Reinforcement Learning & Lab3

Siyuan Wang
2018-05-09

Outline

- Reinforcement Learning for Gomoku
- POMDP for Spoken Dialog Systems
- Lab3

Reinforcement Learning for Gomoku

Reinforcement Learning for Gomoku

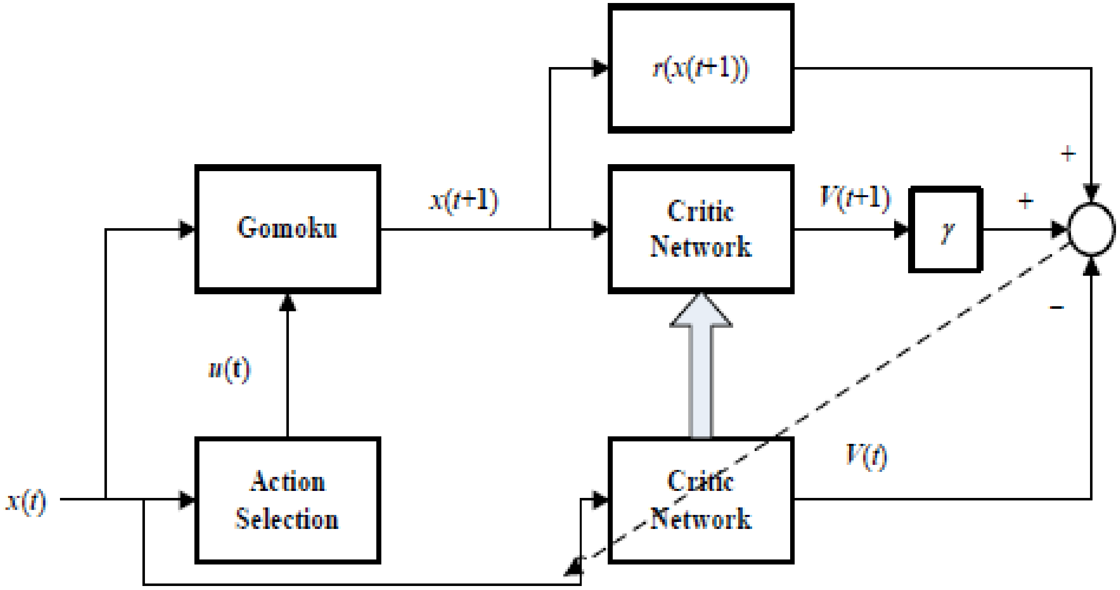
- Adaptive Dynamic Programming for Gomoku
- ADP with MCTS for Gomoku

ADP for Gomoku

- Key idea of ADP

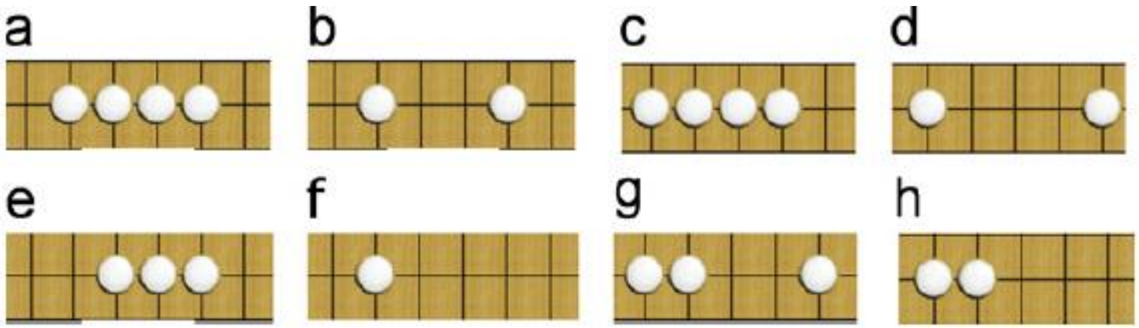
  In TD learning, the action decision or value function can be described in continuous form, approximated by nonlinear function such as neural network

- The ADP structure



ADP for Gomoku

- Input state to describe a board situation
  - 20 patterns for each of two players, totally 40 patterns



  - Whose turn to move
  - In the offensive/defensive(Who is first to move)

ADP for Gomoku

- Input state to describe a board situation
  - Five input nodes indicate the number of every pattern except for five-in-a-row (n denotes the number of a pattern)

Value of n	Input 1	Input 2	Input 3	Input 4	Input 5
0	0	0	0	0	0
1	1	0	0	0	0
2	1	1	0	0	0
3	1	1	1	0	0
4	1	1	1	1	0
>4	1	1	1	1	$(n-4)/2$

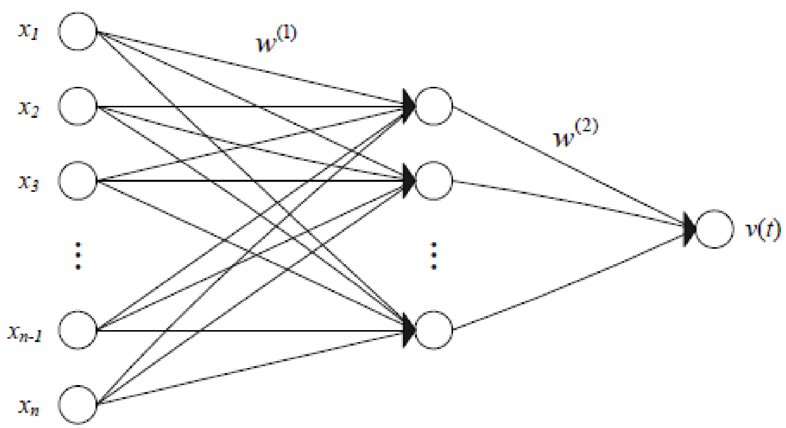
  - The number of the special pattern five-in-a-row, is represented by 1 input node. If this pattern shows up, then its input is 1, otherwise 0

ADP for Gomoku

- Input state to describe a board situation
  - For each pattern we assign two input nodes to represent the turn
  - Use two input nodes to indicate which player is the first to move
  - Totally 19*5*2+1*1*2+40*2+2 = 274 input nodes

ADP for Gomoku

- Critic Network in the ADP
  - Used to evaluate board situations(winning probability of player1)
  - A feed forward three-layer fully connected neural network



  - Unnecessary to be neural network, you can try other functions

ADP for Gomoku

- Critic Network in the ADP
  - Train the neural network
    - Define the prediction error

$$e(t) = \alpha[r(t+1) + \gamma V(t+1) - V(t)]$$

    - To minimize the objective error

$$E(t) = \frac{1}{2}e^2(t)$$

ADP for Gomoku

- Action
  - Player 1 chooses the move that leads to the state with the maximal output value obtained from the neural network.
  - Player 2 selects the move that leads to the state with the minimal output value obtained by the same neural network.

ADP for Gomoku

- Action
  - Reduce the action space
    - Only considering the empty positions near the ones occupied
    - When there are several alternative actions which have equally high evaluation, we simply choose the one that is last found

ADP for Gomoku

- Action
  - Cope with the exploration and exploitation dilemma
    - Let player 2 randomly select his first move, meanwhile player 1 place his piece on the center of the board if he is in the offensive and select his first move randomly if he is in the defensive
    - Let both players select moves following ϵ-greedy policy

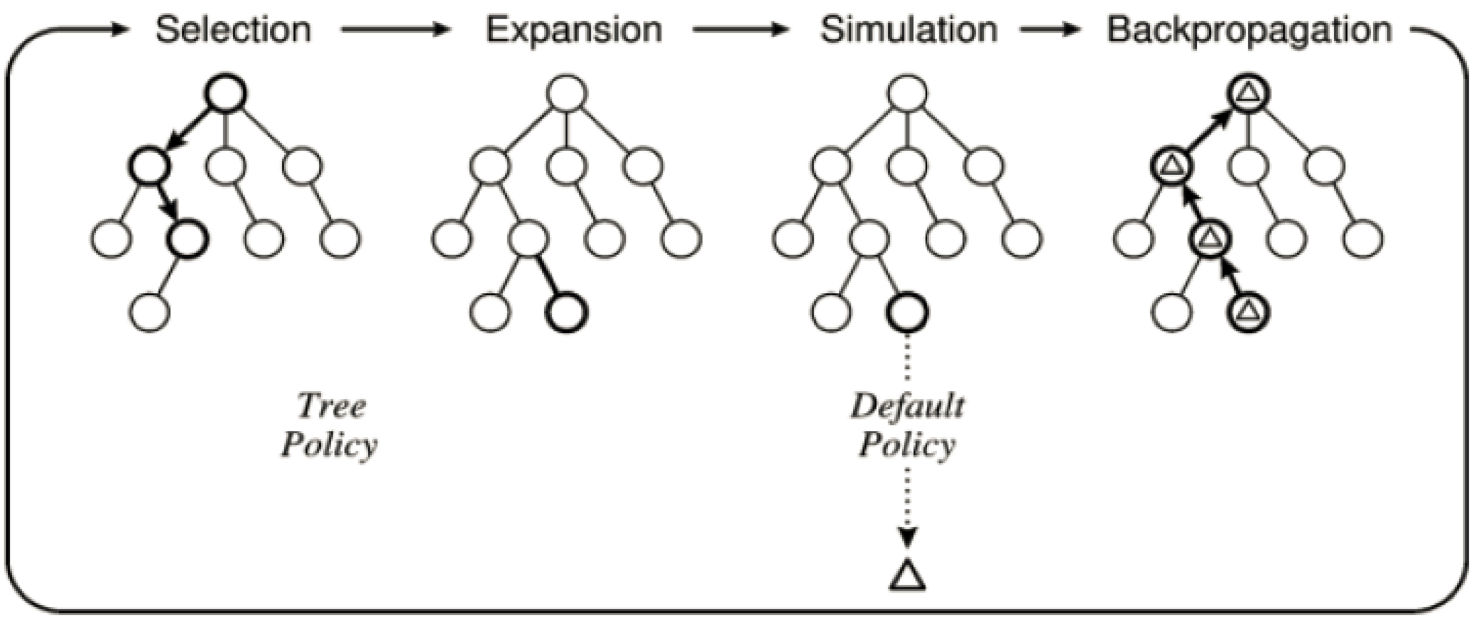
$$a(t) = \begin{cases} \arg\max_a V(t+1) & \text{with probability } 1-\varepsilon \\ \text{random action} & \text{with probability } \varepsilon \end{cases}$$

ADP for Gomoku

- Reward
  - The reward is set to 0 during the game.
  - After a game, if player 1 wins, the final reward is 1, if he loses, the reward is 0

ADP with MCTS for Gomoku

- Monte Carlo Tree Search(MCTS)



ADP with MCTS for Gomoku

- Heuristic Monte Carlo Tree Search(HMCTS)
- Save more time in simulation than random sampling and get converge earlier

Algorithm 1: HMCTS for Gomoku

```
input original state s_0;
output action a corresponding to the highest value of MCTS;
add Heuristic Knowledge;
obtain possible action moves M from state s_0;
for each move m in moves M do
   reward r_total ← 0;
   while simulation times < assigned times do
      reward r ← Simulation(s(m));
      r_total ← r_total + r;
      simulation times add one;
    end while
    add (m, r_total) into data;
    end for each
return action Best(data)

Simulation(state s_t)
   if (s_t is win and s_t is terminal)  then return 1.0;
                                        else return 0.0;
    end if
    if (s_t satisfied with Heuristic Knowledge)
      then obtain forced action a_f;
             new state s_t+1 ← f(s_t, a_f);
      else choose random action a_r ∈ untried actions;
             new state s_t+1 ← f(s_t, a_r);
    end if
    return Simulation(s_t+1)

Best(data)
    return action a   //the maximum r_total of m from data
```

ADP with MCTS for Gomoku

- Heuristic rules
  - If four-in-a-row is occurred in my side, the player will be forced to move its piece to the position where it can emerge five-in-a-row in my side.
  - If four-in-a-row is occurred in opposite side, the player will be forced to move its piece to the position where it can block five-in-a-row in opposite side.
  - If three-in-a-row is occurred in my side, the player will be forced to move its piece to the position where it can emerge four-in-a-row in my side.
  - If three-in-a-row is occurred in opposite side, the player will be forced to move its piece to the position where it can block four-in-a-row in opposite side.

ADP with MCTS for Gomoku

- Upper Confidence bounds for Tree(UCT)
  - Based on Upper Confidence Bounds(UCB)

$$\frac{Q(v')}{N(v')} + c\sqrt{\frac{2\ln N(v)}{N(v')}}$$

  - $\frac{Q(v')}{N(v')}$ is the average reward of node v', $N(v')$ and $N(v)$ is the visited count of node v' and v
  - Balance the conflict between exploration and exploitation and find out the final result earlier

ADP with MCTS for Gomoku

- Upper Confidence bounds for Tree (UCT)

Algorithm 2: UCT for Gomoku

input create root node v_0 with state s_0;
output action a corresponding to the highest value of UCT;
while within computational budget **do**
  $v_l \leftarrow$ Tree Policy(v_0);
  Policy $\leftarrow$ Heuristic Knowledge;
  reward $r \leftarrow$ Policy($s(v_l)$);
  Back Update(v_l, r);
end while
return action a(Best Child(v_0))

Tree Policy(node v)
  while v is not in terminal state **do**
    if v not fully expanded **then** **return** Expand(v);
      else $v \leftarrow$ Best Child(v, $1/\sqrt{2}$);
    end if
  end while
  return v //this is the best child node

Expand(node v)
  choose random action $a \in$ untried actions from $A(s(v))$;
  add a new child v' to v
   with $s(v') \leftarrow f(s(v), a)$ and $a(v') \leftarrow a$;
  return v' //this is the expand node

Best Child(node v, parameter c)
  return $\arg\max_{v' \in child}((Q(v')/N(v')) + c\sqrt{2\ln N(v)/N(v')})$

Policy(state s)
  while s is not terminal **do**
    if s satisfied with heuristic knowledge **then**
      obtain forced action a;
    else choose random action $a \in A(s)$ uniformly;
    end if
    $s \leftarrow f(s, a)$;
  end while
  return reward for state s

Back Update(node v, reward r)
  while v is not null **do**
    $N(v) \leftarrow N(v) + 1$;
    $Q(v) \leftarrow Q(v) + r$;
    $v \leftarrow$ parent of v;
  end while

ADP with MCTS for Gomoku

- ADP with MCTS
  - Use ADP to train critic network, get 5 candidate moves and their ADP winning probabilities
  - Take candidate moves as the root node of MCTS and simulate, get their MCTS winning probabilities
  - Calculate the weighted sum of two winning probabilities:

$$w_p = \lambda w_1 + (1 - \lambda) w_2$$

ADP with MCTS for Gomoku

- ADP with MCTS

Algorithm 3: ADP with MCTS

input original state s_0;
output action *a correspond to ADP with MCTS*;
M_{ADP}, $W_{ADP} \leftarrow$ ADP Stage(s_0);
$W_{MCTS} \leftarrow$ MCTS Stage(M_{ADP});
for each w_1, w_2 in pairs(W_{ADP}, W_{MCTS}) **do**
  $w_p \leftarrow \lambda w_1 + (1\text{-}\lambda)w_2$;
  add p into P;
end for each
return action a correspond to max p in P

ADP Stage(state s)
  obtain top 5 winning probability W_{ADP} from ADP(s) ;
  obtain their moves M_{ADP} correspond to W_{ADP};
  return M_{ADP}, W_{ADP}

MCTS Stage(moves M_{ADP})
  for each move m in M_{ADP} **do**
    create m as root node with correspond state s
    obtain w_2 from *MTCS*(m, s)
    add w_2 into W_{MCTS}
  end for each
  return W_{MCTS}

ADP with MCTS for Gomoku

- ADP with MCTS
  - Compared to ADP :
    - Eliminate the neural network evaluation function's “short sight” defect, ensure the accuracy of the search
  - Compared to MCTS :
    - Save a large amount of time to find out the suitable action for Gomoku

Reinforcement Learning for Gomoku

- References:
    - Self-teaching adaptive dynamic programming for Gomoku
    - ADP with MCTS Algorithm for Gomoku

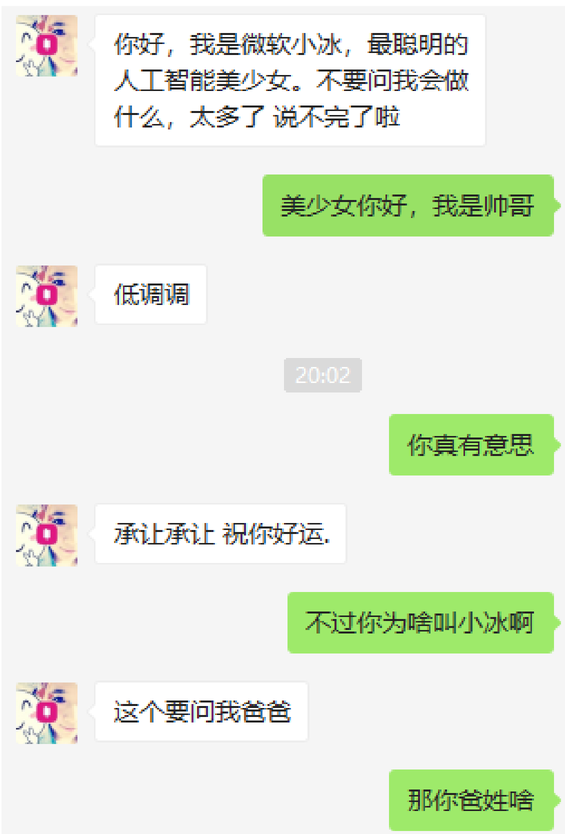
POMDP for Spoken Dialog Systems

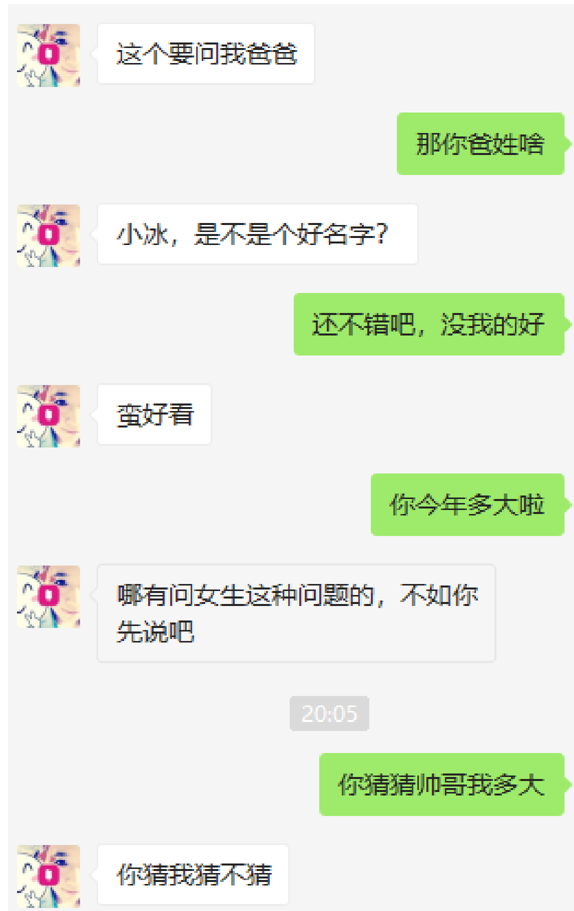
POMDP for Spoken Dialog Systems

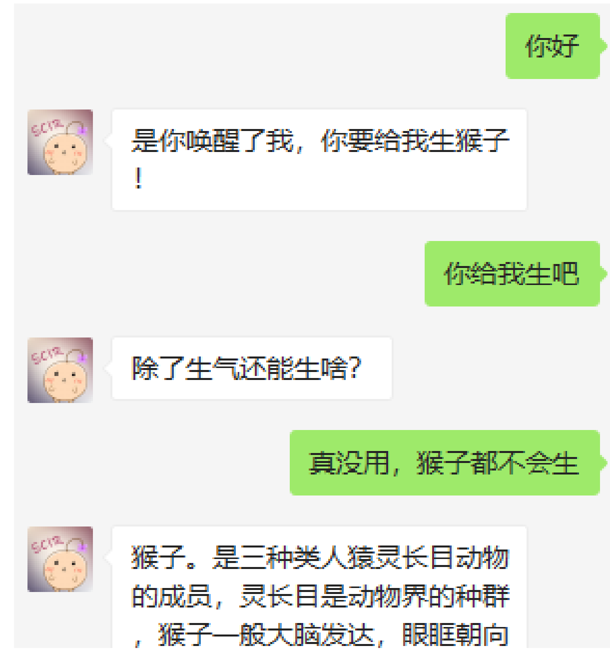
- Spoken Dialog System(SDS)
- Partially Observable Markov Decision Process(POMDP)
- POMDP-Based SDSs

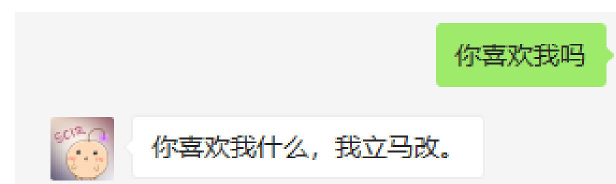
Spoken Dialog System(SDS)

- Chatbot vs SDS





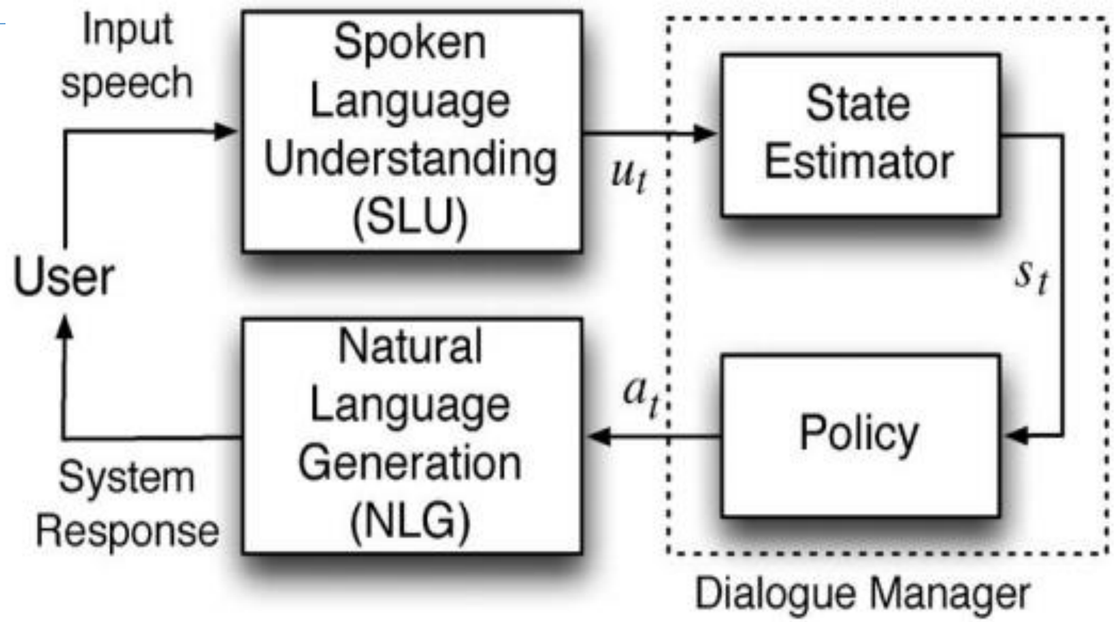




Spoken Dialog System(SDS)

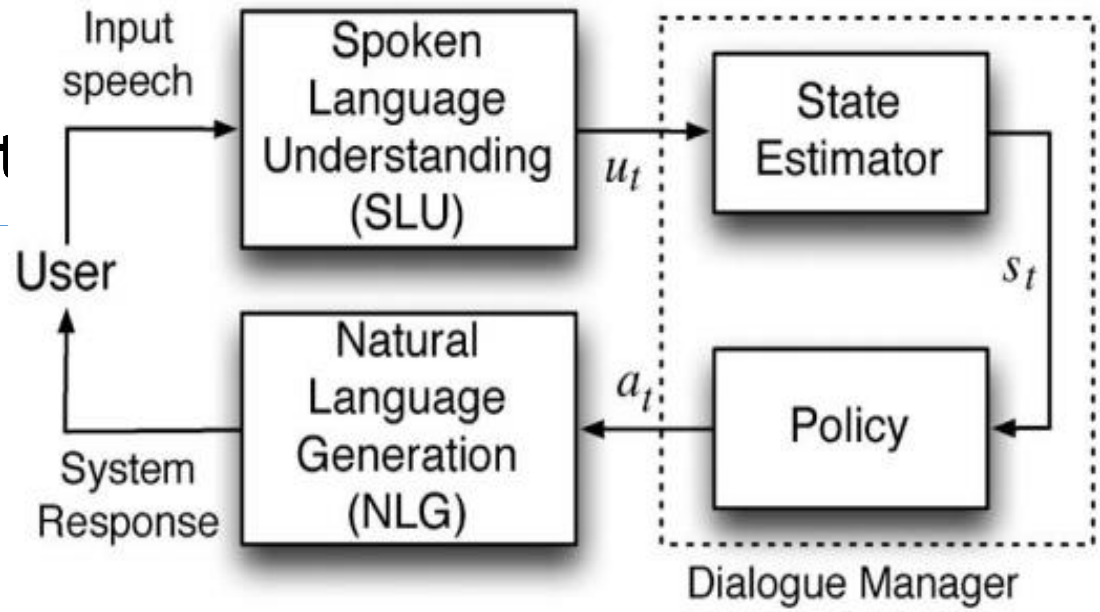
- Chatbot vs SDS
  - Compared to chatbot, SDS is task-oriented and user has an intent

Spoken Dialog System(SDS)



- Structure
  - SLU: Map user's input speech into some abstract semantic representation u_t
  - NLG: Map the action a_t (semantic representation of DM's decisions) into response speech
  - DM: Make decision according to u_t, which is our focus

Spoken Dialog Syst



- Working process
  - At each turn t, SLU component converts each spoken input into an abstract semantic representation called a user dialog act u_t
  - The system updates its internal state s_t and a deterministic decision rule called a policy maps the state into an action a_t
  - The system act at is then converted back into speech via NLG component

- One approach is to fill the slot-value pairs until all slots are filled
- Example: travel booking system
  - DM: {from: _ to: _ time: _ date: _ people: _}
  - Input speech: "I want to go from Paris to London"
  - u_t: {from: Paris to: London}
  - a_t: {time: _}
  - System response: "When do you want to go? "

Spoken Dialog System(SDSs)

- Problems
  - Large cost of laboriously handcrafting complex dialog manager
  - Speech recognition error rate is still in the range 15%~30%, which make it fragile in operation
- Solution
  - Take a good statistical approach such as MDP find a optimal policy through Reinforcement learning
  - However, MDP assume that the entire state is observable

POMDP

- Definition
  - A MDP is defined as a tuple (S, A, T, R, γ)
  - A POMDP is defined as a tuple (S, A, T, R, O, Z, γ, b_0)
    - S is a set of states with $\mathrm{s} \in \mathrm{S}$; A is a set of actions with $\mathrm{a} \in \mathrm{A}$
    - T defines a transition probability of environment P($s_t|s_{t-1}, a_{t-1}$)
    - R defines the reward r(s_t, a_t) $\in \mathrm{R}$
    - O is a set of observations with $\mathrm{o} \in \mathrm{O}$
    - Z defines an observation probability P($o_t|s_t, a_{t-1}$)
    - b_0 is an initial belief state

POMDP

- Definition
  - The interpretation state s_t is not directly observable
  - A belief state b_t is maintained where $b_t(s_t)$ indicates the probability of being in a particular state s_t, reflecting the uncertainty of the true environment state

POMDP

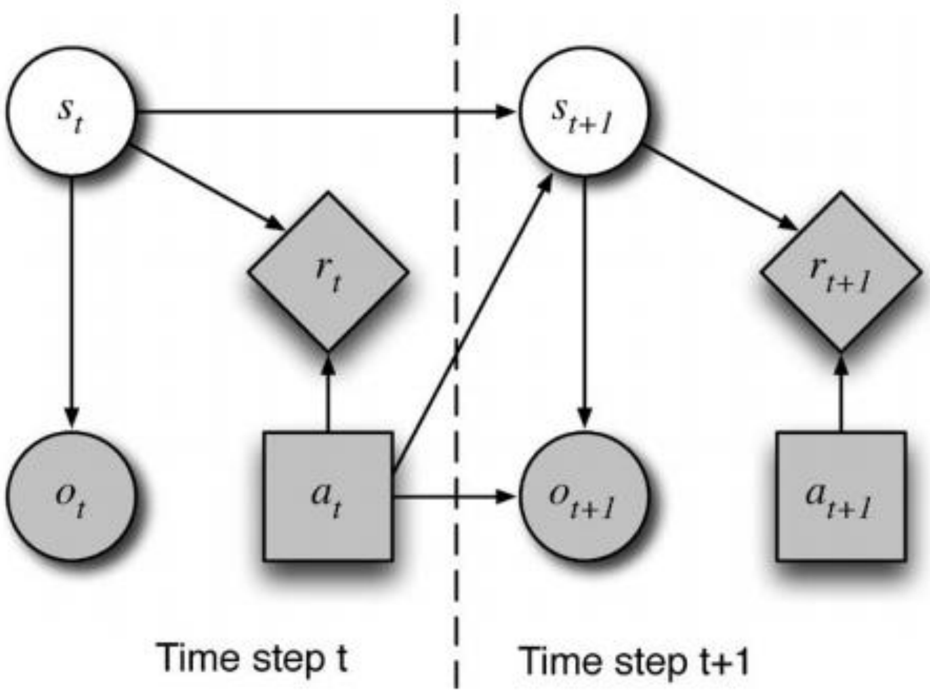
- Definition
  - The goal is to take actions at each time step that maximize its expected future discounted reward:

$$E\left[\sum_{t=0}^{\infty} \gamma^t r_t\right]$$

  - Because of the uncertainty of the true environment state， the agent must make decisions based on belief state

POMDP

- Influence diagram



POMDP

- Belief Monitoring
  - Through interacting with the environment and receiving observations, the agent may update its belief in the true state(belief distribution of true state)

POMDP

$$b_{t+1}(s_{t+1}) = p(s_{t+1}|o_{t+1}, a_t, \boldsymbol{b}) = \frac{p(o_{t+1}|s_{t+1}, a_t, b)\,p(s_{t+1}|a_t, \boldsymbol{b})}{p(o_{t+1}|a_t, \boldsymbol{b})}$$

$$= \frac{p(o_{t+1}|s_{t+1}, a_t) \sum_{s_t} p(s_{t+1}|a_t, \boldsymbol{b}, s_t)\,p(s_t|a_t, \boldsymbol{b})}{p(o_{t+1}|a_t, \boldsymbol{b})}$$

$$= k \cdot p(o_{t+1}|s_{t+1}, a_t) \sum_{s_t} p(s_{t+1}|a_t, s_t)\,b_t(s_t)$$

k = $1/p(o_{t+1}|a_t, \boldsymbol{b})$ is a normalizing constant with

$$p(o_{t+1}|a_t, \boldsymbol{b}) = \sum_{s_{t+1}} p(o_{t+1}|s_{t+1}, a_t) \sum_{s_t} p(s_{t+1}|a_t, s_t)\,b_t(s_t)$$

POMDP

- Belief MDP
  - A Markovian belief state allows a POMDP to be formulated as a MDP where every belief state is a state
  - Belief MDP is defined as a tuple (B, A, τ, r, γ)
    - B is the set of belief states over the POMDP states
    - A is the same finite set of action as for the original POMDP
    - τ is the belief state transition function
    - r is reward function on belief states, r(b_t, a_t)

POMDP

- Belief MDP
  - τ and r need to be derived from original POMDP

$$\tau(b_{t+1}|a_t, b_t) = \sum_{o_{t+1}} p(b_{t+1}|b_t, a_t, o_{t+1})p(o_{t+1}|a_t, b_t)$$

$$\mathrm{r}(b_t, a_t) = \sum_{s_t} R(s_t, a_t)b_t(\mathrm{s}_t)$$

  - The belief MDP is not partially observable anymore, since at any given time the agent knows its belief state

POMDP

- Optimizing POMDP
  - Value function

$$V^{\pi}(b_t) = r(b_t, \pi(b_t)) + \gamma \sum_{o_{t+1}} P(o_{t+1} | b_t, \pi(b_t)) V^{\pi}(b_{t+1})$$

  - Bellman optimality equation

$$V^*(b_t) = \max_{a_t} \left[r(b_t, a_t) + \gamma \sum_{o_{t+1}} P(o_{t+1} | b_t, a_t) V^*(b_{t+1}) \right]$$

  - Optimal Policy

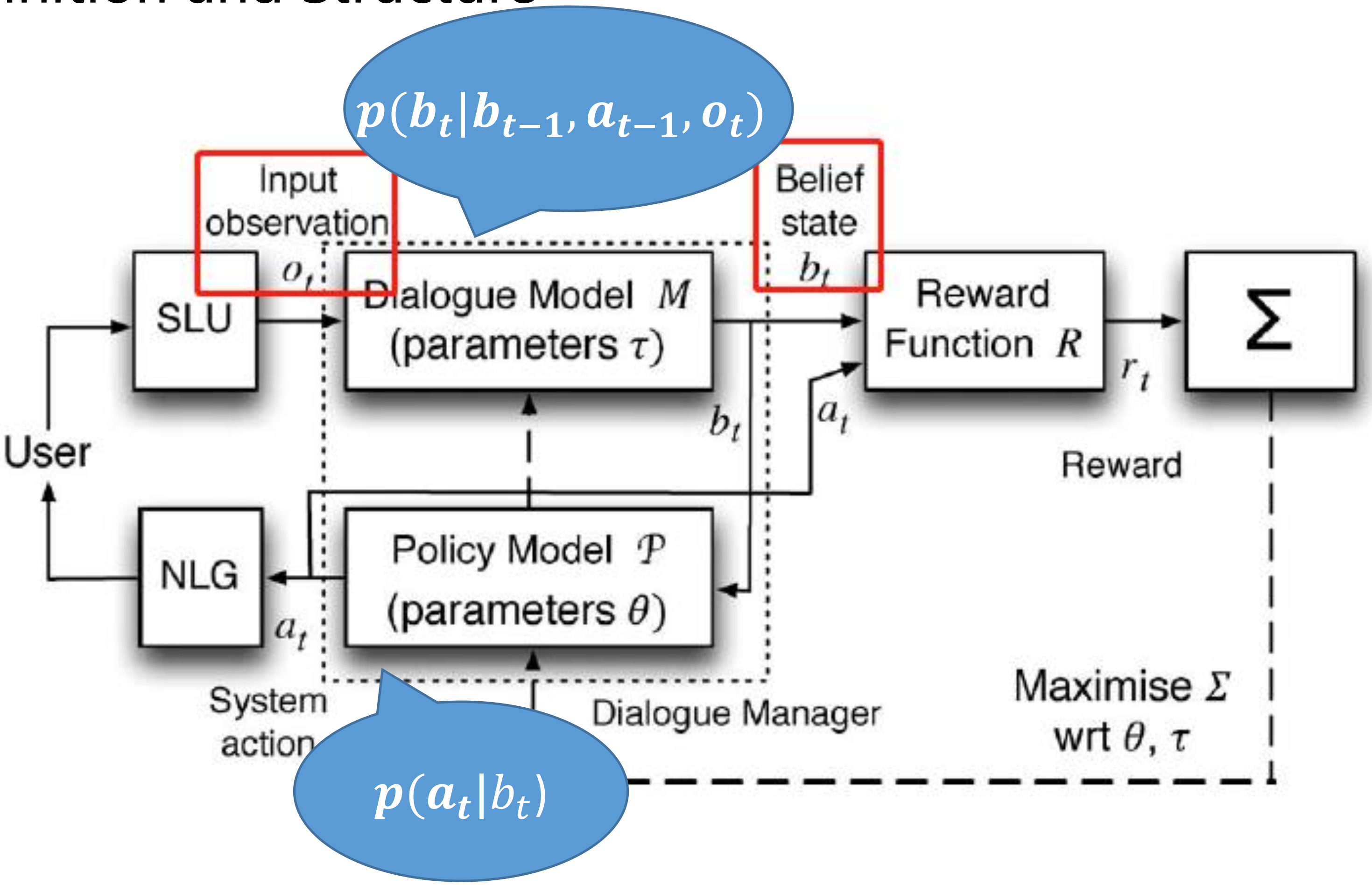
$$\pi^*(b_t) = argmax_{a_t} \left[r(b_t, a_t) + \gamma \sum_{o_{t+1}} P(o_{t+1} | b_t, a_t) V^*(b_{t+1}) \right]$$

POMDP-Based SDSs

- Why POMDP?
  - Modelling the inherent uncertainty of user utterances (provide robustness against the speech recognition errors)
  - Reduce the cost of laboriously handcrafting complex dialog managers

POMDP-Based SDSs

- Definition and Structure



POMDP-Based SDSs

- User Simulators
  - Why need?

    Algorithm requires around 10^5 dialogs to optimize a policy,

    However the availability of large diverse corpora and/or large numbers of real users willing to interact with a partially trained dialog system is normally limited

POMDP-Based SDSs

- User Simulators
  - Operations
    - Given a sequence of user acts and system responses, the aim is to model the distribution of plausible user responses from which an actual user response can be sampled

$$p(u_t | a_t, u_{t-1}, a_{t-1}, u_{t-2}, \ldots)$$

POMDP-Based SDSs

- User Simulators
  - N-gram based on Information State
  - Treat a dialogue as a sequence of pairs of speech acts and tasks

POMDP-Based SDSs

- User Simulators
  - It takes as input the n-1 most recent<speech act, task> pairs in the dialogue history, and uses the statistics of n-grams in the training set to decide on the next user action
  - If no n-grams match the current history, the model can back-off to smaller n-grams

POMDP-Based SDSs

- User Simulators
  - Example： flight reservation
  - History：

ACT： opening， instruction, request_info,
[provide _info], implicit_confirm, request_info,
[provide _info], implicit_confirm, request_info

TASK: meta_greeting_goobbye, meta_instruct, orig_city,
[orig_city], orig_city, dest_city,
[dest_city], orig_dest_city, depart_date

POMDP-Based SDSs

- User Simulators
  - An example 3-gram that would lead to the user action shown before

    {

    <system, implicit_confirm, orig_dest_city >,

    < system, request_info, depart_date >,

    < user, [(provide_info, depart_date), (provide_info, depart_time)] >

    }

POMDP-Based SDSs

- User Simulators
  - N-gram need a large amount of handcrafting
  - Dynamic Bayesian network
  - Hidden Markov Model
  - Train a POMDP-based dialog system to behave like a user

POMDP-Based SDSs

- Problems

  Standard POMDP do not scale to the complexity needed to represent a real-world SDSs, because the number of states, actions, and observations can each easily be extremely large

- Solution
  - Approximation

POMDP-Based SDSs

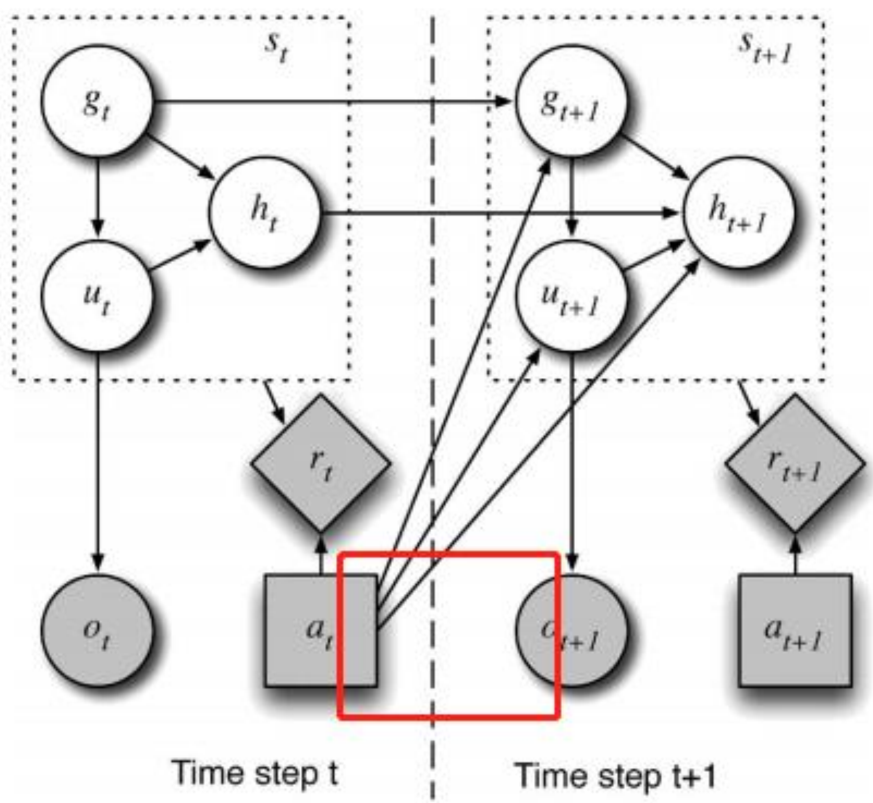
- Dialog Model Representation(Belief state)
  - Factorization

$$s_t = (g_t, u_t, h_t)$$

  - The user's goal g_t, the intent of the most recent user utterance u_t, and the dialog history h_t

POMDP-Based SDSs

- Influence Diagram



POMDP-Based SDSs

- Dialog Model Representation
  - Plugging the factorization into the belief update

$$
\begin{aligned}
b_{t+1}(g_{t+1}, u_{t+1}, h_{t+1}) = & \eta P(o_{t+1} | u_{t+1}) \\
& \cdot P(u_{t+1} | g_{t+1}, a_t) \\
& \cdot \sum_{g_t} P(g_{t+1} | g_t, a_t) \\
& \cdot \sum_{h_t} P(h_{t+1} | g_{t+1}, u_{t+1}, h_t, a_t) \\
& \cdot b_t(g_t, h_t).
\end{aligned}
$$

POMDP-Based SDSs

- Dialog Model Representation
  - Benefits of Factorization
    - Reduce the dimensions of the state transition matrix
    - Reduce the number of conditional dependencies

POMDP-Based SDSs

- Dialog Model Representation
  - Further Approximation
    - N-best approach with pruning and recombination
    - The factored Bayesian network approach

POMDP-Based SDSs

- Policy Model Representation
  - Policy
    - A mapping between belief state b and appropriate system action a
  - Objective
    - To find an optimal policy π^* that maximizes the expected sum of discounted rewards at the end of the dialog

POMDP-Based SDSs

- Policy Model Representation
  - Summary space
    - A compressed feature space in which both states and actions are simplified
    - A subspace of the full master space
    - Belief monitoring is performed in master space
    - Decision taking and policy optimization take place in summary space

POMDP-Based SDSs

- Policy Model Representation
  - Operations of summary-space POMDP
    - After belief updating, the belief state b in master space is mapped to a vector of features $\hat{b}$ and a corresponding set of candidate actions $\{\hat{a}\}$
    - The policy is then used to select the best action to take $\hat{b} \longrightarrow \hat{a}$ from the set of candidate actions and a second heuristic is used to map $\hat{a}$ back into a full action a in master space

POMDP-Based SDSs

- Policy Optimization
  - The Policy is now a function of the summary belief state and actions $\hat{b} \longrightarrow \hat{a}$
  - To find a optimal Q-function(for a deterministic policy)

$$\pi^*(\hat{b}) = \arg\max_{\hat{a}} \{Q^*(\hat{b}, \hat{a})\}.$$

POMDP-Based SDSs

- Policy Optimization
  - Planning Under Uncertainty
  - Value Iteration
  - Monte Carlo Optimization
  - Least Squares Policy Iteration
  - Natural Actor-Critic Optimization

POMDP for Spoken Dialog Systems

- References:
  - USING POMDPS FOR DIALOG MANAGEMENT
  - POMDP-Based Statistical Spoken Dialog Systems: A Review
  - Partially Observable Markov Decision Processes for Spoken Dialog Systems
  - User Simulation for Spoken Dialogue Systems: Learning and Evaluation
  - Learning User Simulations for Information State Update Dialogue Systems
  - Error Handling in Spoken Dialogue Systems

Lab3

Lab3

- Problem:
  - Solve the Grid World Problem based on MDP
- Requirement:
  - Print optimal value of all states using value iteration and policy iteration
- Address: http://10.88.3.60/problem.php?id=1002

Lab3

- Value Iteration

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
rewards $R(s)$, discount γ
ϵ, the maximum error allowed in the utility of any state
local variables: U, U', vectors of utilities for states in S, initially zero
δ, the maximum change in the utility of any state in an iteration

repeat
$U \leftarrow U'$; $\delta \leftarrow 0$
for each state s **in** S **do**
$$U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$$
if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
until $\delta < \epsilon(1-\gamma)/\gamma$
return U

Figure 17.4 The value iteration algorithm for calculating utilities of states. The termination condition is from Equation (17.8).

Lab3

- Policy Iteration

function POLICY-ITERATION(mdp) **returns** a policy
inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
local variables: U, a vector of utilities for states in S, initially zero
π, a policy vector indexed by state, initially random

repeat
$U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
$unchanged?$ $\leftarrow$ true
for each state s **in** S **do**
if $\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\, U[s']$ **then do**
$\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']$
$unchanged?$ $\leftarrow$ false
until $unchanged?$
return π

Figure 17.7 The policy iteration algorithm for calculating an optimal policy.