Reasoning with Uncertainty

Markov Decision Models

Markov Decision Process Models

- Markov models represent the behavior of a random process, including its internal state and the externally visible observations
  - So far represents a passive process that is being observed but can not be actively influenced
  - Represents a Markov chain
    - Observation probabilities and emission probabilities are just a different view of the same model component
- General Markov Decision Processes extend this to represent random processes that can be actively influenced through the choice of actions

Markov Decision Process Model

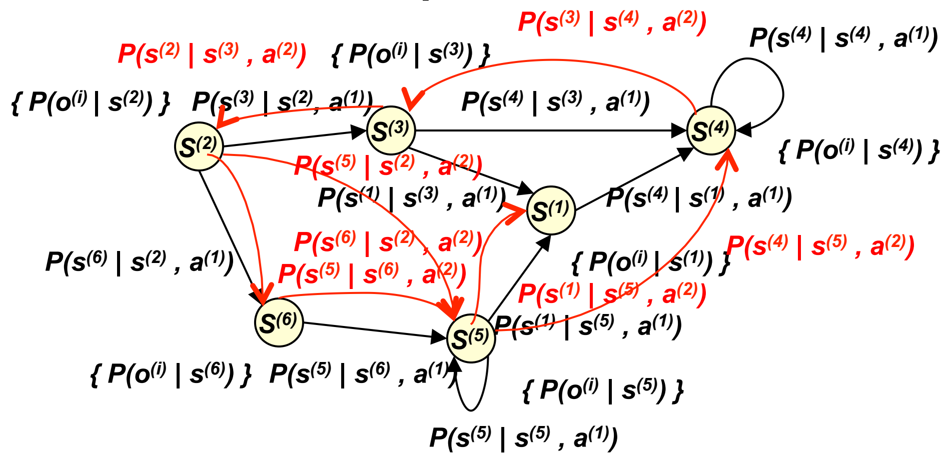
- Extends the Markov chain model
  - Adds actions to represent decision options
  - Modifies transitions to reflect all possibilities
- In its most general form the Markov model for a system with decision options contains:

  $<S, A, O, T, B, \pi>$

  - $S=\{s^{(1)},\ldots,s^{(n)}\}$: State set
  - $A=\{a^{(1)},\ldots,a^{(l)}\}$: Action set
  - $O=\{o^{(1)},\ldots o^{(m)}\}$: Observation set
  - T: $P(s^{(i)} | s^{(j)}, a^{(k)})$: Transition probability distribution
  - B: $P(o^{(i)} | s^{(j)})$: Observation probability distribution
  - π: $P(s^{(i)})$: Prior state distribution

Markov Decision Process Model

- The general Markov Decision Process model represents all possible Markov chains that result by applying a decision policy
  - Policy, *Π* represents a mapping from states/ situations to probabilities of actions



$P(s^{(3)} \mid s^{(4)}, a^{(2)})$
$P(s^{(4)} \mid s^{(4)}, a^{(1)})$
$P(s^{(2)} \mid s^{(3)}, a^{(2)})$
$\{ P(o^{(i)} \mid s^{(3)}) \}$
$\{ P(o^{(i)} \mid s^{(2)}) \}$
$P(s^{(3)} \mid s^{(2)}, a^{(1)})$
$P(s^{(4)} \mid s^{(3)}, a^{(1)})$
$S^{(2)}$
$S^{(3)}$
$S^{(4)}$
$\{ P(o^{(i)} \mid s^{(4)}) \}$
$P(s^{(5)} \mid s^{(2)}, a^{(2)})$
$P(s^{(1)} \mid s^{(3)}, a^{(1)})$
$S^{(1)}$
$P(s^{(4)} \mid s^{(1)}, a^{(1)})$
$P(s^{(6)} \mid s^{(2)}, a^{(2)})$
$P(s^{(6)} \mid s^{(2)}, a^{(1)})$
$P(s^{(5)} \mid s^{(6)}, a^{(2)})$
$\{ P(o^{(i)} \mid s^{(1)}) \}$
$P(s^{(4)} \mid s^{(5)}, a^{(2)})$
$P(s^{(1)} \mid s^{(5)}, a^{(2)})$
$S^{(6)}$
$S^{(5)}$
$P(s^{(1)} \mid s^{(5)}, a^{(1)})$
$\{ P(o^{(i)} \mid s^{(6)}) \}$
$P(s^{(5)} \mid s^{(6)}, a^{(1)})$
$\{ P(o^{(i)} \mid s^{(5)}) \}$
$P(s^{(5)} \mid s^{(5)}, a^{(1)})$

Policy

- A policy represents a decision strategy
  - Under the Markov assumptions, an action choice only needs to depend on the current state

    $\Pi(s)$
    - Deterministic policy

      $\Pi(s^{(i)}) = a^{(j)}$
    - Probabilistic policy

      $\Pi(s^{(i)}, a^{(j)}) = P(a^{(j)} \mid s^{(i)})$
- Under a policy the general Markov decision process model reduces to a Markov chain
  - Transition probabilities could be re-written

$$P_\Pi(s^{(i)} \mid s^{(j)}) = \sum_k P(s^{(i)} \mid s^{(j)}, a^{(k)}) \Pi(s^{(j)}, a^{(k)})$$

Policy

- In the general formulation of the problem the state is generally not known
  - Policy as defined so far can only be executed inside the underlying model
  - In the general case this requires external policies to be defined in terms of the complete observation sequence
    - Or alternatively in terms of the Belief state
      - Belief state is the state of the belief of the system (i.e. the probability distribution over states given the observations)
      - Note: it is *not* the state that you believe the system is in (i.e. the most likely state)

Markov Decision Process Model

- When applying a known policy the general system resembles a Hidden Markov Model
  - The tasks of determining the quality of the model, of determining the "best" state sequence, or to learn the model parameters can be solved using the same HMM algorithms if the policy is known
- Markov Decision Process tasks are related to determining the "best" policy
  - Requires a definition of "best"
    - Uses utility theory and rational decision making

Markov Decision Processes

- Partially Observable Markov Decision Processes (POMDPs) use the model definition with a task definition
  - Rather than defining it directly with utilities it defines the task using Reward
    - Reward can be seen as the instantaneous "value" gain
      - Reward can be defined as a function of the state and action independent of the policy
      - Utility of a state is a function of the policy
    - Model/environment "generates" rewards at each step

$<S, A, O, T, B, \pi, R>$

- $R: S \times A \rightarrow IR: R(s,a)$: Reward function

From Reward to Utility

- To obtain a utility needed for decision making a relation between rewards and utilities has to exist
  - Utility of a policy in a state is driven by all the rewards that will be obtained when starting to execute the policy in this state
    - Sum of future rewards

      $$V(s_t) = E\left[\sum_{\tau=t}^{end\ \ of\ \ time} R(s_\tau, a_\tau)\right]$$

      To be a valid rational utility, it has to be finite
      - Finite horizon utility $V(s_t) = E\left[\sum_{\tau=t}^{t+T} R(s_\tau, a_\tau)\right]$
      - Average reward utility $V(s_t) = E\left[\sum_{\tau=t}^{t+\Delta} 1/\Delta\, R(s_\tau, a_\tau)\right]$
      - Discounted sum of future rewards $V(s_t) = E\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} R(s_\tau, a_\tau)\right]$

Reward and Utility

- All three formulations of utility are used
- The most commonly used formulation is the discounted sum of rewards formulation
  - Simplest to treat mathematically in most situations
    - Exception is tasks that naturally have a finite horizon
  - Discount factor choice influences task definition
    - Discount factor represents how much more "important" immediate reward is relative to future reward
    - Alternatively it can be interpreted as the probability with which the task continues (rather than stop)

Markov Decision Processes

- Markov Decision Process (MDP) usually refers to the fully observable case of the Markov Decision Process model
  - Fully observable implies that observations always allow to identify the system state
- An MDP can thus be formulated as

  $<S, A, T, R>$

  - $S=\{s^{(1)},\ldots,s^{(n)}\}$: State set
  - $A=\{a^{(1)},\ldots,a^{(l)}\}$: Action set
  - T: $P(s^{(i)} \mid s^{(j)}, a^{(k)})$: Transition probability distribution
  - R: $R(s^{(i)}, a^{(j)})$: Reward function

Markov Decision Processes

- Reward is sometimes defined in alternative ways:
  - State reward: *R(s)*
  - State/action/next state reward: *R(s, a, s')*
- All formulations are valid but might require different state representations to make the expected value of the reward stationary
  - Expected value of the reward can only depend on the arguments

Markov Decision Processes

- The main task addressed in Markov Decision Processes is to determine the policy that maximizes the utility
- Value function represents the utility of being in a particular state

$$V^{\Pi}(s) = E_{s_t=s}\left[\sum_{\tau=t}^{\infty} \gamma^{\tau-t} R(s_\tau)\right]$$

$$= R(s) + E\left[\sum_{\tau=t+1}^{\infty} \gamma^{\tau-t} R(s_\tau)\right] = R(s) + \gamma E\left[\sum_{\tau=t+1}^{\infty} \gamma^{\tau-(t+1)} R(s_\tau)\right]$$

$$= R(s) + \gamma \sum_{s'} \sum_{a} \Pi(s,a) P(s' \mid s,a) E_{s_{t+1}=s'}\left[\sum_{\tau=t'}^{\infty} \gamma^{\tau-t'} R(s_\tau)\right]$$

$$= R(s) + \gamma \sum_{s'} \sum_{a} \Pi(s,a) P(s' \mid s,a) V^{\Pi}(s')$$

Markov Decision Processes

- Value function for a given policy can be written as a recursion
  - Alternatively we can interpret the formula as a system of linear equations over the state values

$$V^{\Pi}(s) = R(s) + \gamma \sum_{s'} \sum_{a} \Pi(s,a) P(s' \mid s,a) V^{\Pi}(s')$$

  - Two ways to compute the value function for a given policy
    - Solve the system of linear equations (Polynomial time)
    - Iterate over the recursive formulation
      - Starting with a random function $V_0^{\Pi}(s)$
      - Update the function for each state

$$V_{t+1}^{\Pi}(s) = R(s) + \gamma \sum_{s'} \sum_{a} \Pi(s,a) P(s' \mid s,a) V_t^{\Pi}(s')$$

      - Repeat step 2 until the function no longer changes significantly

Markov Decision Processes

- To be able to pick the best policy using the value (utility) function, there has to be a value function that is at least as good in every state as any other value function
  - Two value functions have to be comparable
  - Consider the modified value function
    $$V'^{\Pi}(s) = R(s) + \gamma \max_{\Pi'} \sum_{s'} \sum_{a} \Pi'(s,a) P(s' \mid s,a) V'^{\Pi}(s')$$
    - This effectively picks according to policy *Π′* for one step in state s but otherwise behaves like policy *Π*
      - In state s this function is at least as large as the original value function for policy *Π*
      - Consequently it is at least as large as the value function for policy *Π* in every state

Markov Decision Processes

- There is at least one "best" policy
  - Has a value function that in every state is at least as large as the one of any other policy
  - "Best" policy can be picked by picking the policy that maximizes the utility in each state
- Considering picking a deterministic policy

$$V'^{\Pi}(s) = R(s) + \gamma \max_{\Pi'} \sum_{s'} \sum_{a} \Pi'(s,a) P(s' \mid s,a) V'^{\Pi}(s')$$

$$= R(s) + \gamma \max_{\Pi'} \sum_{a} \Pi'(s,a) \sum_{s'} P(s' \mid s,a) V'^{\Pi}(s')$$

$$= R(s) + \gamma \max_{a} \sum_{s'} P(s' \mid s,a) V'^{\Pi}(s')$$

  - At least one of the "best" policies is always deterministic

Value Iteration

- A "best" policy can be determined using Value iteration
  - Use dynamic programming using the recursion for best policy to determine the value function
    - Start with a random value function *$V_0(s)$*
    - Update the function based on the previous estimate

      $$V_{t+1}(s) = R(s) + \gamma \max_a \sum_{s'} P(s' \mid s, a) V_t(s')$$
    - Iterate until the value function no longer changes
    - The resulting value function is the value function of the optimal policy, *V^**
  - Determine the optimal policy

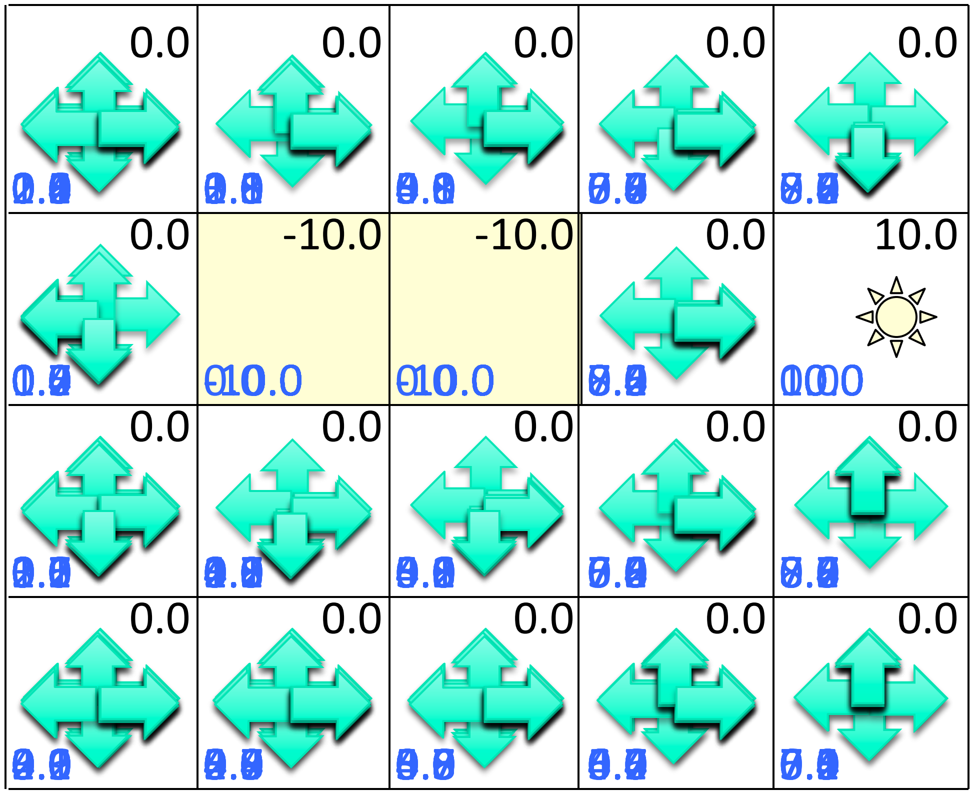
$$\Pi^*(s) = \arg\max_a R(s) + \gamma \sum_{s'} P(s' \mid s, a) V^*(s')$$

Value Iteration

- Value iteration provides a means of computing the optimal value function and, given the model is known, the optimal policy
  - Will converge to the optimal value function
    - Number of iterations needed for convergence is related to the longest possible state sequences that leads to non zero reward
      - Usually requires to stop iteration before complete convergence using a threshold on the change of the function
- Solving as a system of equations is no longer efficient
  - Nonlinear, non-differentiable equations due to the presence of *max* operation

Value Iteration Example

- Grid world task with four actions: up, down, left, right
  - Goal and obstacle are absorbing
  - Actions succeed with probability 0.8 and otherwise move sideways
  - Discout factor is 0.9



Value Iteration

- The *Q* function provides an alternative utility function defined over state/action pairs
  - Represents utility defined over a state space where the state representation includes the action to be taken
    - Alternatively, it represents the value if the first action is chosen according to the parameter and the remainder according to the policy

$$Q^{\Pi}(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) V^{\Pi}(s')$$

$$V^{\Pi}(s) = \sum_{a} \Pi(s,a) Q^{\Pi}(s,a)$$

$$Q^{\Pi}(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \sum_{b} \Pi(s',b) Q^{\Pi}(s',b)$$

    - The Q function can also be defined recursively

Value Iteration

- As with state utility, state/action utility can be used to determine an optimal policy
  - Pick initial *Q* function Q_0
  - Update function using the recursive definition

    $$Q_{t+1}(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_b Q_t(s',b)$$
  - Repeat until it converges
    - Converges to optimal state/action utility function *Q**
  - Determine optimal policy as

    $$\Pi^*(s) = \arg\max_a Q^*(s,a)$$
- State/action utility requires computation of more values but does not need transition probabilities to pick optimal policy from *Q**

Value Iteration

- Convergence of value iteration in systems where state sequences leading to some reward can be arbitrary long can only be achieved approximately
  - Need threshold on change of value function
    - Some chance that we terminate before the value function produces the optimal policy
      - But: policy will be approximately optimal (i.e. the value of the policy will be very close to optimal
- To guarantee optimal policy we need an algorithm that is guaranteed to converge in finite time

Policy Iteration

- Value iteration first determines the value function and then extracts the policy
- Policy iteration directly improves the policy until it has found the best one
  - Optimize the utility of the policy by adjusting the policy parameters (action choices)
    - Can be represented as optimization of a marginal probability of policy parameters and the hidden utilities
  - Policy iteration uses a variation of Expectation Maximization to optimize the policy parameters such as to achieve an optimal expected utility

Policy Iteration

- Policy iteration directly improves the policy
  - Start with a randomly picked (deterministic) policy Π_0
  - E-Step:
    - Compute the utility of the policy for each state $V^{\Pi}(s)$ assuming the current policy
      - Usually this is done by solving the linear system of equations

$$V^{\Pi_t}(s) = R(s) + \gamma \sum_{s'} \sum_{a} \Pi(s,a) P(s' \mid s,a) V^{\Pi}(s')$$

  - M-Step:
    - Determine the optimal policy parameter for each state assuming the expected utilityfunction from the E-step

$$\Pi_{t+1}(s) = \operatorname{argmax}_a R(s) + \gamma \sum_{s'} P(s' \mid s,a) V^{\Pi_t}(s')$$

  - Repeat until policy no longer changes

Policy Iteration

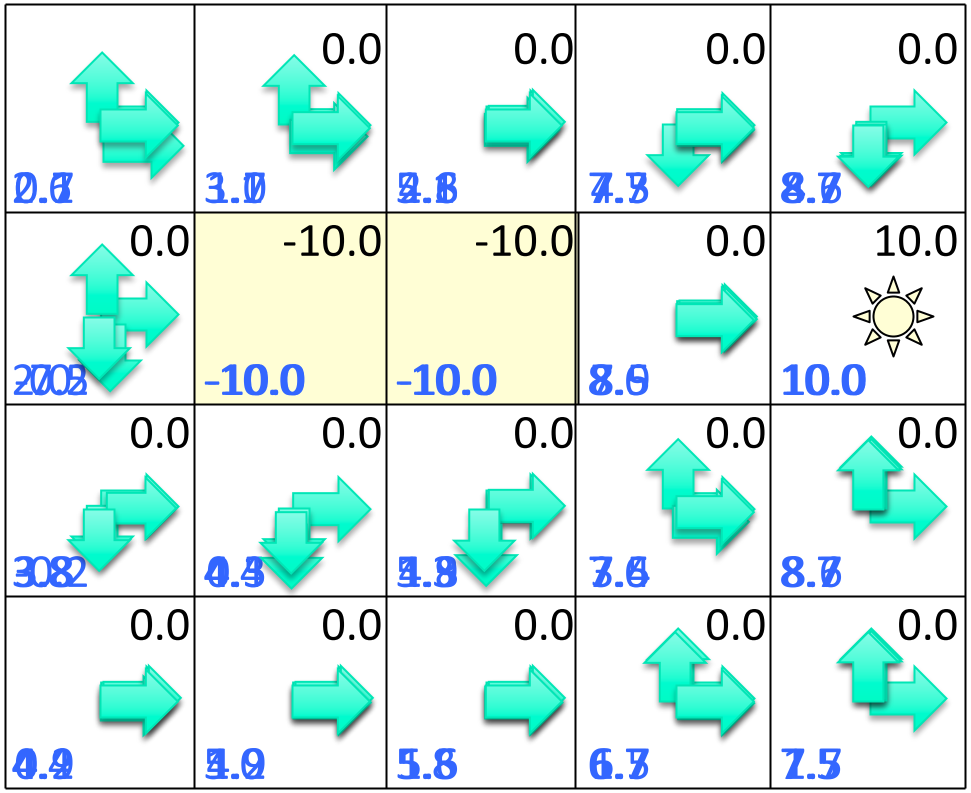
- In each M-step, the algorithm either strictly improves the policy or terminates
  - The state utility function does not have local maxima in terms of the policy parameters
    - Follows since if a change in action in a single state improves the utility for that state it can not reduce the utility for any other state
    - Implies that if the algorithm converges it has to converge to a globally optimal policy
  - Since no policy can be repeated and there are only a finite number of deterministic policies, the algorithm will converge in finite time
    - Thus policy iteration is guaranteed to converge to the globally optimal policy in finite time

Policy Iteration

- Policy iteration has detectable, guaranteed convergence
  - Policy no longer changing in the M-step
- Each iteration of policy iteration is more complex than an iteration of value iteration
  - One iteration of Value iteration: $O(l*n^2)$
  - One iteration of Policy iteration: $O(n^3+l*n^2)$
    - Assuming use of $O(n^3)$ algorithm for solving system of linear equations; best known is $O(n^{2.4})$ but impractical
    - In each M-step, the algorithm either strictly improves the policy or terminates
- Value iteration is easier to implement

Policy Iteration Example

- Grid world task with four actions: up, down, left, right
  - Goal and obstacle are absorbing
  - Actions succeed with probability 0.8 and otherwise move sideways
  - Discout factor is 0.9



Monte Carlo Solutions

- Both Value and Policy iteration require knowledge of the model parameters (i.e. the transition probabilities)
- Value iteration can be performed using Monte Carlo sampling of states without explicit use of the transition probabilities
  - Monte Carlo dynamic programming requires to replace the value update with a sampled version
    - Assuming transition sample set *D*

$$Q_{t+1}(s,a) = R(s) + \gamma \sum_{s'} P(s' \mid s,a) \max_b Q_t(s',b)$$

$$\approx R(s) + \gamma \sum_{(s,a,s')\in D} \frac{1}{\#(s,a,?) \in D} \max_b Q_t(s',b)$$

Monte Carlo Solutions

- Instead of first collecting all the samples and then using them for the value function calculation we can also update the function incrementally for each sample
  - Implies that number of samples for a state action pair is not known a-priori
  - Implies that each update is to be done based on a different value function
- Generally Monte Carlo solutions use one of two averaging approaches
  - Incremental averaging
  - Exponentially weighted averaging

Monte Carlo Solutions

- Incremental averaging update

$$Q_{t+1}(s,a) = \frac{k(s,a)}{k(s,a)+1} Q_t(s,a) + \frac{1}{k(s,a)+1}\left(R(s) + \gamma \max_b Q_t(s',b)\right)$$

  - k is the number of samples so far
- Exponentially weigthed averaging update

$$Q_{t+1}(s,a) = \left(1-\alpha_t\right) Q_t(s,a) + \alpha_t \left(R(s) + \gamma \max_b Q_t(s',b)\right)$$

- Each update is based on a single sample
  - Both formulations will converge to the optimal *Q* function under certain circumstances
  - Exponentially weighted averaging is more commonly used.
    - Is more robust towards very bad initial guesses at the value function

Monte Carlo Solutions

- Exponentially weighted averaging converges if certain conditions on α have to be fulfilled
  - Too large values will cause instability
    - Over-commitment to the new sample
  - Too small values will not allow enough change to reach the optimal function
    - Under-commitment to samples and thus non-vanishing influence of initial guess
  - There is no fixed definition for too large and too small, but conditions:

$$\sum_{t=1}^{\infty} \alpha_t \rightarrow \infty \quad \leftarrow \quad \text{“Large enough”}$$

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty \quad \leftarrow \quad \text{“Not too large”}$$

Monte Carlo Solutions

- Monte Carlo simulation techniques allow to generate optimal policies and value functions from data without knowledge of the system model
  - Policies take into account the uncertainty in the transitions

Partially Observable Markov Decision Process (POMDP)

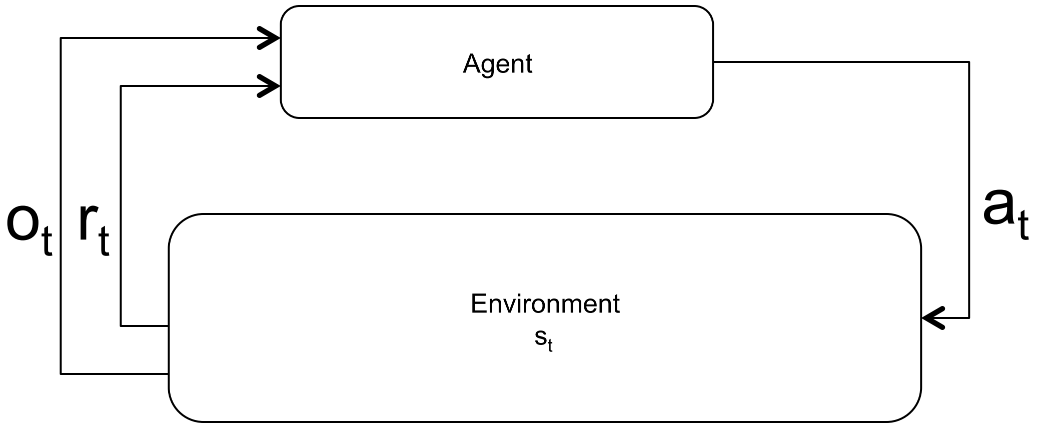
- POMDPs include partial observability
  - Again represents task with a reward function

    $<S, A, O, T, B, \pi, R>$
    - $S=\{s^{(1)},\ldots,s^{(n)}\}$: State set
    - $A=\{a^{(1)},\ldots,a^{(l)}\}$: Action set
    - $O=\{o^{(1)},\ldots o^{(m)}\}$: Observation set
    - T: $P(s^{(i)} \mid s^{(j)}, a^{(k)})$: Transition probability distribution
    - B: $P(o^{(i)} \mid s^{(j)})$: Observation probability distribution
    - π: $P(s^{(i)})$: Prior state distribution
    - R: $R(s^{(i)}, a^{(j)})$: Reward function
  - Markov Property:

$$P(r_t, s_t \mid s_{t-1}, a_{t-1}, s_{t-1}, \ldots, s_1) = P(r_t, s_t \mid s_{t-1}, a_{t-1})$$

$$P(o_t \mid s_t, a_t, o_{t-1,} s_{t-1}, \ldots, s_1) = P(o_t \mid s_t)$$

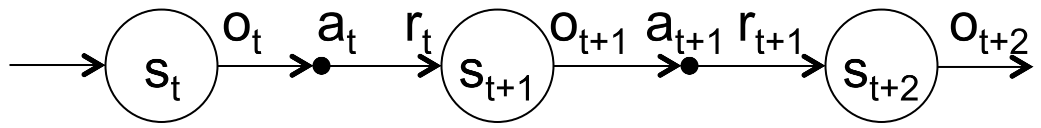
Sequential Decision Making in Partially Observable Systems



- State only exists inside the environment
  - Inaccessible to the agent
- Observations are obtained by the agent
  - Agent can try to infer state from observations

Sequential Decision Making in Partially Observable Systems

- Executions can be represented as sequences
  - From the environments view:



    - state / observation / action / reward sequences
  - From the agents view:

$$\pi_0,\ o_0, a_0, r_0, o_1, \ldots, o_t, a_t, r_t, \ldots$$

    - observation / action / reward sequences
- Agent has to make decisions based on knowledge extracted from the observations

Partially Observable Markov Decision Processes

- Underlying system behaves as in MDP except
  - In every state it emits a probabilistic observation
- For analysis simplifications made in the case of MDPs will be made
  - Transition probabilities are independent of the reward probabilities $T: P(s_i \mid s_j, a)$
  - Reward probabilities only depend on the state and are static $R(s) = \sum_r P(r \mid s) r$
  - Observations contain all obtainable information about state (i.e. reward does not add state info)

Designing POMDPs

- Design the MDP of the underlying system
  - Ignore whether state attributes are observable
- Determine the set of observations and design the observation probabilities
  - Ensure that observations only depend on state
    - If that is not the case the state representation of the underlying MDP has to be augmented
- Design a reward function for the task
  - Ensure that reward only depends on the state

Belief State

- In a POMDP the state is unknown
  - Decisions have to be made based on the knowledge about the state that the agent can gather from observations
- Belief state is the state of the agent's belief about the state it is in
  - Belief state is a probability distribution over the state space

$$b_t : b_t(s) = P(s_t = s \mid \pi_0, o_0, a_0, \ldots, a_{t-1}, o_t)$$

Belief State

- Belief state contains all information about the past of the system

$$P(s_t = s \mid b_{t-1}, o_0, a_0, \ldots, a_{t-1}, o_t) = P(s_t = s \mid b_{t-1}, a_{t-1}, o_t)$$

$$P(r_t \mid b_t, o_0, a_0, r_0, \ldots, a_{t-1}, o_t) = P(r_t \mid b_t)$$

  - POMDP is Markov in terms of the belief state

- Belief state can be tracked and updated to maintain information

$$b_t(s') = P(s_t = s' \mid b_{t-1}, a_{t-1}, o_t) = \frac{P(s_t = s', o_t \mid b_{t-1}, a_{t-1})}{P(o_t \mid b_{t-1}, a_{t-1})}$$

$$= \alpha P(s_t = s', o_t \mid b_{t-1}, a_{t-1}) = \alpha P(s_t = s' \mid b_{t-1}, a_{t-1}) P(o_t \mid s_t = s', b_{t-1}, a_{t-1})$$

$$= \alpha P(o_t \mid s') \sum_s P(s' \mid s, a_{t-1}) b_{t-1}(s)$$

Decision Making in POMDPs

- Value-function based methods have to compute the value of a belief state
  $$V^{\pi}(b) \quad , \quad Q(b,a)$$
- System is Markov in terms of the belief state
  - Belief-state MDP
    $$\langle A,\{b\},T_b,R\rangle \quad , \quad P(b'\mid b,a)=\begin{cases} P(o\mid b,a) & b'\leftarrow b,\ a,\ o \\ 0 & otherwise \end{cases}$$
    $$R(b)=\sum_s b(s)R(s)$$
  - Can apply any MDP learning method on this space
    - Belief state space is continuous (thus infinite)
    - Need a function approximator

Value Function Approaches for POMDPs

- Q-function on finite horizon POMDPs is locally linear in terms of the belief state

  $$Q(b,a) = \max_{q \in L_a} \sum_s q(s) b(s)$$

  - Compute set of value vectors, q
    - Number of vectors grows exponentially with the duration of policies
  - Different algorithms have been used to compute the locally linear function
    - Exact value iteration
    - Approximate systems with finite vector sets

Value Function Approaches for POMDPs

- The simplest approximate model is linear

$$Q(b,a) = \sum_s q(s,a)b(s)$$

  - Approaches differ in the way they estimate the values *q(s,a)*
    - Q_{MDP} computes *q(s,a)* assuming full observability
      - Use value iteration to compute *q(s,a)=Q*(s,a)*
    - Replicated Q-learning uses the assumption that parameter values independently predict

$$q_{t+1}(s,a) = q_t(s,a) + \alpha\left(r + \gamma \max_c Q_t(b',c) - q(s,a)\right)$$

    - Linear Q-learning treats states separately

$$q_{t+1}(s,a) = q_t(s,a) + \alpha\left(r + \gamma \max_c Q_t(b',c) - Q_t(b,a)\right)$$

Value Function Approaches for POMDPs

- The linear approximation limits the degree to which the optimal value function and thus policy can be computed
  - In addition, Q_{MDP} strictly over-estimates the value
    - Assumes state information will be known in the next step
    - Will not take actions that only remove state uncertainty
- Better approximations can be maintained by building more complex approximate representations

Value Function Approaches for POMDPs

- POMDP can be approximated using completely sampling-based (Monte Carlo POMDP)
  - Compute (track) Belief state using a particle filter
  - Represent Value function (Q-function) as linear function over support points in belief state space

$$Q(b,a) = \sum_{b' \in SP} \frac{w_{b,b'}}{\sum_{b'' \in SP} w_{b,b''}} Q(b',a)$$

    - Representation as a set *SP* of Q-values over belief states
    - Weights represent similarity of the two Belief states
      - E.g. KL-Divergence of the two distributions $KL(b \,\|\, b') = \sum_s b(s) ld \frac{b(s)}{b'(s)}$
    - Often simplified using only *k* most similar elements of *SP*

Value Function Approaches for POMDPs

- Monte Carlo POMDP
  - Update sampled Belief state Q-values based on current samples
    - Belief state value for state sample *b* can be updated using sampled value iteration
      - For each action sample observations according to *P(o|b,a)* and compute the corresponding future belief state *b'*
      - Compute update

$$\Delta Q(b,a) = \left(R(b) + \gamma \sum_{b'} \max_{a'} \frac{1}{\sum_{b''\in SP} w_{b',b''}} \sum_{b''\in SP} w_{b',b''} Q_t(b',a') \right) - Q_t(b,a)$$

      - Distribute update over support points according to weight
        - If few elements in SP are similar, add b to SP

Policy Approaches for POMDPs

- Policy improvement approaches can be applied using the same value function approximations
  - Working with exact locally linear value function is difficult since in each iteration new coefficients have to be computed
  - Approximate representations are more efficient for policy improvement
    - Usually maximization of the policy (and thus EM) is not possible.

Policy Approaches for POMDPs

- The representation of the Belief state in terms of a probability distribution over states is difficult to handle for policy approaches
  - Alternative representation of Belief state in terms of an observation/action sequence
    - Each complete observation/action sequence represents a unique Belief state

      $$h = o_0, a_0, \ldots, o_k \quad \rightarrow \quad b_h$$
  - Sampled representation of value function in terms of set of observation/action/reward histories

    $$h_r = o_0, a_0, r_0, \ldots, o_k$$

Policy Approaches for POMDPs

- Value function of a policy is weighted sum over value of histories

$$V^{\Pi}(o_0,a_0,\ldots,o_k)=\frac{1}{\left|\{h: prefix_k(h)=o_0,a_0,\ldots,o_k\}\right|}\sum_{h:prefix_k(h)=o_0,a_0,\ldots,o_k}\sum_{l=k}^{\infty}\gamma^{l-k}r_l^{(h)}$$

  - Policy improvement by finding what changes to the policy would improve the value function
    - Value of a modified policy can be estimated from the same samples using importance sampling

$$V^{\Pi'}(o_0,a_0,\ldots,o_k)=\frac{1}{\sum_{h:prefix_k(h)=o_0,a_0,\ldots,o_k}\frac{P(h\mid\Pi')}{P(h\mid\Pi^{(h)})}}\sum_{h:prefix_k(h)=o_0,a_0,\ldots,o_k}\frac{P(h\mid\Pi')}{P(h\mid\Pi^{(h)})}\sum_{l=k}^{\infty}\gamma^{l-k}r_l^{(h)}$$

    - Can compute gradient of the value estimate with respect to probabilistic policy parameters

Policy Approaches for POMDPs

- Probability of a history is a function of the transition probabilities, observation probabilities, and policy

$$P(h \mid \Pi) = P(o_0,\ldots,o_k \mid \pi,T,B,a_0,\ldots,a_{k-1})P(a_0,\ldots,a_{k-1} \mid \Pi)$$

$$= P(o_0,\ldots,o_k \mid \pi,T,B,a_0,\ldots,a_{k-1})\prod_{t=0}^{k-1}\Pi(h^{0..t},a_t)$$

  - Value function only depends on policy parameters

$$V^{\Pi}(o_0,a_0,\ldots,o_k) = \frac{1}{\sum_{h:prefix_k(h)=o_0,a_0,\ldots,o_k}\frac{P(a_0,\ldots,a_l \mid \Pi)}{P(a_0,\ldots,a_l \mid \Pi^{(h)})}}\sum_{h:prefix_k(h)=o_0,a_0,\ldots,o_k}\frac{P(a_0,\ldots,a_l \mid \Pi)}{P(a_0,\ldots,a_l \mid \Pi^{(h)})}\sum_{l=k}^{\infty}\gamma^{l-k}r_l^{(h)}$$

    - Gradient of value function only depends on value at the current policy and the derivative of the policy

Policy Approaches for POMDPs

- To perform policy improvement the policy has to be parameterized
  - Often as a probabilistic Softmax policy

$$\Pi(b,a,v) = \frac{e^{\sum_i v_i x_i(b,a)}}{\sum_c e^{\sum_i v_i x_i(b,c)}}$$

  - Allows for gradient calculation based on histories
- Results in effective algorithms to locally improve policies
  - Has local minima based on policy parameterization

Markov Decision Processes

- Partially Observable Markov Decision Processes are a very general means to model uncertainty in sequential processes involving decisions
  - Extend Hidden Markov Models with actions and tasks
    - Tasks are represented with reward functions
  - Utility characterizes action selection under uncertainty
    - Outcomes as well as observations can be uncertain
- Provides a powerful framework to model process uncertainty and uncertainty in decisions
  - Efficient algorithms for the fully observable case
    - Approximation approaches for partially observable case