/T}}$$

Markov Games

- ***Def***: $M = \langle S, A_1, A_2, T, R_1, R_2, \gamma \rangle$ is a *Markov game* where
  - S is a finite set of states of the environment
  - A_1 is a finite set of actions for player 1; A_2 is a finite set of actions for player 2
  - $T: S \times A_1 \times A_2 \rightarrow \pi(S)$ is a state transition function where $T(s,a_1,a_2,s^*)$ is the probability of ending state s^* given P1 takes action a_1 and P2 takes action a_2 in state s.
  - R_1: $S \times A_1 \times A_2 \rightarrow \Re$ is a reward function where $R_1(s,a_1,a_2)$ is the expected reward to P1 for actions a_1 and a_2 in state s. R_2: $S \times A_1 \times A_2 \rightarrow \Re$ is a reward function where $R_2(s,a_1,a_2)$ is the expected reward to P2 for actions a_1 and a_2 in state s.
  - $\gamma \in [0,1]$ is a discount factor.

MONTANA STATE UNIVERSITY

Mountains Minds

Value Function for MGs

$$V^{\pi_1}(s_i) = E_{\pi_1}\left[\sum_{t=0}^{\infty} \gamma^t R_1(s_t, \pi_1(s_t), \pi_2(s_t)) \mid s_0 = s_i\right]$$

$$V^{\pi_2}(s_i) = E_{\pi_2}\left[\sum_{t=0}^{\infty} \gamma^t R_2(s_t, \pi_1(s_t), \pi_2(s_t)) \mid s_0 = s_i\right]$$

For zero-sum Markov games:

$$V^{\pi}(s_i) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, \pi_1(s_t), \pi_2(s_t)) \mid s_0 = s_i\right]$$

$$V^{\pi}(s_i) \approx Q^V(s_i, a_1, a_2)$$

$$= R(s_i, a_1, a_2) + \gamma \sum_{s_j} \Pr(s_j \mid s_i, a_1, a_2) V(s_j)$$

MONTANA STATE UNIVERSITY

Mountains Minds

Determining a Policy

- We can now establish a combined policy, π, based on the current estimate of the value function.
- Select $\pi(s_i) = \langle a_1, a_2 \rangle$ such that

$$Q^{V^*(s_i)}(s_i, \pi_1(s_i), \pi_2(s_i)) = \sum_{a_1 \in A_1} \sum_{a_2 \in A_2} \Pr(a_1 \mid s_i) \Pr(a_2 \mid s_i) Q^V(s_i, a_1, a_2)$$

Note: We need to find the appropriate probabilities for the associated mixed strategies.

MONTANA STATE UNIVERSITY

Mountains Minds

Linear Programming (revisited)

Player 1

$$\text{minimize } V^{\pi}(s_i) \text{ subject to:}$$
$$\sum_{a_1 \in A_1} \Pr(a_1 \mid s_i) = 1, \text{ where } \Pr(a_1 \mid s_i) \geq 0$$
$$\forall a_1 \in A_1, \sum_{a_2 \in A_2} \Pr(a_1 \mid s_i) Q^V(s_i, a_1, a_2) - V^{\pi}(s_i) \leq 0$$

Player 2

$$\text{maximize } V^{\pi}(s_i) \text{ subject to:}$$
$$\sum_{a_2 \in A_2} \Pr(a_2 \mid s_i) = 1, \text{ where } \Pr(a_2 \mid s_i) \geq 0$$
$$\forall a_2 \in A_2, \sum_{a_1 \in A_1} \Pr(a_2 \mid s_i) Q^V(s_i, a_1, a_2) - V^{\pi}(s_i) \geq 0$$

MONTANA STATE UNIVERSITY

Mountains Minds

Learning in Markov Games

- M. Littman explored using *Q*-Learning in Markov Games.
- L. Baird explored using TD-Learning (see later) in differential Markov Games.
- J. Sheppard explored using memory-based *Q*-Learning in differential Markov Games.

$$Q_{t+1}(s, a_1, a_2) = \alpha_t(s, a_1, a_2)[R(s, a_1, a_2) + \gamma Q_t(s^*, \pi_1(s^*), \pi_2(s^*))] + [1 - \alpha_t(s, a_1, a_2)]Q_t(s, a_1, a_2)$$

MONTANA STATE UNIVERSITY

Mountains Minds

Applications to Control

- Autonomous vehicle path planning.
  - Obstacle avoidance/evasive maneuvers
  - Shortest path
- Robotic arm object manipulation.
  - Inverted pendulum
  - Devil sticking
- Optimal control
  - Elevator control
  - Manufacturing process control

MONTANA STATE UNIVERSITY

Mountains Minds

Applications to Games

- Checkers & Backgammon
- Robot Soccer
- Simple games of pursuit/evasion
- Full differential games
  - Continuous state space
  - Continuous action space
  - Games of kind (discrete, terminal payoff)
  - Games of degree (continuous payoff)

$$\rho_i = \int_0^T g_i^t(s^t, a^t)dt + v_i^T(s^T), \qquad i = 1, \ldots p$$

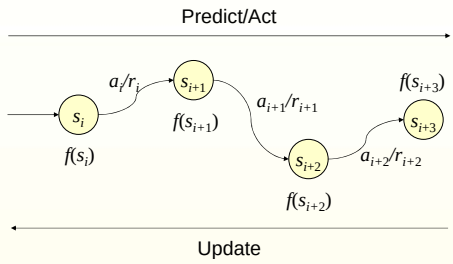
MONTANA STATE UNIVERSITY
Mountains Minds
76

Temporal Difference Learning

- Applied in "multi-step prediction problems."
- What are we predicting?
  - The optimal value function.
- *Q*-Learning is a form of TD-Learning (as we will see).
- TD-Learning is an extension of supervised learning by gradient descent.

MONTANA STATE UNIVERSITY
Mountains Minds
77

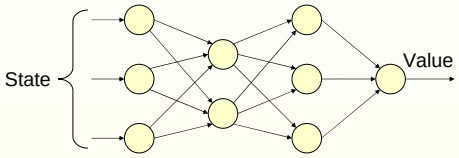
Temporal Difference Learning



MONTANA STATE UNIVERSITY
Mountains Minds
78

Temporal Difference Learning

Connectionist:



$$\Delta w_t = \eta(P_{t+1} - P_t)\sum_{k=1}^{t}\lambda^{t-k}\nabla_w P_k$$

MONTANA STATE UNIVERSITY

Mountains Minds

Gradient Descent

- Assume our prediction depends on a vector of modifiable "weights" *w* and a vector of state variables *x*.
- Supervised learning uses a set of paired prediction vectors and outcome vectors to modify the weights.
- Weights are modified by following "gradient" of error surface.

MONTANA STATE UNIVERSITY

Mountains Minds

Single-Step Prediction

- Let η be the learning rate.
- Let z be the value of actual payoff.
- Let P_t be the predicted payoff at time t.
- Then
  - $\Delta w_t = \eta(z - P_t)\,\nabla_w P_t$
  - where $\nabla_w P_t$ is the gradient of P_t w.r.t. w.
- Works fine with single-step prediction problems.

MONTANA STATE UNIVERSITY

Mountains Minds

Multi-Step Prediction

- Update only occurs at end of sequence.
- Such delay can "deceive" the decision-making agent.
- Temporal difference learning solves problem by permitting incremental update.
- Successive predictions are used as "ground truth" for previous step in update.

MONTANA STATE UNIVERSITY

Mountains Minds

Temporal Difference Updating

- Consider an *m*-step sequence.
- Let $P_m = z$ (i.e., the actual payoff).
- Observe that

$$z - P_t = \sum_{k=t}^{m-1} (P_{k+1} - P_k)$$

Then

$$\Delta w_t = \eta (P_{t+1} - P_t) \sum_{k=1}^{t} \nabla_w P_k$$

MONTANA STATE UNIVERSITY

Mountains Minds

TD(λ)

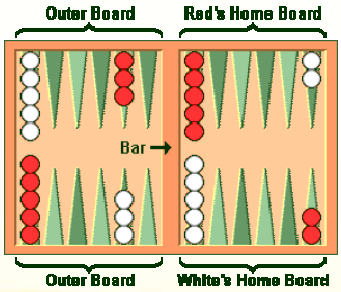
- Sutton defines a family of temporal difference algorithms, parameterized by a discount factor, λ.
- When λ = 0, TD(λ) ignores past updates.
- When λ = 1, TD(λ) gives all past updates equal weight.
- Otherwise, past updates weighted by recency (λ^k)

$$\Delta w_t = \eta (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_w P_k$$

MONTANA STATE UNIVERSITY

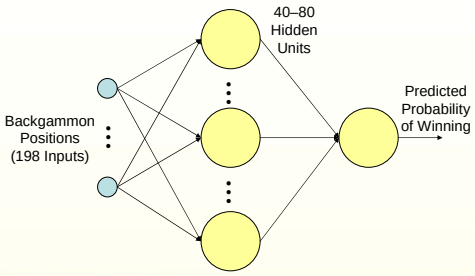
Mountains Minds

Tesauro's TD-Gammon



MONTANA STATE UNIVERSITY

Tesauro's Neural Network



MONTANA STATE UNIVERSITY

The Input Units

- Four units indicate number of white pieces for each "point" (unary): 24 x 4 = 96.
- Four units indicate number of black pieces for each "point" (unary): 24 x 4 = 96.
- One unit to indicate number of white pieces on the bar.
- One unit to indicate number of black pieces on the bar.
- One unit to indicate number of white pieces removed.
- One unit to indicate number of black pieces removed.
- Two units indicate whether it is white's or black's move.
- Total units = 198.

MONTANA STATE UNIVERSITY

Training

- Focus of training was to learn probability of a win for the current player for a particular board position.
- To train, we need a set of games. These came from allowing TD-Gammon to play itself.
- In a game, for a board position, the current player would roll the dice and consider all possible moves (i.e., the possible next states).
- That player would then select the move that yielded transition to a successor state that maximized its probability of winning.

Training

- Initially, the weights of the network were set to small random values.
- The sequence of states (and corresponding moves) were retained for training.
- Weights were updated using TD rule
  - $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \eta[r_{t+1} + \gamma V_t(s') - V_t(s)]\boldsymbol{e}_t$
  - $\boldsymbol{e}_t = \gamma\lambda\boldsymbol{e}_{t-1} + \nabla_{\boldsymbol{\theta}(t)} V_t(s_t)$

Summary of Results

Program	Hidden Units	Training Games	Opponents	Results
TD-Gam 0.0	40	300,000	Other Pgms	Tied for Best
TD-Gam 1.0	80	300,000	Robertie, Magriel	–13 pts/51 games
TD-Gam 2.0	40	800,000	Var. Grandmasters	–7 pts/38 games
TD-Gam 2.1	80	1,500,000	Robertie	–1 pts/40 games
TD-Gam 3.0	80	1,500,000	Kazaros	+6 pts/20 games

Evolutionary Neural Networks

- Apply “genetic algorithms” to learn neural networks.
- Create population of networks.
- Evaluate population against “fitness cases.”
- Select based on fitness.
- Recombine for next generation.
- Repeat until convergence.

MONTANA STATE UNIVERSITY

Mountains Minds

Evolutionary Neural Networks

- Cliff & Miller (Univ. Sussex) learned pursuit-evasion behavior.
- Evolved dynamic recurrent neural network controllers.
- Applied “co-evolution.”
- Used mutation and gene duplication.

MONTANA STATE UNIVERSITY

Mountains Minds