高级人工智能

第十章

强化学习
Reinforcement Learning

史忠植

中国科学院计算技术研究所

内容提要

- 引言
- 强化学习模型
- 动态规划
- 蒙特卡罗方法
- 时序差分学习
- Q学习
- 强化学习中的函数估计
- 应用

引言

人类通常从与外界环境的交互中学习。所谓强化（reinforcement）学习是指从环境状态到行为映射的学习，以使系统行为从环境中获得的累积奖励值最大。在强化学习中，我们设计算法来把外界环境转化为最大化奖励量的方式的动作。我们并没有直接告诉主体要做什么或者要采取哪个动作,而是主体通过看哪个动作得到了最多的奖励来自己发现。主体的动作的影响不只是立即得到的奖励，而且还影响接下来的动作和最终的奖励。试错搜索(trial-and-error search)和延期强化(delayed reinforcement)这两个特性是强化学习中两个最重要的特性。

引言

强化学习技术是从控制理论、统计学、心理学等相关学科发展而来，最早可以追溯到巴甫洛夫的条件反射实验。但直到上世纪八十年代末、九十年代初强化学习技术才在人工智能、机器学习和自动控制等领域中得到广泛研究和应用，并被认为是设计智能系统的核心技术之一。特别是随着强化学习的数学基础研究取得突破性进展后，对强化学习的研究和应用日益开展起来，成为目前机器学习领域的研究热点之一。

引言

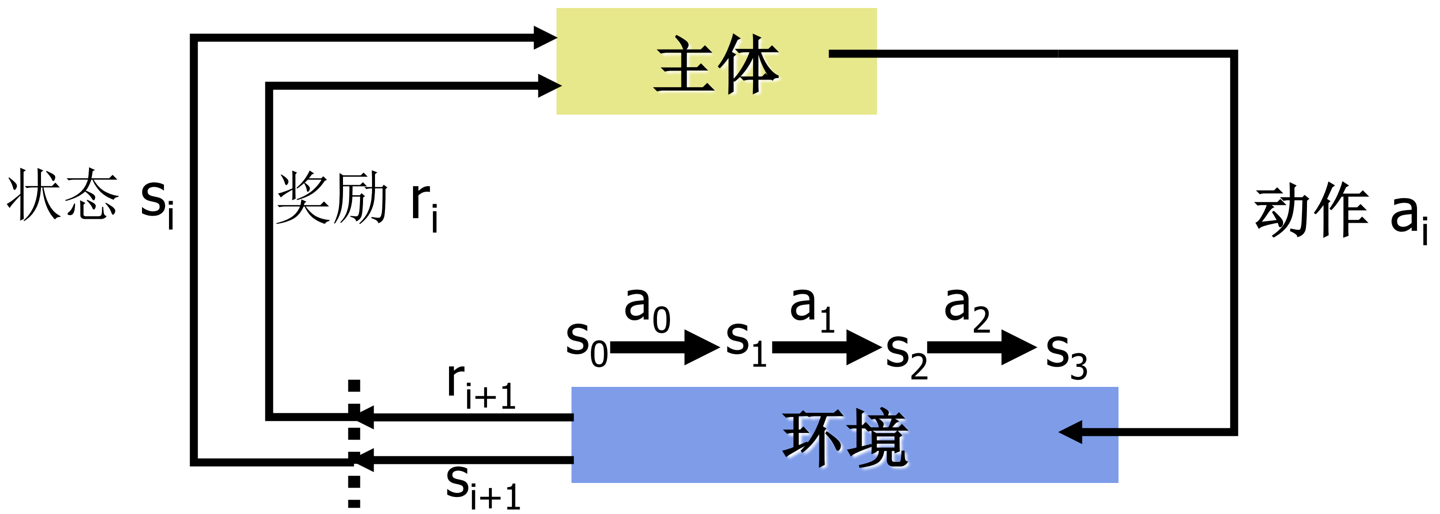
- 强化思想最先来源于心理学的研究。1911年Thorndike提出了效果律（Law of Effect）：一定情景下让动物感到舒服的行为，就会与此情景增强联系（强化），当此情景再现时，动物的这种行为也更易再现；相反，让动物感觉不舒服的行为，会减弱与情景的联系，此情景再现时，此行为将很难再现。换个说法，哪种行为会“记住”，会与刺激建立联系，取决于行为产生的效果。
- 动物的试错学习,包含两个含义：选择（selectional）和联系（associative），对应计算上的搜索和记忆。所以，1954年，Minsky在他的博士论文中实现了计算上的试错学习。同年，Farley和Clark也在计算上对它进行了研究。强化学习一词最早出现于科技文献是1961年Minsky 的论文“Steps Toward Artificial Intelligence”，此后开始广泛使用。1969年， Minsky因在人工智能方面的贡献而获得计算机图灵奖。

引言

- 1953到1957年，Bellman提出了求解最优控制问题的一个有效方法：动态规划（dynamic programming）
- Bellman于 1957年还提出了最优控制问题的随机离散版本，就是著名的马尔可夫决策过程（MDP, Markov decision processe），1960年Howard提出马尔可夫决策过程的策略迭代方法，这些都成为现代强化学习的理论基础。
- 1972年，Klopf把试错学习和时序差分结合在一起。1978年开始，Sutton、Barto、 Moore，包括Klopf等对这两者结合开始进行深入研究。
- 1989年Watkins提出了Q-学习[Watkins 1989]，也把强化学习的三条主线扭在了一起。
- 1992年，Tesauro用强化学习成功了应用到西洋双陆棋（backgammon）中，称为TD-Gammon 。

内容提要

强化学习模型



主体
状态 s_i
奖励 r_i
动作 a_i
$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3$
r_{i+1}
s_{i+1}
环境

i: input

r: reward

s: state

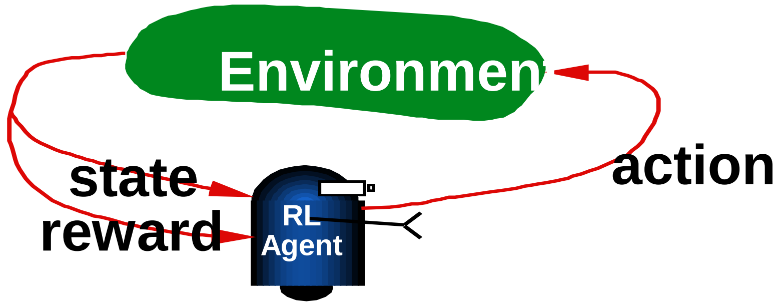
a: action

描述一个环境（问题）

- Accessible vs. inaccessible
- Deterministic vs. non-deterministic
- Episodic vs. non-episodic
- Static vs. dynamic
- Discrete vs. continuous

The most complex general class of environments are inaccessible, non-deterministic, non-episodic, dynamic, and continuous.

强化学习问题

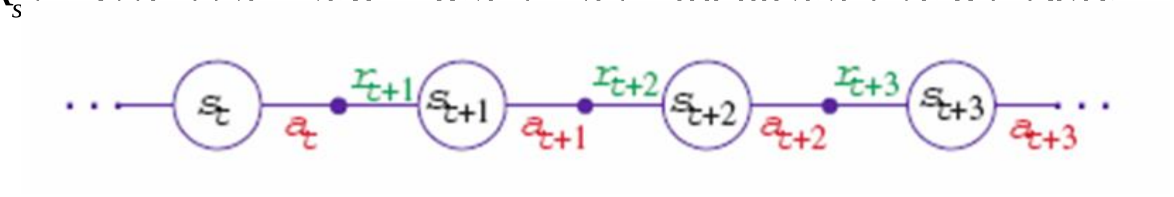


- Agent-environment interaction
  - States, Actions, Rewards
- To define a finite MDP
  - **state and action sets : S and A**
  - **one-step “dynamics” defined by transition probabilities (Markov Property):**

$$P_{ss'}^{a} = \Pr\left\{s_{t+1} = s' \mid s_t = s, a_t = a\right\} \quad \text{for all } s, s' \in S,\ a \in A(s).$$

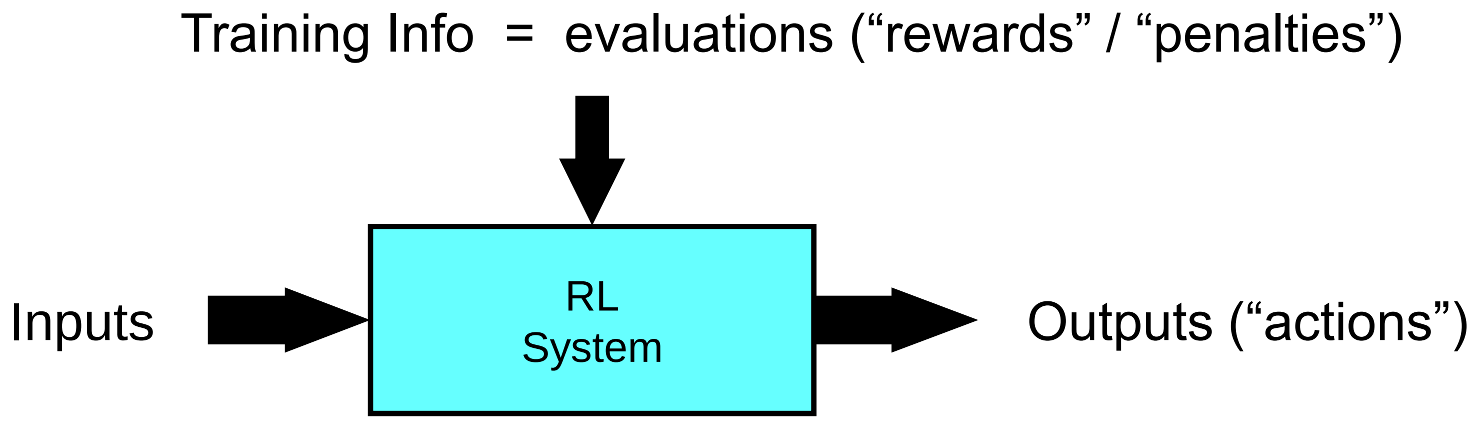
  - **reward probabilities:**

$$R_{ss'}^{a} = E\left\{r_{t+1} \mid s_t = s, a_t = a, s_{t+1} = s'\right\} \quad \text{for all } s, s' \in S,\ a \in A(s)$$



与监督学习对比

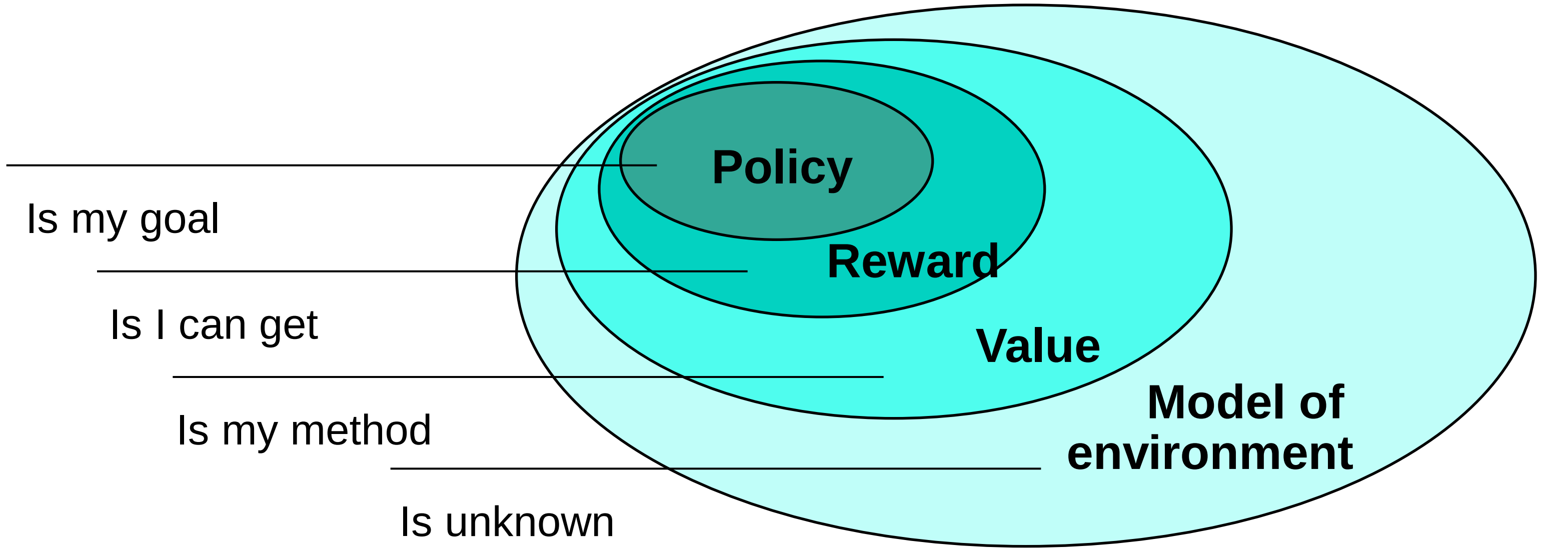
- **Supervised Learning – Learn from examples provided by a knowledgable external supervisor.**



- **Reinforcement Learning – Learn from interaction**
  - **learn from its own experience, and the objective is to get as much reward as possible. *The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them*.**

强化学习要素

- Policy: **stochastic rule for selecting actions**
- Return/Reward: **the function of future rewards agent tries to maximize**
- Value: **what is good because it predicts reward**
- Model: **what follows what**



在策略Π下的Bellman公式

The basic idea:

$$\begin{aligned} R_t &= r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} \cdots \\ &= r_{t+1} + \gamma\left(r_{t+2} + \gamma r_{t+3} + \gamma^2 r_{t+4} \cdots\right) \\ &= r_{t+1} + \gamma R_{t+1} \end{aligned}$$

γ is the discount rate

So:

$$\boxed{\begin{aligned} V^\pi(s) &= E_\pi\left\{R_t \middle| s_t = s\right\} \\ &= E_\pi\left\{r_{t+1} + \gamma V(s_{t+1}) \middle| s_t = s\right\} \end{aligned}}$$

Or, without the expectation operator:

$$V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'} \left[R^a_{ss'} + \gamma V^\pi(s')\right]$$

Bellman最优策略公式

$$
\begin{aligned}
V^*(s) &= \max_{a \in A(s)} E\left\{ r_{t+1} + \gamma V^*(s_{t+1}) \middle| s_t = s, a_t = a \right\} \\
&= \max_{a \in A(s)} \sum_{s'} P_{ss'}^{a} \left[R_{ss'}^{a} + \gamma V^*(s') \right]
\end{aligned}
$$

其中：

V^*： 状态值映射

S： 环境状态

R： 奖励函数

P： 状态转移概率函数

γ： 折扣因子

马尔可夫决策过程
MARKOV DECISION PROCESS

由四元组<S，A，R，P>定义。

- 环境状态集S
- 系统行为集合A
- 奖励函数R：$S\times A\rightarrow\Re$
- 状态转移函数P：$S\times A\rightarrow PD(S)$

记$R(s, a, s')$为系统在状态s采用a动作使环境状态转移到s'获得的瞬时奖励值；记$P(s, a, s')$为系统在状态s采用a动作使环境状态转移到s'的概率。

马尔可夫决策过程 MARKOV DECISION PROCESS

- 马尔可夫决策过程的本质是：当前状态向下一状态转移的概率和奖励值只取决于当前状态和选择的动作，而与历史状态和历史动作无关。因此在已知状态转移概率函数*P*和奖励函数*R*的环境模型知识下，可以采用动态规划技术求解最优策略。而强化学习着重研究在*P*函数和*R*函数未知的情况下，系统如何学习最优行为策略。

MARKOV DECISION PROCESS

Characteristics of MDP:

a set of states : $\mathcal{S}$

a set of actions : $\mathcal{A}$

a reward function :

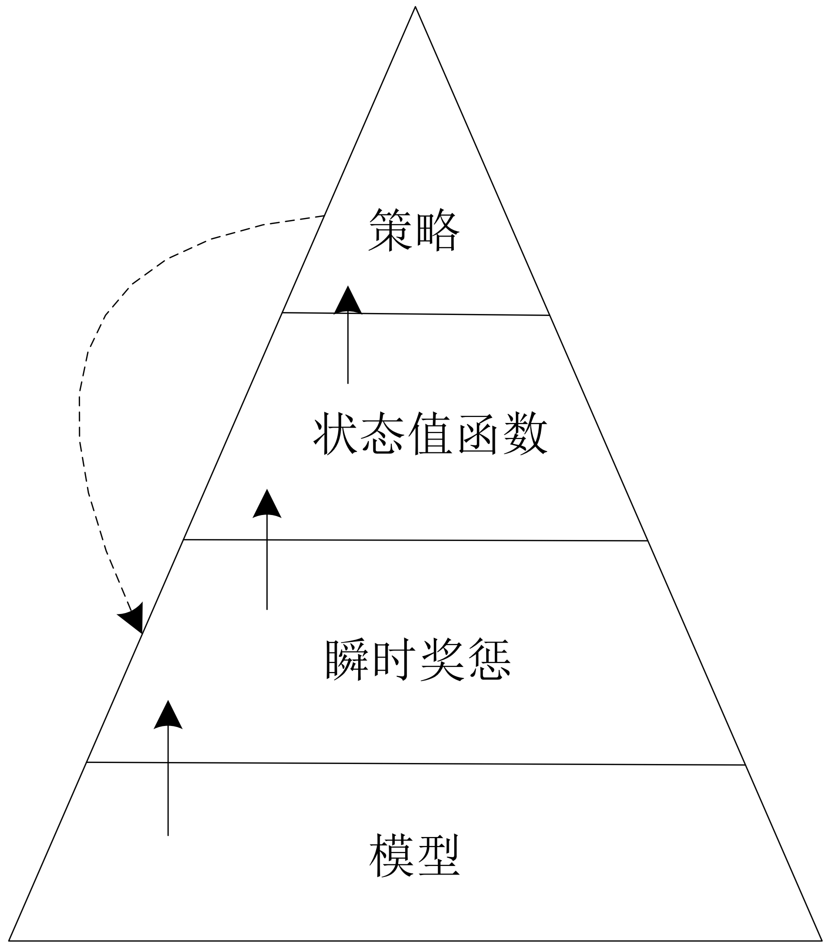
$\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$

A state transition function:

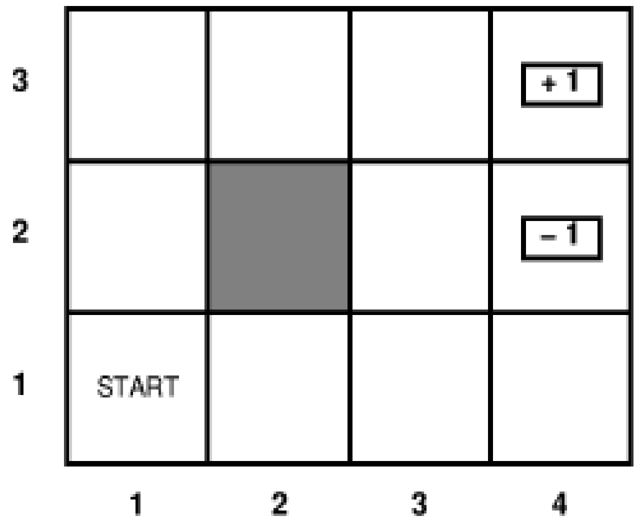
$\mathcal{T}: \mathcal{S} \times \mathcal{A} \rightarrow \prod(\mathcal{S})$

$\mathcal{T}$(s,a,s'): probability of transition from s to s' using action a

马尔可夫决策过程
MARKOV DECISION PROCESS



MDP EXAMPLE:



States and rewards

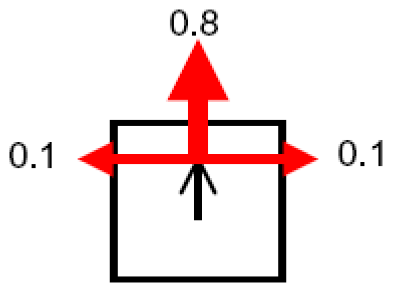
$$V^*(s) = \max_\pi E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right)$$

Bellman Equation:

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

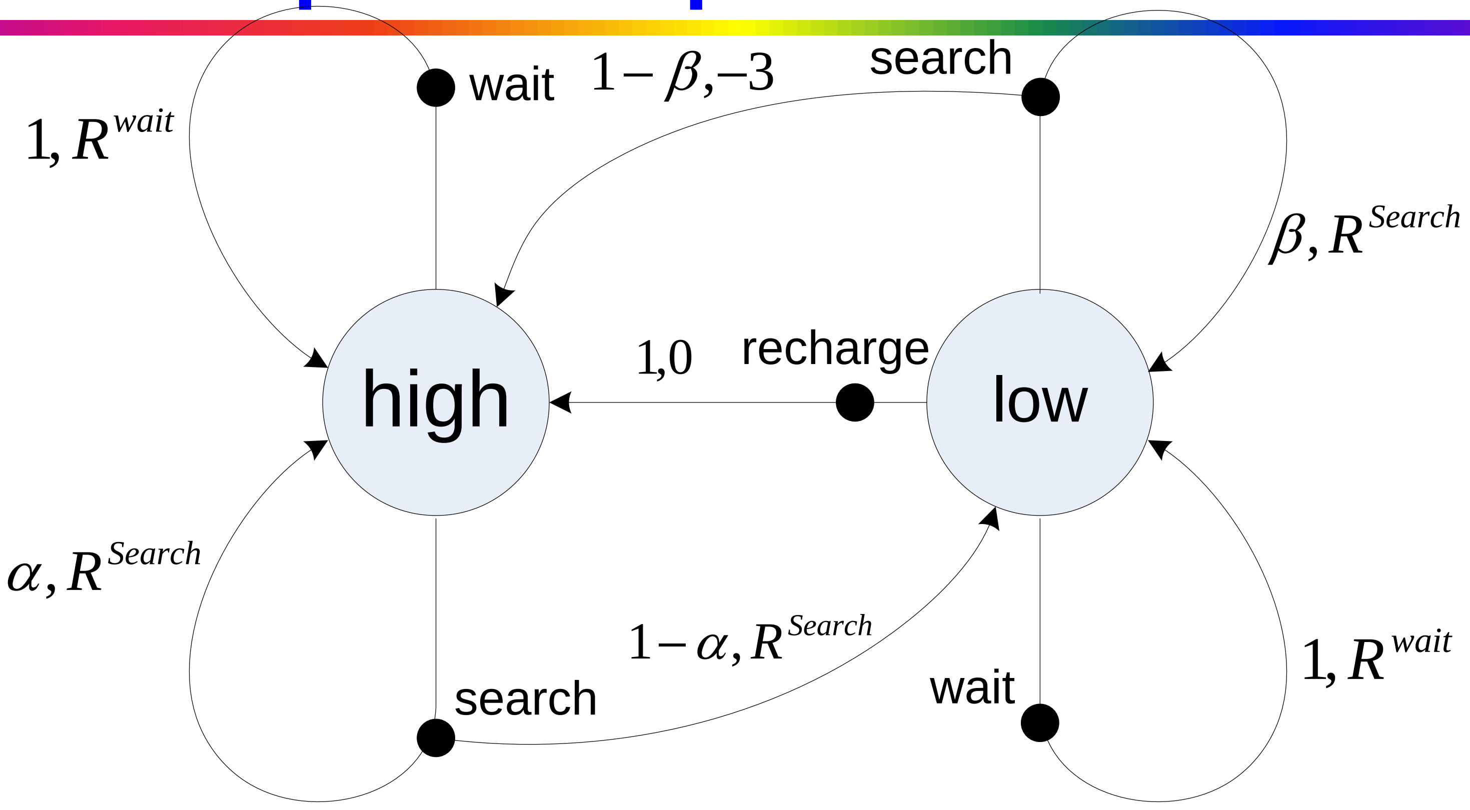
$$\pi^*(s) = \arg\max_a \left(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s') \right)$$

(Greedy policy selection)



Transition function

MDP Graphical Representation



β, α : T (s, action, s’)

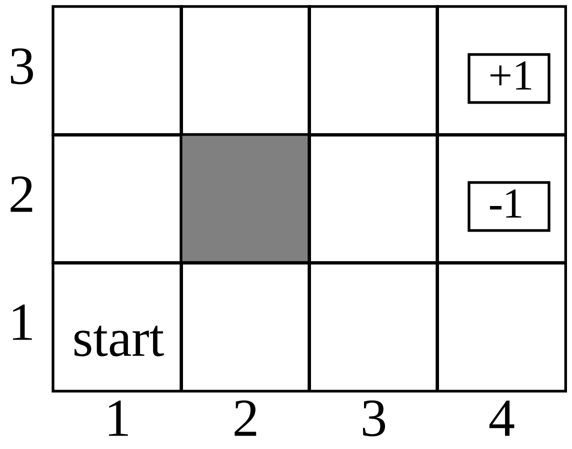
Similarity to Hidden Markov Models (HMMs)

Reinforcement Learning …

$$\begin{tikzcd} & \text{Deterministic transitions} \\ {} \arrow[ur, dash] \arrow[dr, dash] & \\ & \text{Stochastic transitions} \end{tikzcd}$$

M_{ij}^{a} is the probability to reaching state j when taking action a in state i

A simple environment that presents the agent with a sequential decision problem:



Move cost = 0.04

(Temporal) credit assignment problem sparse reinforcement problem

Offline alg: action sequences determined ex ante

Online alg: action sequences is conditional on observations along the way;

Important in stochastic environment (e.g. jet flying)

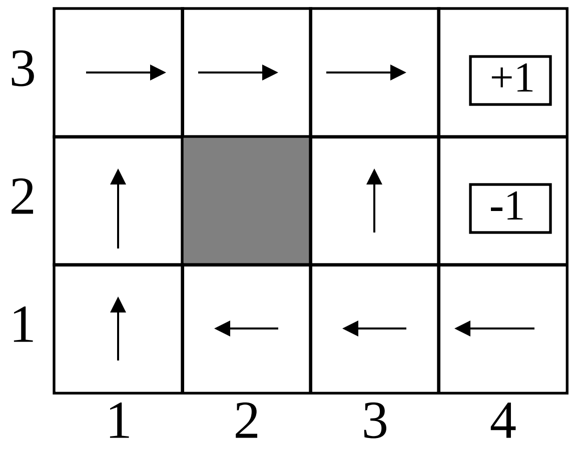
Reinforcement Learning …

M = 0.8 in direction you want to go

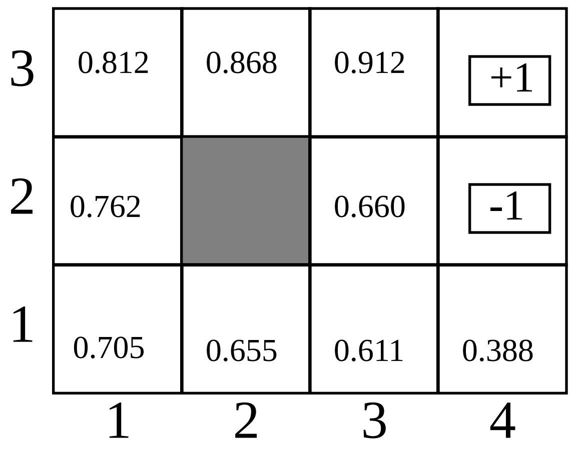
0.2 in perpendicular < 0.1 left / 0.1 right

Policy: mapping from states to actions

An optimal policy for the stochastic environment:



utilities of states:



Environment < Observable (accessible): percept identifies the state / Partially observable

Markov property: Transition probabilities depend on state only, not on the path to the state.

Markov decision problem (MDP).

Partially observable MDP (POMDP): percepts does not have enough info to identify transition probabilities.

动态规划Dynamic Programming

- 动态规划(dynamic programming)的方法通过从后继状态回溯到前驱状态来计算赋值函数。动态规划的方法基于下一个状态分布的模型来接连的更新状态。强化学习的动态规划的方法是基于这样一个事实：对任何策略*π*和任何状态*s*，有下式迭代**的一致的等式成立**

$$V^{\pi}(s) = \sum_{a} \pi(a \mid s) \times \sum_{s'} \pi(s \rightarrow s' \mid a) \times (R^{a}(s \rightarrow s') + \gamma(V^{\pi}(s')))$$

$\pi(a \mid s)$是给定在随机策略π下状态s时动作a的概率。
$\pi(s \rightarrow s' \mid a)$是在动作$a$下状态$s$转到状态$s'$的概率。这就是对$V^{\pi}$的Bellman(1957)等式。

动态规划

Dynamic Programming - Problem

- A discrete-time dynamic system
  - States {1, … , *n*} + termination state 0
  - Control *U*(*i*)
  - Transition Probability $p_{ij}(u)$
- Accumulative cost structure

$$\gamma^k r(i,u,j), 0<\alpha\le 1$$

- Policies

$$\pi=\{\mu_0,\mu_1,\ldots\}$$

$$p(i_{k+1}=j\,|\,i_k=i)=p_{ij}\left(\mu_k(i)\right)$$

动态规划

Dynamic Programming – Iterative Solution

- Finite Horizon Problem

$$V_N^\pi(i) = E\left[\gamma^N G(i_N) + \sum_{k=0}^{N-1} \gamma^k r(i_k, \mu_k(i_k), i_{k+1}) \middle| i_0 = i\right]$$

- Infinite Horizon Problem

$$V^\pi(i) = \lim_{n\to\infty} E\left[\sum_{k=0}^{N-1} \gamma^k r(i_k, \mu_k(i_k), i_{k+1}) \middle| i_0 = i\right]$$

- Value Iteration

$$T(V(i)) = \max_{u\in U(i)} \sum_{j=0}^{n} p_{ij}(u)(r(i,u,j) + \gamma \cdot V(j)), i = 1, \ldots, n$$

$$V_{k+1}(i) = T(V_k(i)) \qquad V^*(i) = \min_\pi V^\pi(i)$$

动态规划中的策略迭代/值迭代

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \cdots \pi^* \rightarrow V^* \rightarrow \pi^*$$

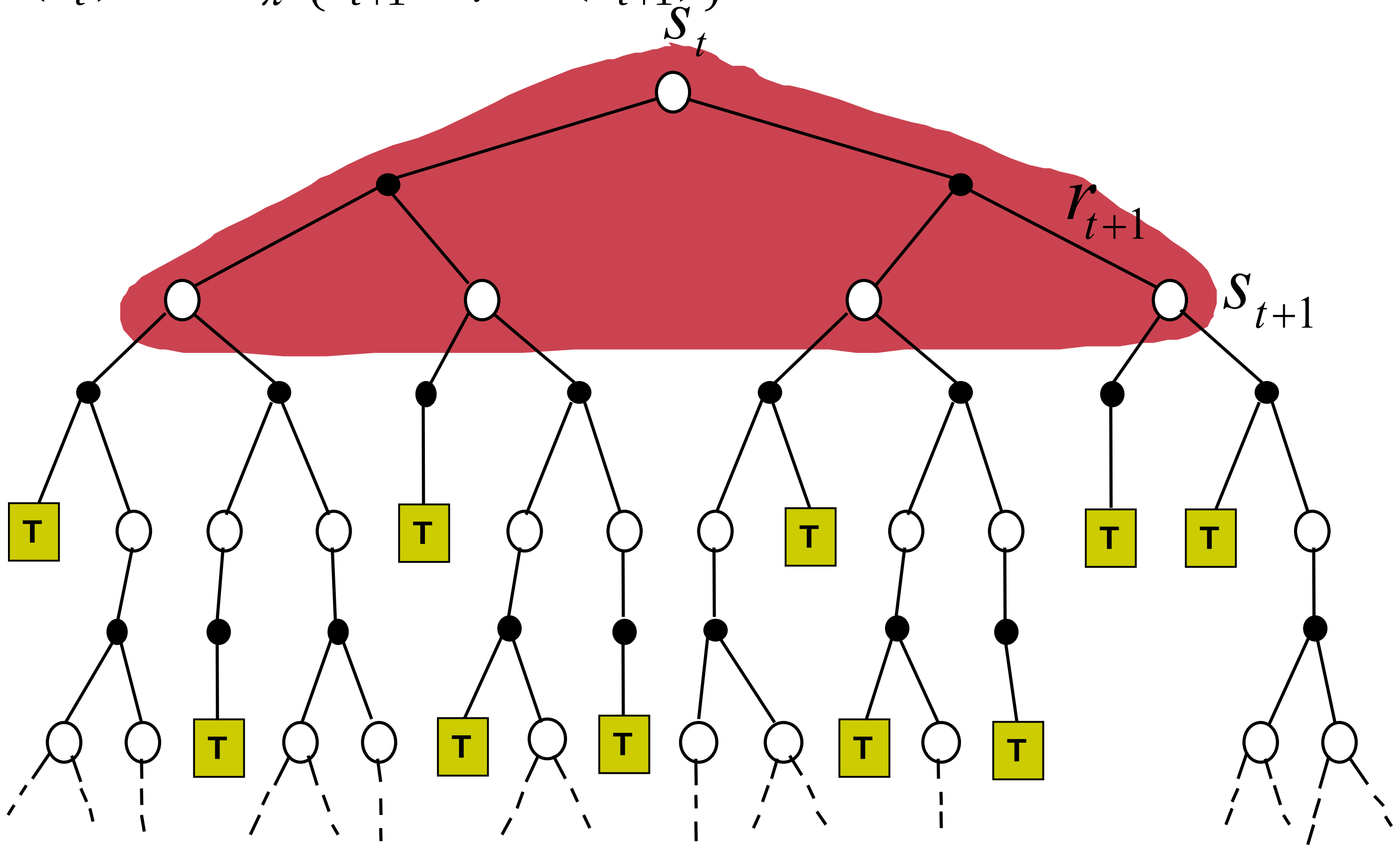
policy evaluation　　policy improvement
"greedification"

$$\left.\begin{aligned} \pi'(s) &= \arg\max_a \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V^\pi(s') \right] \\ V_{k+1}(s) &\leftarrow \sum_a \pi(s,a) \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V_k(s') \right] \end{aligned}\right\} \text{Policy Iteration}$$

$$V_{k+1}(s) \leftarrow \max_a \sum_{s'} P_{ss'}^a \left[R_{ss'}^a + \gamma V_k(s') \right] \Longrightarrow \text{Value Iteration}$$

动态规划方法

$$V(s_t) \leftarrow E_\pi \{ r_{t+1} + \gamma V(s_{t+1}) \}$$



自适应动态规划(ADP)

Idea: use the constraints (state transition probabilities) between states to speed learning.

Solve

$$U(i) = R(i) + \sum_j M_{ij} U(j)$$

= value determination.

No maximization over actions because agent is passive unlike in value iteration.

using DP

→Large state space

e.g. Backgammon: 10^{50} equations in 10^{50} variables

Value Iteration Algorithm

```
initialize $V(s)$ arbitrarily
loop until policy good enough
    loop for $s \in \mathcal{S}$
        loop for $a \in \mathcal{A}$
            $Q(s,a) := R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s')V(s')$
        $V(s) := \max_a Q(s,a)$
    end loop
end loop
```

Stop Iteration when V(s) differs less than **є.**
Policy difference ratio =< 2**єγ** / (1-**γ**)
(Williams & Baird 1993b)

AN ALTERNATIVE ITERATION: (Singh,1993)

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

(Important for model free learning)

Policy Iteration Algorithm

Policies converge faster than values.

```
choose an arbitrary policy $\pi'$
loop
    $\pi := \pi'$
    compute the value function of policy $\pi$:
        solve the linear equations
            $V_\pi(s) = R(s, \pi(s)) + \gamma \sum_{s' \in \mathcal{S}} T(s, \pi(s), s') V_\pi(s')$
    improve the policy at each state:
        $\pi'(s) := \arg\max_a \left( R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') V_\pi(s') \right)$
until $\pi = \pi'$
```

Why faster convergence?

动态规划Dynamic Programming

- 典型的动态规划模型作用有限，很多问题很难给出环境的完整模型。仿真机器人足球就是这样的问题，可以采用实时动态规划方法解决这个问题。在实时动态规划中不需要事先给出环境模型，而是在真实的环境中不断测试，得到环境模型。可以采用反传神经网络实现对状态泛化，网络的输入单元是环境的状态*s*, 网络的输出是对该状态的评价*V(s)*。

没有模型的方法
Model Free Methods

Models of the environment:

$\mathcal{T}: \mathcal{S} \times \mathcal{A} \rightarrow \prod(\mathcal{S})$ and $\mathcal{R}: \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$

Do we know them? *Do we have to know them?*

- Monte Carlo Methods
- Adaptive Heuristic Critic
- Q Learning

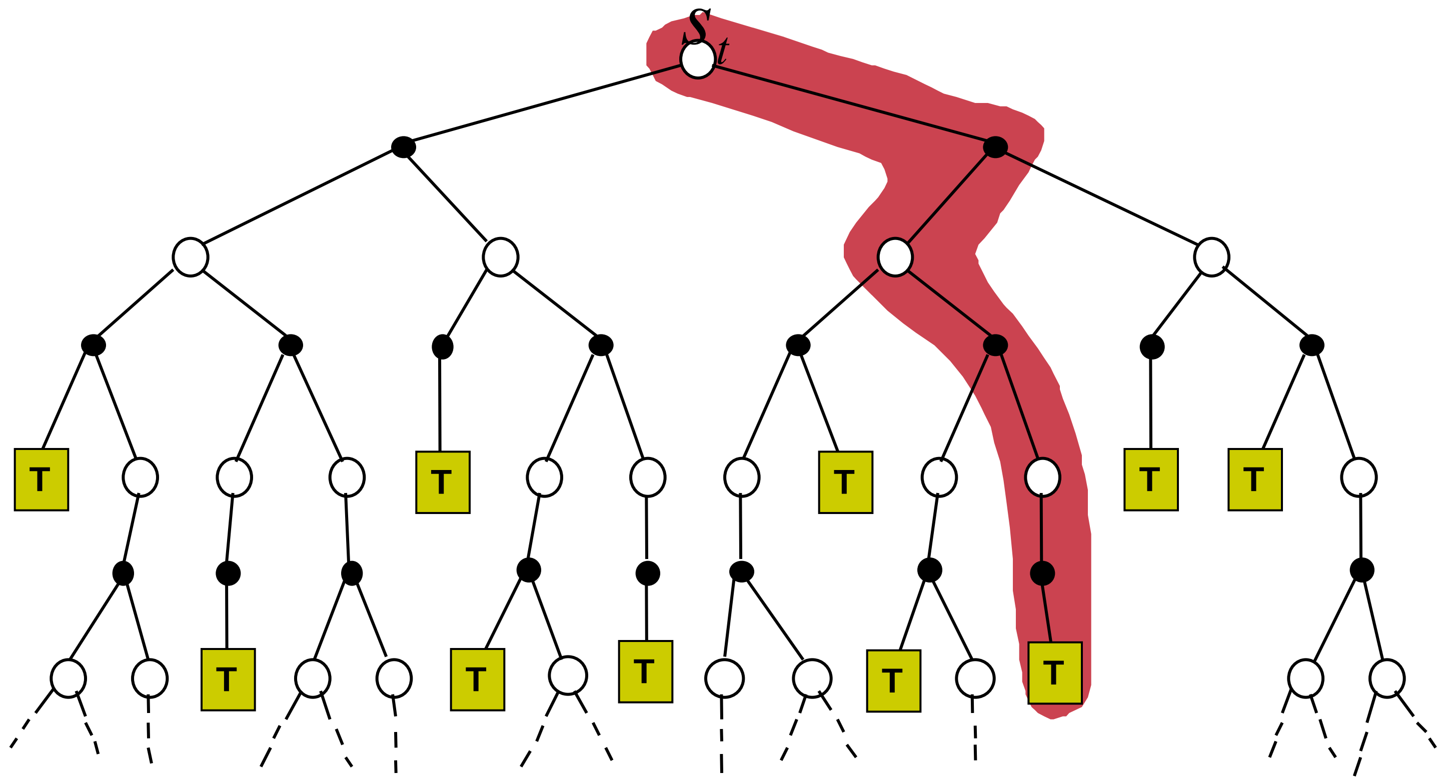
蒙特卡罗方法 Monte Carlo Methods

- 蒙特卡罗方法不需要一个完整的模型。而是它们对状态的整个轨道进行抽样，基于抽样点的最终结果来更新赋值函数。蒙特卡罗方法不需要经验，即从与环境联机的或者模拟的交互中抽样状态、动作和奖励的序列。联机的经验是令人感兴趣的，因为它不需要环境的先验知识，却仍然可以是最优的。从模拟的经验中学习功能也很强大。它需要一个模型，但它可以是生成的而不是分析的，即一个模型可以生成轨道却不能计算明确的概率。于是，它不需要产生在动态规划中要求的所有可能转变的完整的概率分布。

Monte Carlo方法

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

where R_t is the actual return following state s_t.



蒙特卡罗方法 Monte Carlo Methods

- *Idea*:

  Hold statistics about rewards for each state

  Take the average

  This is the V(s)

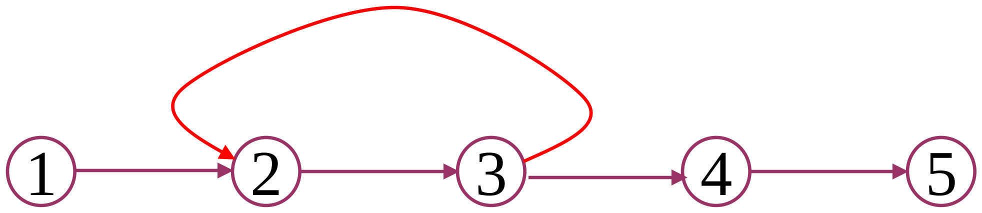
- Based only on experience ☺
- Assumes episodic tasks ☹

  (Experience is divided into episodes and all episodes will terminate regardless of the actions selected.)

- Incremental in episode-by-episode sense not step-by-step sense.

Monte Carlo 策略评价

- *Goal:* learn $V^{\pi}(s)$ under *P* and *R* are unknown in advance
- *Given:* some number of episodes under π which contain *s*
- *Idea:* Average returns observed after visits to *s*



- *Every-Visit MC:* average returns for *every* time *s* is visited in an episode
- *First-visit MC:* average returns only for *first* time *s* is visited in an episode
- Both converge asymptotically

蒙特卡罗方法

Problem: Unvisited <s, a> pairs
(problem of ***maintaining exploration***)

For every <s, a> make sure that:
P(<s, a> selected as a start state and action) >0

(Assumption of ***exploring starts***)

蒙特卡罗控制

How to select Policies:
(Similar to policy evaluation)

$$\begin{tikzcd}[column sep=huge]
\pi \arrow[r, bend left=50, "\overset{\text{evaluation}}{Q \rightarrow Q^{\pi}}"] & Q \arrow[l, bend left=50, "\underset{\text{improvement}}{\pi \rightarrow greedy(Q)}"]
\end{tikzcd}$$

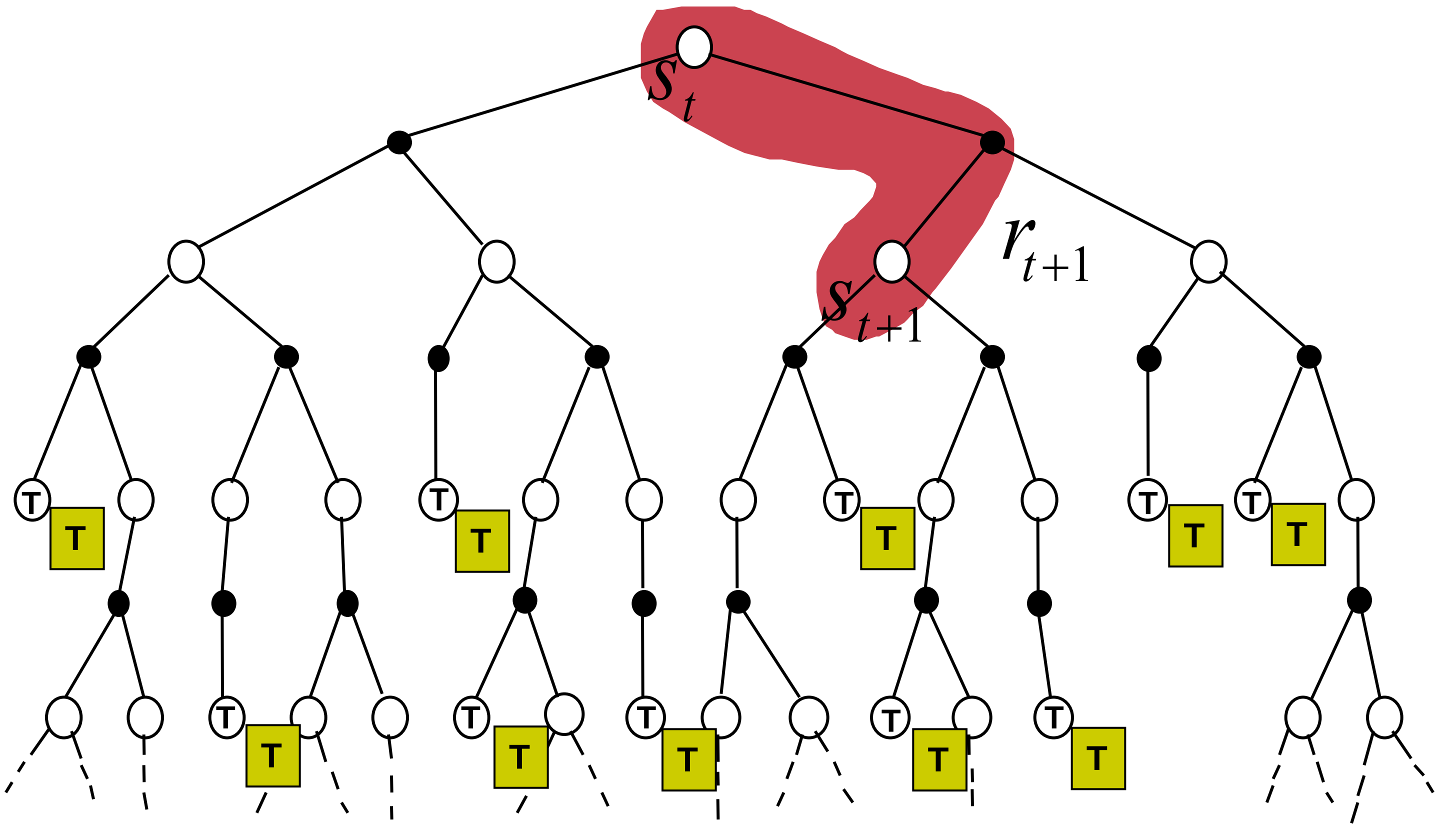
• MC policy iteration: Policy evaluation using MC methods followed by policy improvement
• Policy improvement step: greedify with respect to value (or action-value) function

时序差分学习 Temporal-Difference

时序差分学习中没有环境模型，根据经验学习。每步进行迭代，不需要等任务完成。预测模型的控制算法，根据历史信息判断将来的输入和输出，强调模型的函数而非模型的结构。时序差分方法和蒙特卡罗方法类似，仍然采样一次学习循环中获得的瞬时奖惩反馈，但同时类似与动态规划方法采用自举方法估计状态的值函数。然后通过多次迭代学习，去逼近真实的状态值函数。

时序差分学习 TD

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



时序差分学习 Temporal-Difference

Simple every-visit Monte Carlo method :

$$V(s_t) \leftarrow V(s_t) + \alpha\left[R_t - V(s_t)\right]$$

$\uparrow$

target: the actual return after time *t*

The simplest TD method, TD(0) :

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

target: an estimate of the return

时序差分学习 (TD)

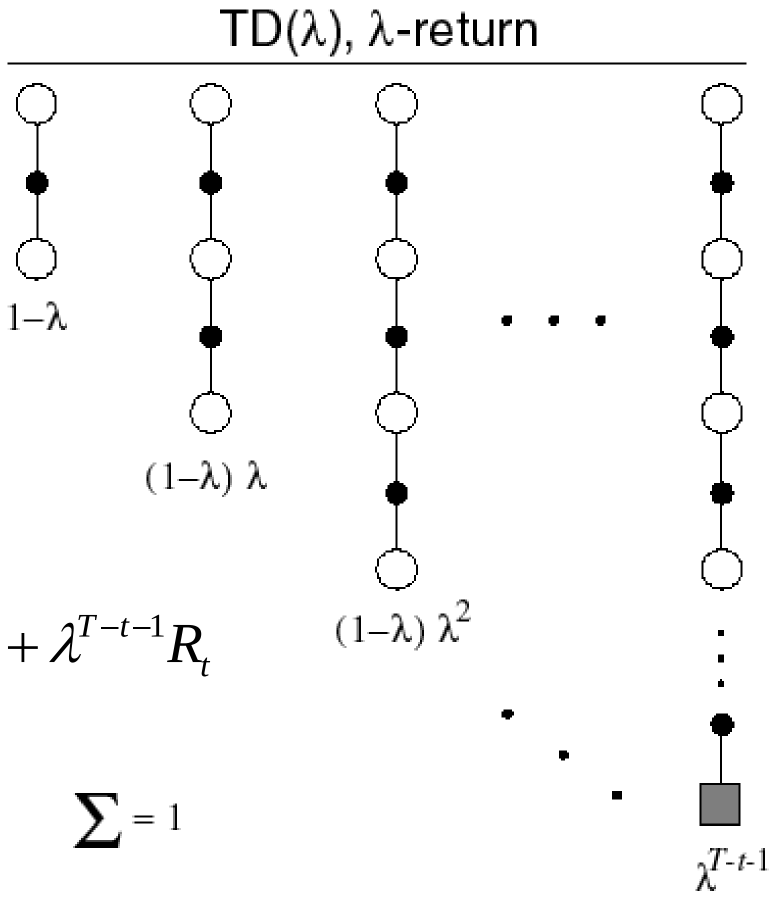
Idea: Do ADP backups on a per move basis, not for the whole state space.

$$U(i) \leftarrow U(i) + \alpha[R(i) + U(j) - U(i)]$$

Theorem: Average value of U(i) converges to the correct value.

Theorem: If α is appropriately decreased as a function of times a state is visited (α=α[N[i]]), then U(i) itself converges to the correct value

TD(λ) – A Forward View



- TD(λ) is a method for averaging all n-step backups
  - weight by λ^{n-1} (time since visitation)
  - λ-return:

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

- Backup using λ-return:

$$\Delta V_t(s_t) = \alpha\left[R_t^\lambda - V_t(s_t)\right]$$

时序差分学习算法 TD(λ)

Idea: update from the whole epoch, not just on state transition.

$$U(i) \leftarrow U(i) + \alpha \sum_{m=k}^{\infty} \lambda^{m-k} [R(i_m) + U(i_{m+1}) - U(i_m)]$$

Special cases:

λ=1: Least-mean-square (LMS), Mont Carlo

λ=0: TD

Intermediate choice of λ (between 0 and 1) is best.

Interplay with α …

时序差分学习算法 TD(λ)

算法 10.1 TD(0)学习算法

```
Initialize $V(s)$ arbitrarily, $\pi$ to the policy to be evaluated
Repeat (for each episode)
  Initialize $s$
  Repeat (for each step of episode)
    Choose $a$ from $s$ using policy $\pi$ derived from $V$(e.g., ε-greedy)
    Take action $a$, observer $r$, $s'$
        $V(s) \leftarrow V(s) + \alpha\left[r + \gamma V(s') - V(s)\right]$
        $s \leftarrow s'$
  Until $s$ is terminal
```

时序差分学习算法

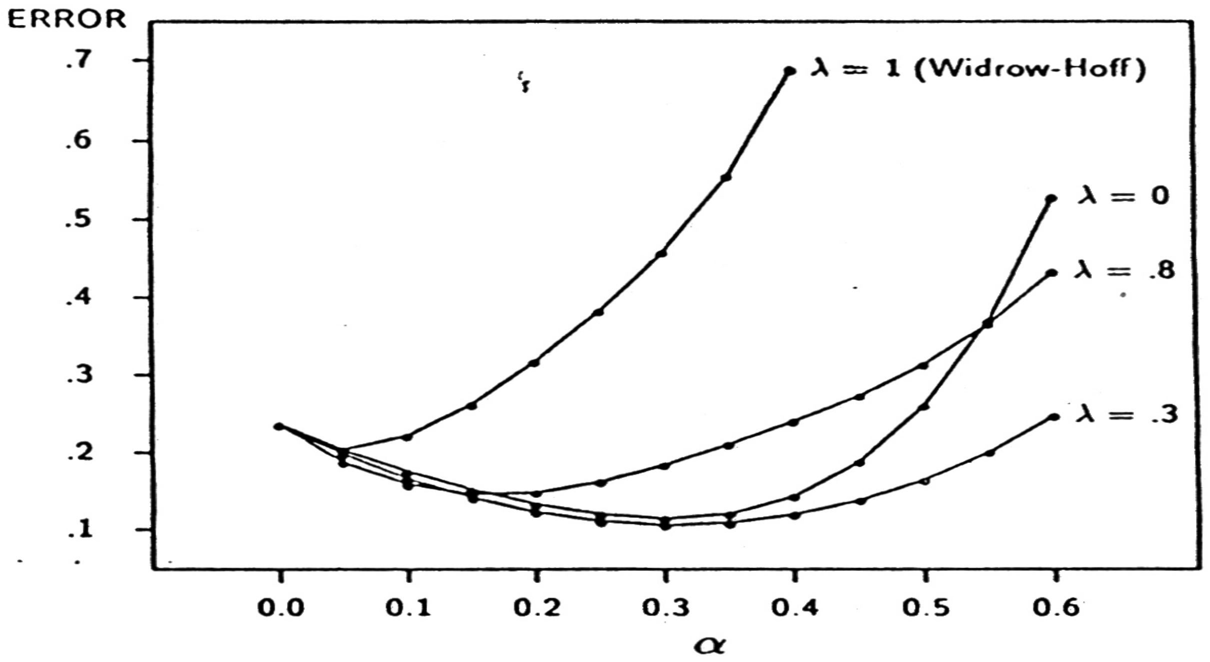


Figure 4. Average error on random walk problem after experiencing 10 sequences. All data are from TD(λ) with different values of α and λ. The dependent measure is the RMS error between the ideal predictions and those found by the learning procedure after a single presentation of a training set. This measure was averaged over 100 training sets. The λ = 1 data points represent performances of the Widrow-Hoff supervised-learning procedure.

时序差分学习算法收敛性TD(λ)

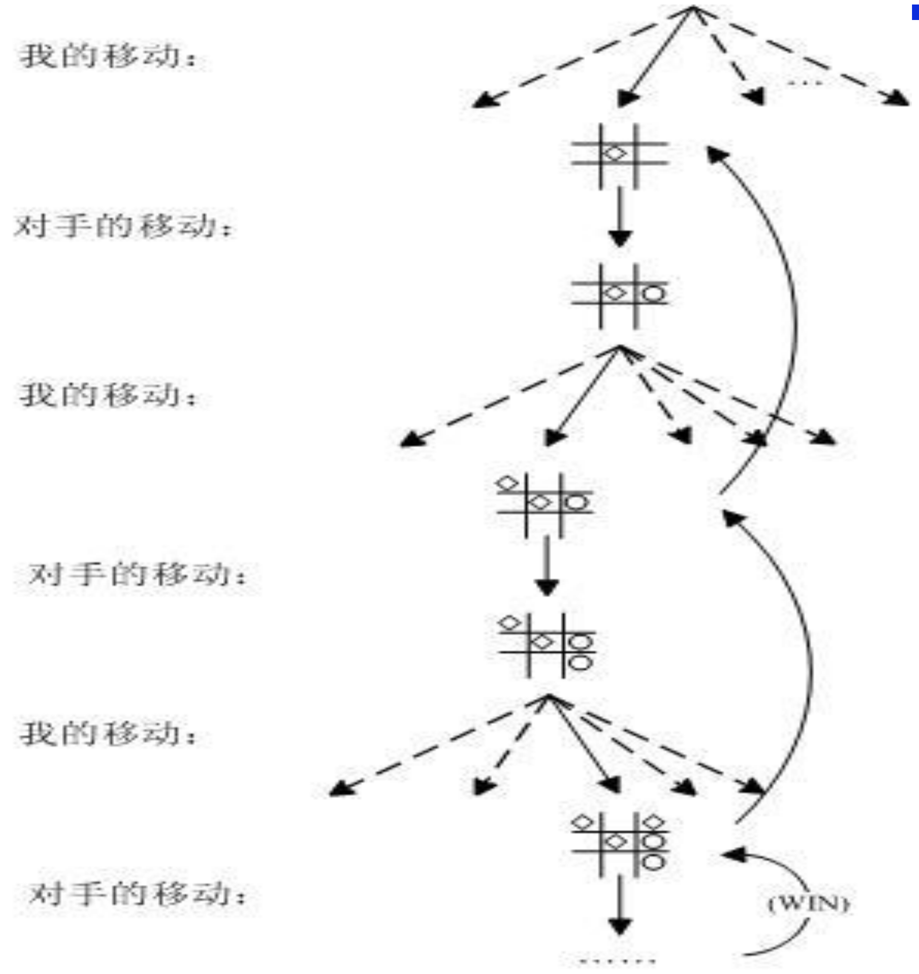
Theorem: Converges w.p. 1 under certain boundaries conditions.
Decrease $\alpha_i(t)$ s.t.

$$\sum_t \alpha_i(t) = \infty$$

$$\sum_t \alpha_i^2(t) < \infty$$

In practice, often a fixed α is used for all i and t.

时序差分学习 TD



Q-learning

Watkins, 1989

在Q学习中，回溯从动作结点开始，最大化下一个状态的所有可能动作和它们的奖励。在完全递归定义的Q学习中，回溯树的底部结点一个从根结点开始的动作和它们的后继动作的奖励的序列可以到达的所有终端结点。联机的Q学习，从可能的动作向前扩展，不需要建立一个完全的世界模型。Q学习还可以脱机执行。我们可以看到，Q学习是一种时序差分的方法。

(10.15)

Q-learning

在Q学习中，*Q*是状态-动作对到学习到的值的一个函数。

对所有的状态和动作：

Q: (state *x* action) → value

对Q学习中的一步：

$$Q(s_t, a_t) \leftarrow (1-c) \times Q(s_t, a_t) + c \times [r_{t+1} + \gamma \underset{a}{\mathrm{MAX}} Q(s_{t+1}, a) - Q(s_t, a_t)]$$

其中*c*和γ都≤1，r_{t+1}是状态s_{t+1}的奖励。

Q-Learning

- Estimate the Q-function using some approximator (for example, linear regression or neural networks or decision trees etc.).
- Derive the estimated policy as an argument of the maximum of the estimated Q-function.
- Allow different parameter vectors at different time points.
- Let us illustrate the algorithm with linear regression as the approximator, and of course, squared error as the appropriate loss function.

Q-learning

Q (a,i)

$$U(i) = \max_a Q(a,i)$$

$$Q(a,i) = R(i) + \sum_j M_{ij}^a \max_{a'} Q(a', j)$$

Direct approach (ADP) would require learning a model M_{ij}^a.

Q-learning does not:

Do this update after each state transition:

$$Q(a,i) \leftarrow Q(a,i) + \alpha[R(i) + \max_{a'} Q(a', j) - Q(a,i)]$$

探查Exploration

在使用(控制)和探查(标识)之间寻求折中

Extremes: greedy vs. random acting
(n-armed bandit models)

Q-learning将收敛到最佳的Q 估值，如果

*(由于探查)，每个状态可以被无限地访问,

*当时间接近无限时，动作选择变得贪婪，和

*学习速率α 被减少足够快但不是太迅速
(我们在TD 学习过程中讨论)

常用探查方法

1. In value iteration in an ADP agent: Optimistic estimate of utility $U^{+}(i)$

$$U^{+}(i) \leftarrow R(i) + \max_{a} f[\sum_{j} M_{ij}^{a} U^{+}(j), N(a,i)]$$

Exploration func

$$f(u,n) = \begin{cases} R^{+} \text{ if } n<N \\ u \text{ o.w.} \end{cases}$$

2. Є-greedy method

   Nongreedy actions $\frac{\varepsilon}{|A(s)|}$

   Greedy action $1-\varepsilon+\frac{\varepsilon}{|A(s)|}$

3. Boltzmann exploration

$$P_{a*} = \frac{e^{-\frac{\sum_{j} M_{ij}^{a^*} U(j)}{T}}}{\sum_{a} e^{-\frac{\sum_{j} M_{ij}^{a} U(j)}{T}}}$$

Q-Learning 算法

Q学习算法

- Initialize *Q*(*s*,*a*) arbitrarily
- Repeat (for each episode)
-   Initialize *s*
-   Repeat (for each step of episode)
-   Choose *a* from *s* using policy derived from Q (e.g., ε-greedy)
- Take action *a*, observer *r*, *s′*

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

$$s \leftarrow s'$$

- Until *s* is terminal

Q-Learning 算法

- Set $Q_{T+1} = 0.$
- For

$$t = T, T-1, \ldots, 0,$$

$$Y_t \leftarrow r_t + \max_{a_{t+1}} Q_{t+1}(\mathbf{S}_{t+1}, \mathbf{A}_t, a_{t+1}; \hat{\theta}_{t+1}).$$

$$\hat{\theta}_t \leftarrow \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left(Y_{t,i} - Q_t(\mathbf{S}_{t,i}, \mathbf{A}_{t,i}; \theta) \right)^2.$$

- The estimated policy satisfies

$$\hat{\pi}_{Q,t}(\mathbf{s}_t, \mathbf{a}_{t-1}) \in \arg\max_{a_t} Q_t(\mathbf{s}_t, \mathbf{a}_t; \hat{\theta}_t), \forall t.$$

直觉是什么?

- Bellman equation gives

$$Q_t^*(\boldsymbol{S}_t, \boldsymbol{A}_t) = E\left[r_t + \max_{a_{t+1}} Q_{t+1}^*(\mathbf{S}_{t+1}, \mathbf{A}_t, a_{t+1}) \mid \mathbf{S}_t, \mathbf{A}_t \right]$$

- If $Q_{t+1} = Q_{t+1}^*$ and the training set were infinite, then Q-learning minimizes

$$E\left[r_t + \max_{a_{t+1}} Q_{t+1}^*(\mathbf{S}_{t+1}, \mathbf{A}_t, a_{t+1}) - Q_t(\mathbf{S}_t, \mathbf{A}_t; \theta) \right]^2$$

which is equivalent to minimizing

$$E[Q_t^* - Q_t(\mathbf{S}_t, \mathbf{A}_t; \theta)]^2$$

A-Learning

Murphy, 2003 and Robins, 2004

- Estimate the A-function (advantages) using some approximator, as in Q-learning.
- Derive the estimated policy as an argument of the maximum of the estimated A-function.
- Allow different parameter vectors at different time points.
- Let us illustrate the algorithm with linear regression as the approximator, and of course, squared error as the appropriate loss function.

A-Learning Algorithm
(Inefficient Version)

- For $t = T, T-1, \dots, 0,$

$$Y_t \leftarrow \sum_{j=t}^{T} r_j - \sum_{j=t+1}^{T} \mu_j(\mathbf{S}_j, \mathbf{A}_j; \hat{\theta}_j).$$

$$\hat{\theta}_t \leftarrow \arg\min_{\theta} \frac{1}{n} \sum_{i=1}^{n} \left[Y_{t,i} - \mu_t(\mathbf{S}_{t,i}, \mathbf{A}_{t,i}; \theta) + E(\mu_t \mid \mathbf{S}_{t,i}, \mathbf{A}_{t-1,i}) \right]^2.$$

$$\mu_t(\mathbf{S}_t, \mathbf{A}_t; \hat{\theta}_t) \leftarrow \mu_t(\mathbf{S}_t, \mathbf{A}_t; \hat{\theta}_t) - \max_{a_t} \mu_t(\mathbf{S}_{t,}, \mathbf{A}_{t-1}, a_t; \hat{\theta}_t)$$

- The estimated policy satisfies

$$\hat{\pi}_{A,t}(\mathbf{s}_t, \mathbf{a}_{t-1}) \in \arg\max_{a_t} \mu_t(\mathbf{s}_t, \mathbf{a}_t; \hat{\theta}_t), \forall t.$$

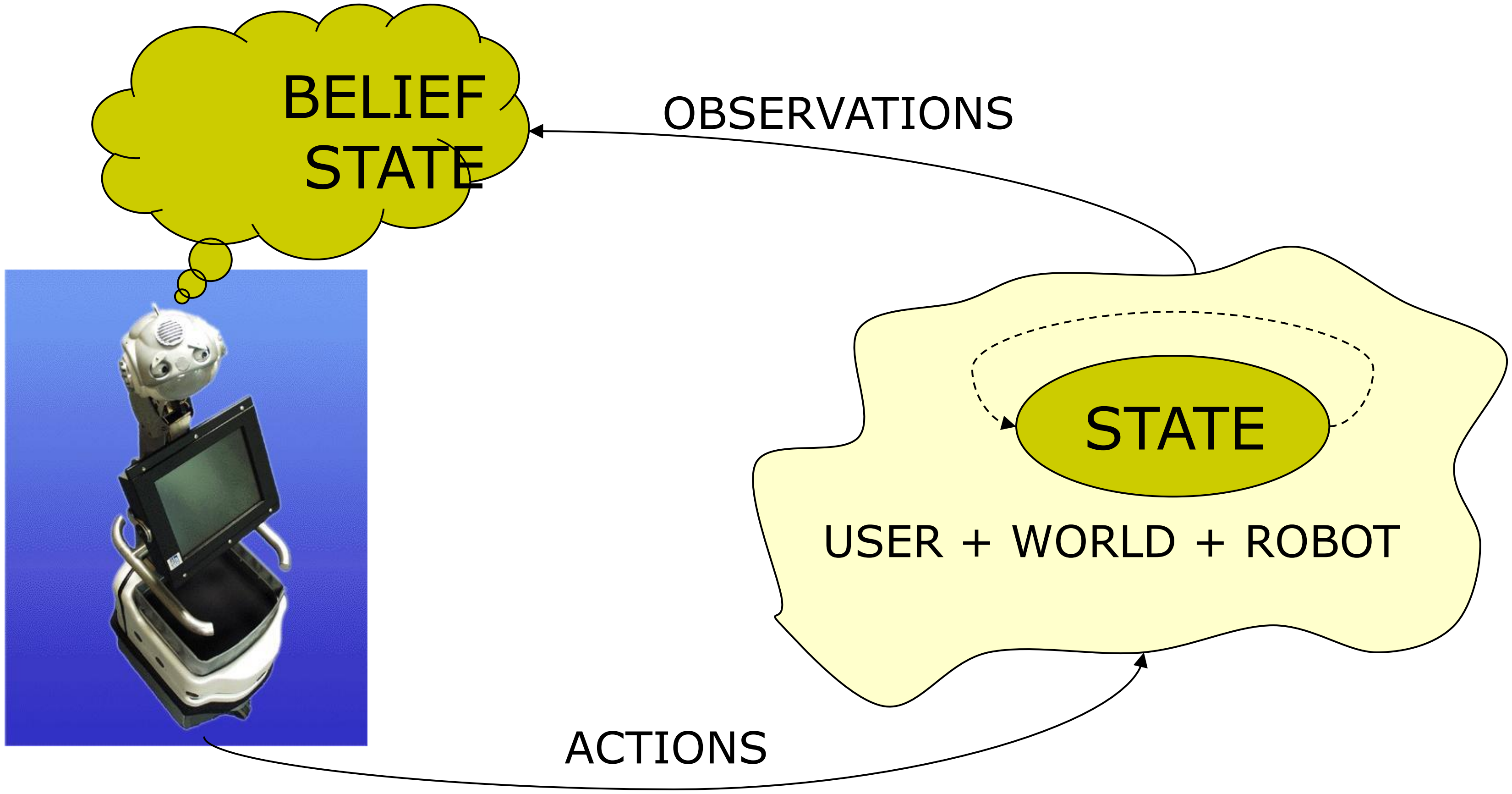
Q-Learning and A-learning差别

- Q-learning
  - At time t we model the main effects of the history, $(\mathbf{S}_{t,},\mathbf{A}_{t\text{-}1})$ and the action $\mathbf{A}_t$ and their interaction
  - Our $Y_{t\text{-}1}$ is affected by how we modeled the main effect of the history in time t, $(\mathbf{S}_{t,},\mathbf{A}_{t\text{-}1})$
- A-learning
  - At time t we only model the effects of $\mathbf{A}_t$ and its interaction with $(\mathbf{S}_{t,},\mathbf{A}_{t\text{-}1})$
  - Our $Y_{t\text{-}1}$ does not depend on a model of the main effect of the history in time t, $(\mathbf{S}_{t,},\mathbf{A}_{t\text{-}1})$

Q-Learning Vs. A-Learning

- 直到现在有关的优点和缺点没被完全知道。
- Q-learning有低的分散，高的偏置。
- A-learning有高的分散，低的偏置。
- 比较Q-learning与A-learning必须在分散与偏置间求折中。

High-level robot behavior control using Partially Observable Markov Decision Processes



POMDP部分感知马氏决策过程

- Rather than observing the state we observe some function of the state.
- Ob – Observable function

  a random variable for each states.
- Problem : different states may look similar

The optimal strategy might need to consider the history.

Framework of POMDP

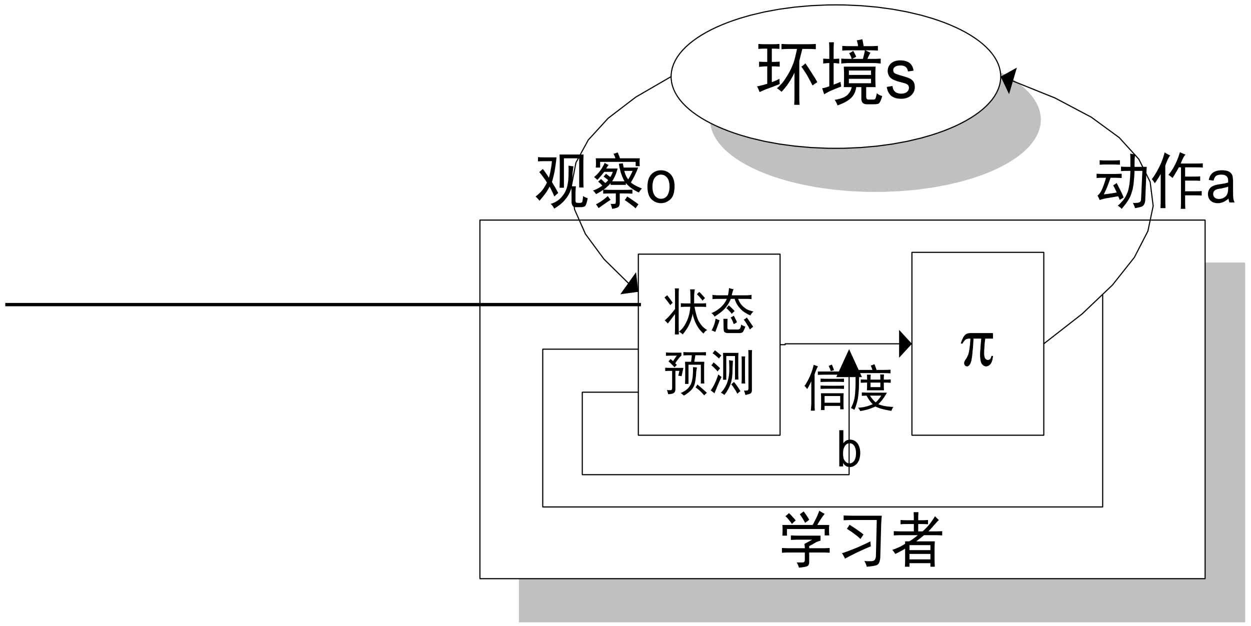
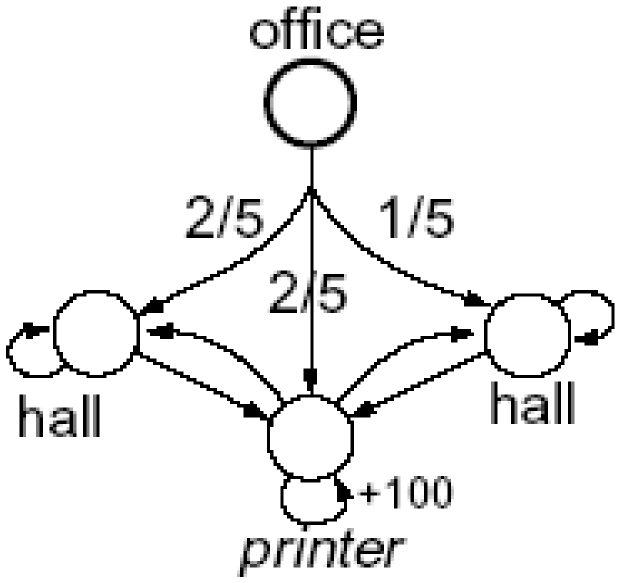


图1 POMDP 问题框架

POMDP由六元组<S, A, R, P, Ω, O>定义。其中<S, A, P, R>定义了环境潜在的马尔可夫决策模型上，Ω是观察的集合，即系统可以感知的世界状态集合，观察函数O：$S\times A\rightarrow \mathrm{PD}$（$\Omega$）。系统在采取动作$a$转移到状态$s'$时，观察函数$O$确定其在可能观察上的概率分布。记为$O$（$s', a, o$）。[1] Ω可以是S的子集，也可以与S无关

POMDPs

What if state information (from sensors) is noisy? Mostly the case!

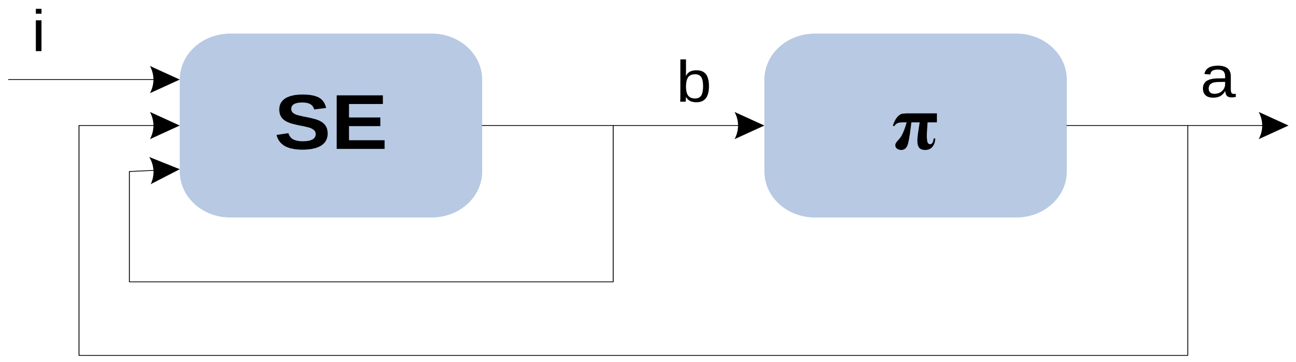


An example of a partially observable environment.

MDP techniques are suboptimal!

Two halls are not the same.

POMDPs – A Solution Strategy



SE: Belief State Estimator (Can be based on HMM)

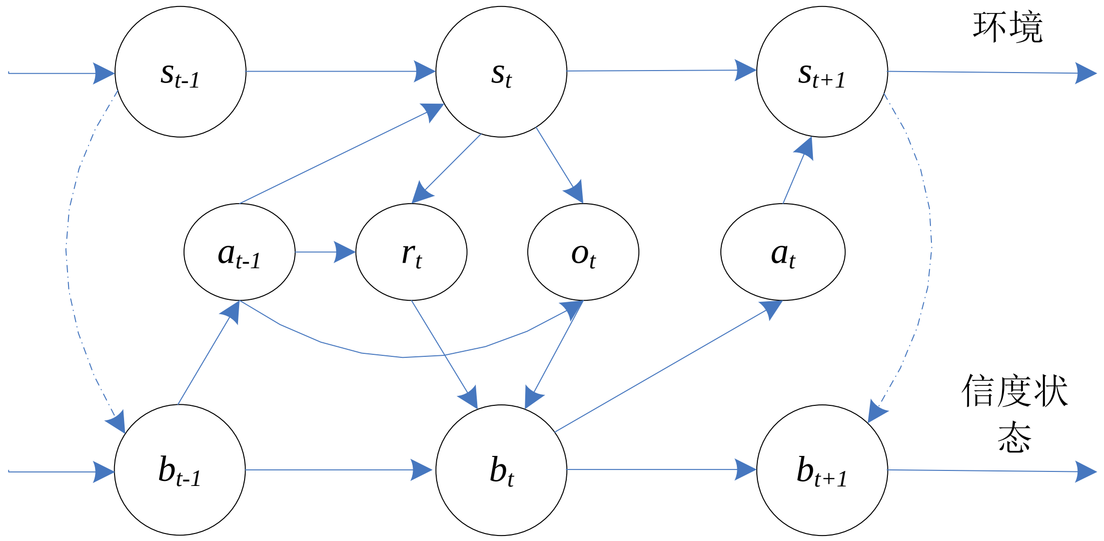
Π: MDP Techniques

POMDP_信度状态方法

- Idea: Given a history of actions and observable value, we compute a posterior distribution for the state we are in (belief state)
- The belief-state MDP
  - States: distribution over S (states of the POMDP)
  - Actions: as in POMDP
  - Transition: the posterior distribution (given the observation)

Open Problem : How to deal with the continuous distribution?

The Learning Process of Belief MDP



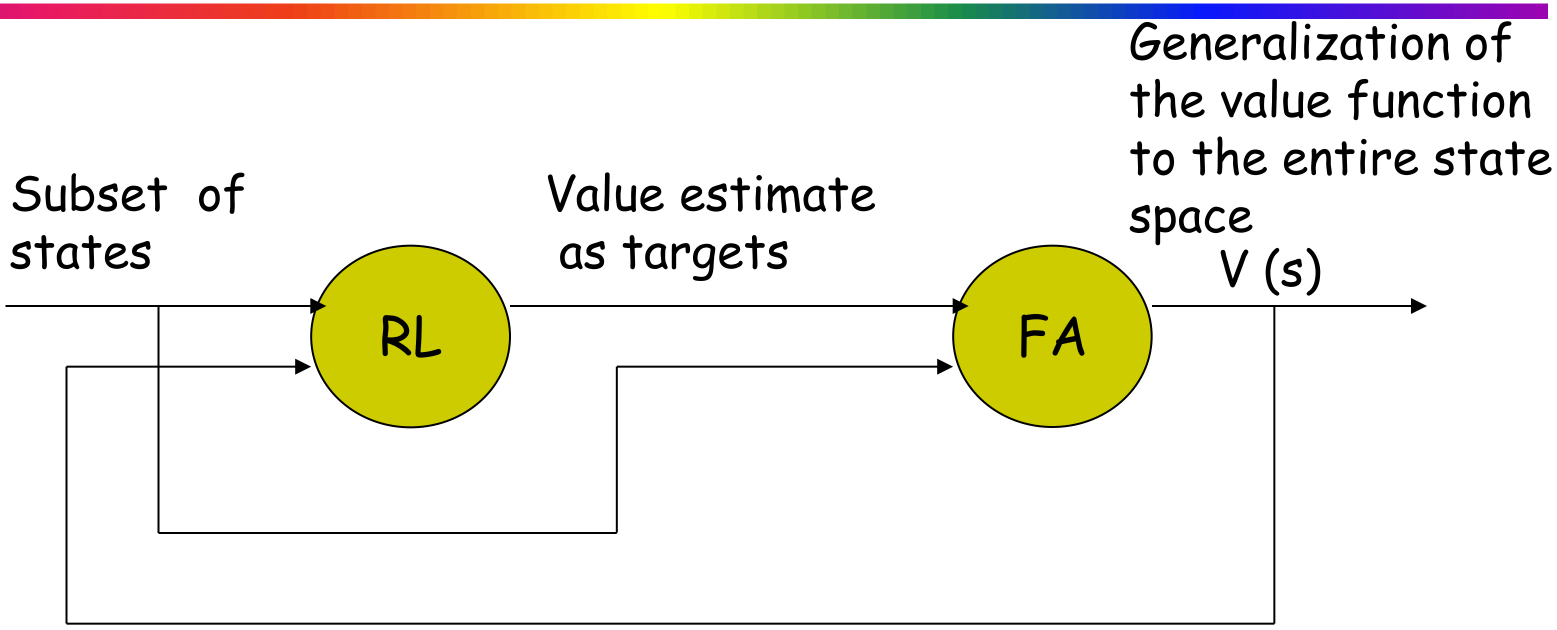
$$b'(s') = \Pr(s' \mid o,a,b) = \frac{O(s',a,o)\sum_{s\in S} P(s,a,s')b(s)}{\Pr(o \mid a,b)}$$

$$\rho(b,a) = \sum_{s\in S} b(s)R(s,a) \qquad \tau(b,a,b') = \sum_{o\in O} \Pr(b' \mid a,b,o)\Pr(o \mid a,b)$$

Major Methods to Solve POMDP

	算法名称	基本思想
学习值函数	**Memoryless policies**	直接采用标准的强化学习算法
	Simple memory based approaches	使用***k***个历史观察表示当前状态
	UDM(Utile Distinction Memory)	分解状态，构建有限状态机模型
	NSM(Nearest Sequence Memory)	存储状态历史，进行距离度量
	USM(Utile Suffix Memory)	综合**UDM**和**NSM**两种方法
	Recurrent-Q	使用循环神经网络进行状态预测
策略搜索	**Evolutionary algorithms**	使用遗传算法直接进行策略搜索
	Gradient ascent method	使用梯度下降（上升）法搜索

强化学习中的函数估计



Generalization of the value function to the entire state space
Subset of states
Value estimate as targets
V (s)
RL
FA

$$V_0, M(V_0), \Gamma(M(V_0)), M(\Gamma(M(V_0))), \Gamma(M(\Gamma(M(V_0)))), \ldots$$

Γ is the TD operator.

M is the function approximation operator.

并行两个迭代过程

- 值函数迭代过程

$$Q(s,a) \leftarrow (1-\alpha)V'(s,a) + \alpha\left(r(s,a,s') + \max_{a'} V'(s',a')\right)$$

- 值函数逼近过程

$$V'(s,a) = M\left(Q(s,a)\right)$$

How to construct the *M* function? Using state cluster, interpolation, decision tree or neural network?

并行两个迭代过程

- Function Approximator:

$V(s) = f(s, w)$

weight vector

- *Update: Gradient-descent Sarsa:*

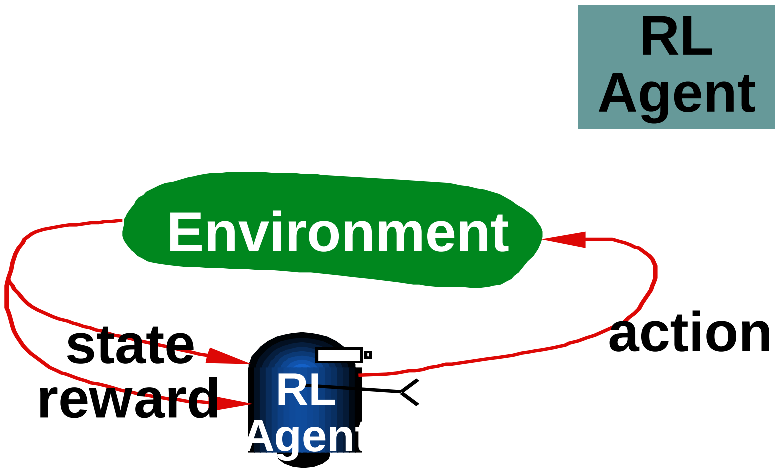
Standard gradient

$$w \leftarrow w + \alpha\,[\underbrace{r_{t+1} + \gamma Q(s_{t+1},a_{t+1})}_{\text{target value}} - Q(s_t,a_t)]\,\nabla_w f(s_t,a_t,w)$$

estimated value

Open Problem : How to design the non-liner FA system which can converge with the incremental instances?

Multi-agent MDP



- Distributed RL
- Markov Game
- Best Response

三种观点

	问题空间	主要方法	算法准则
合作多agent强化学习	分布、同构、合作环境	交换状态	提高学习收敛速度
		交换经验	
		交换策略	
		交换建议	
基于平衡解多agent强化学习	同构或异构、合作或竞争环境	极小极大-Q	理性和收敛性
		NASH-Q	
		CE-Q	
		WoLF	
最佳响应多agent强化学习	异构、竞争环境	PHC	收敛性和不遗憾性
		IGA	
		GIGA	
		GIGA-WoLF	

马尔可夫对策

- 在n个agent的系统中，定义离散的状态集S（即对策集合G），agent动作集A_i的集合A, 联合奖赏函数R_i：$S\times A_1\times\ldots\times A_n\rightarrow\mathcal{R}$和状态转移函数$P$：$S\times A_1\times\ldots\times A_n\rightarrow PD(S)$。

		Agent A		
		剪刀	石头	布
Agent B	剪刀	(0,0)	(-1,+1)	(+1,-1)
	石头	(+1,-1)	(0,0)	(-1,+1)
	布	(-1,+1)	(+1,-1)	(0,0)

		Agent A	
		对抗	合作
Agent B	对抗	(-9,-9)	(0,-10)
	合作	(-10,0)	(-1,-1)

基于平衡解方法的强化学习

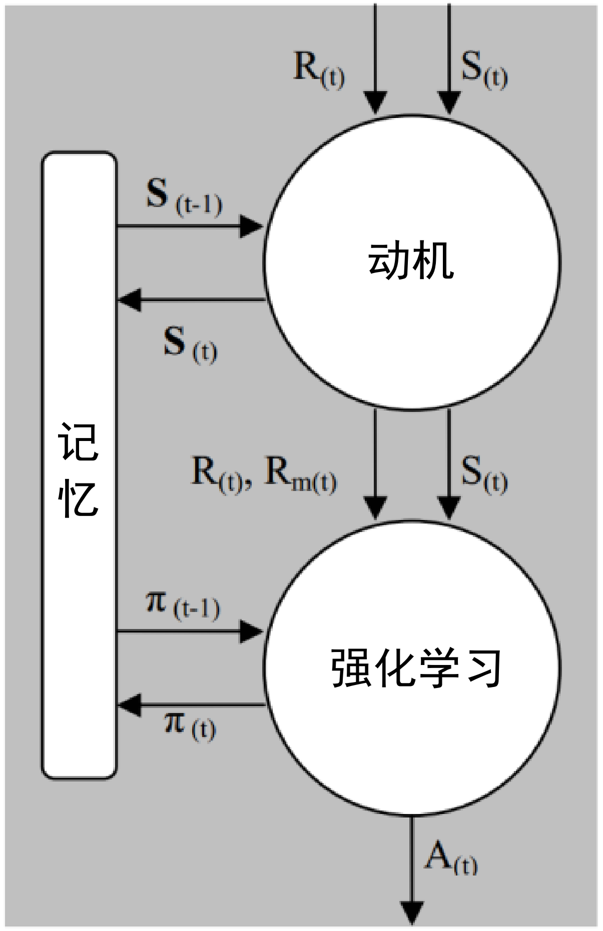
$$V^*(s) = \max_{\vec{a} \in \vec{A}(s)} \sum_{s'} P_{ss'}^{\vec{a}} \left[R_{ss'}^{\vec{a}} + \gamma V^*(s') \right]$$

The optimal policy in single game is Nash equilibrium.

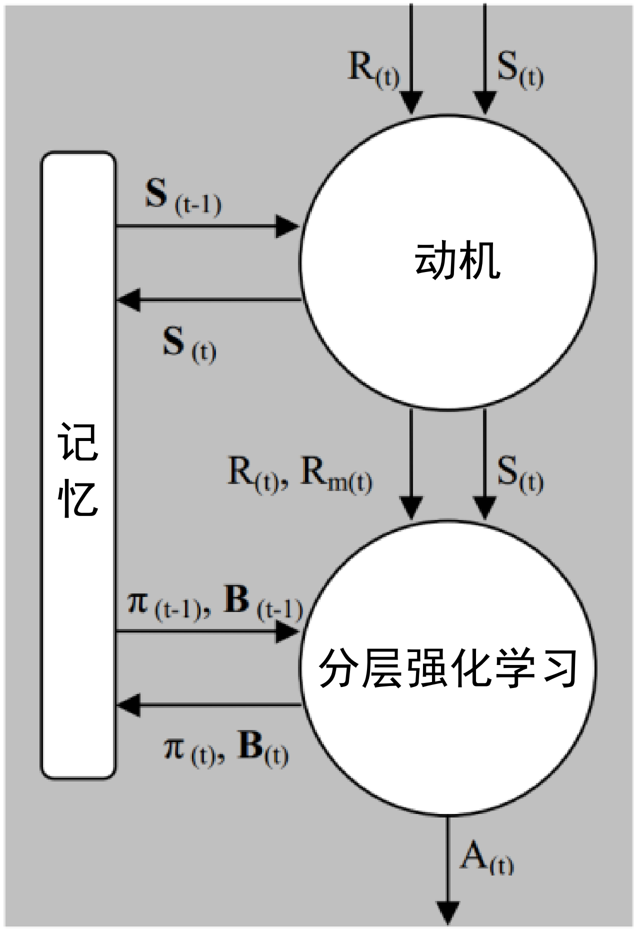
Open Problem : Nash equilibrium or other equilibrium is enough?

基于动机的强化学习

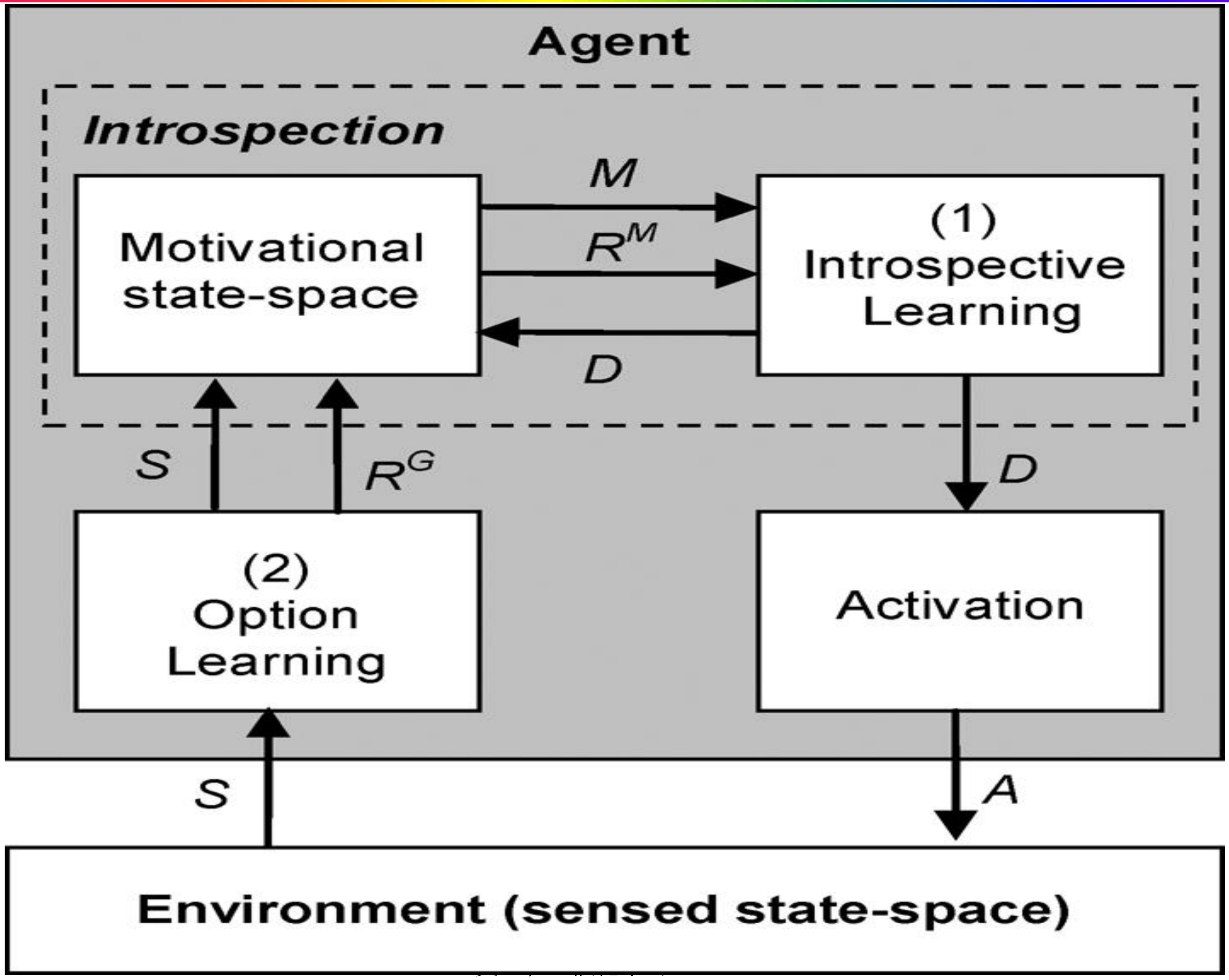
(a)



(b)

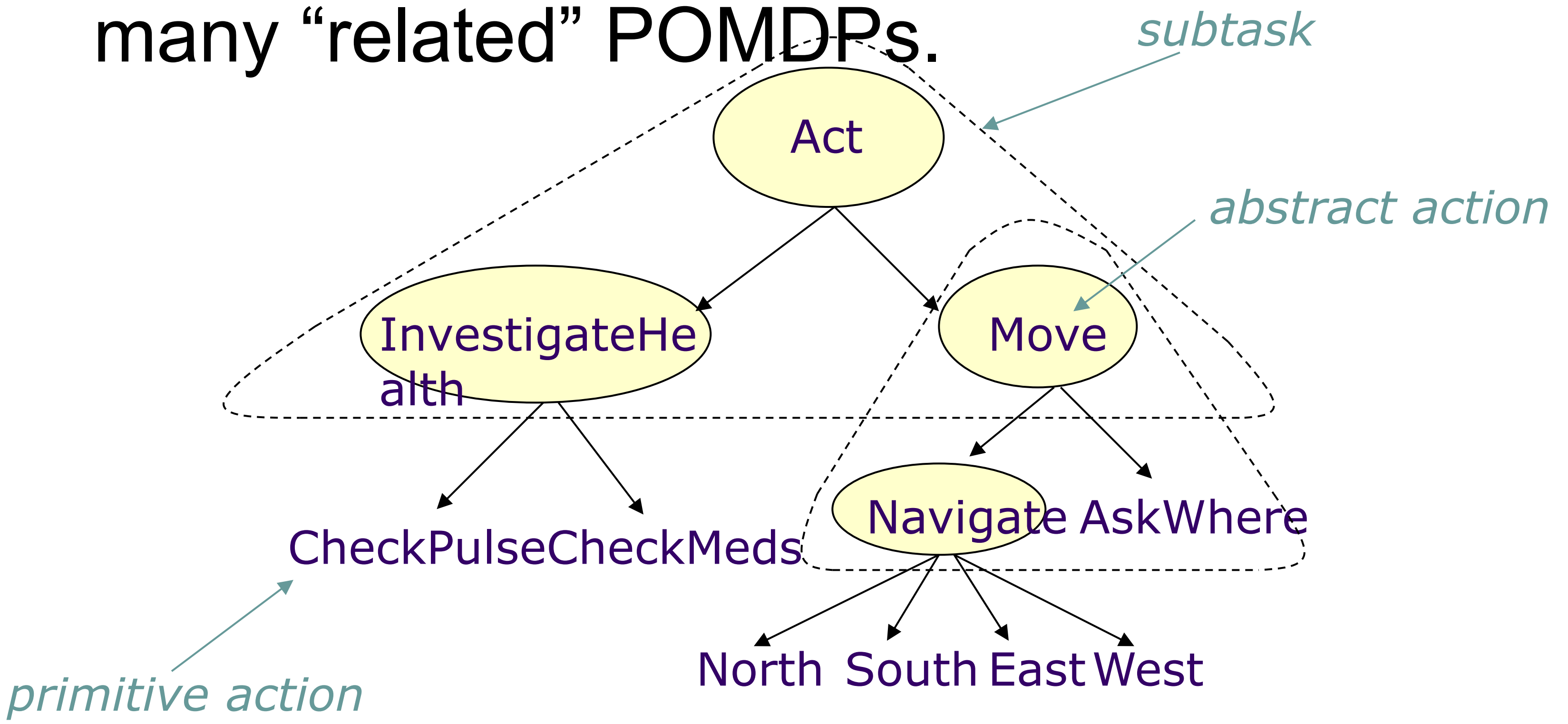


基于动机的强化学习


Agent
Introspection
Motivational state-space
M
R^M
D
(1) Introspective Learning
S
R^G
D
(2) Option Learning
Activation
S
A
Environment (sensed state-space)

Hierarchical POMDPs

- **Key Idea:** Exploit hierarchical structure in the problem domain to break a problem into many “related” POMDPs.



分层POMDPs规划

- Given POMDP model $M = \{ S, A, \Omega, b, T, O, R \}$ and hierarchy H
- For each subtask $h \in H$:

  1) Set components

  $A_h \Leftarrow$ children nodes

  $S_h \Leftarrow S$

  $\Omega_h \Leftarrow \Omega$

  b_h, T_h, O_h, R_h

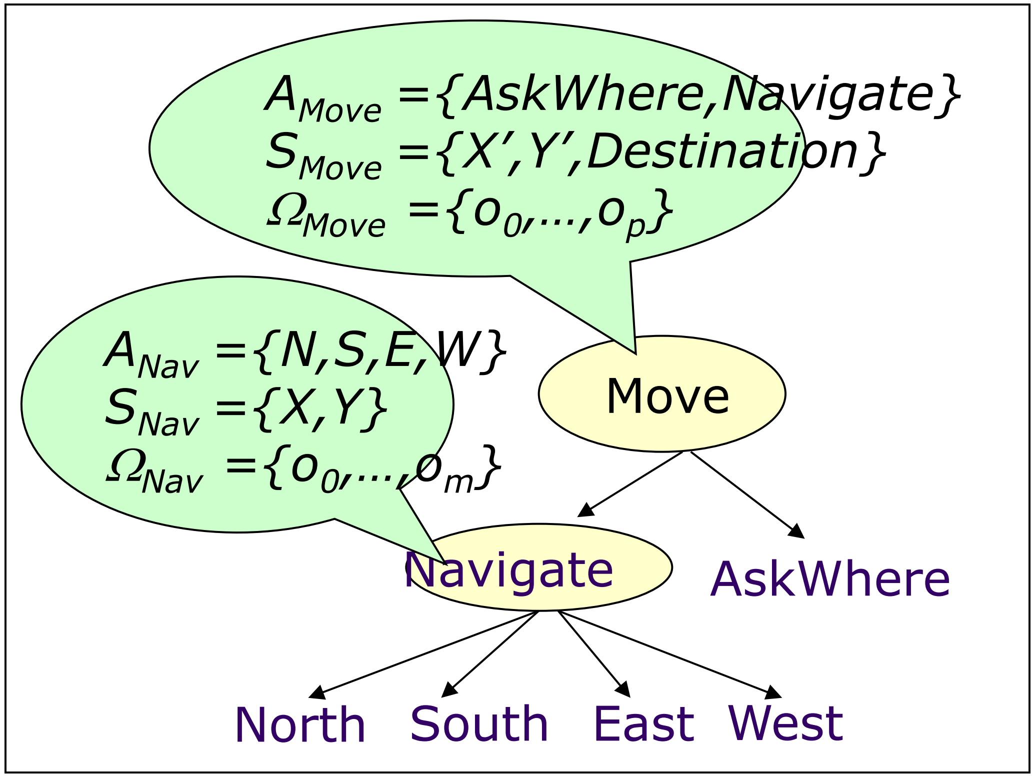
  2) Minimize model

  $S_h \Leftarrow \{z_h(s_0), \ldots, z_h(s_n)\}$

  $\Omega_h \Leftarrow \{y_h(o_0), \ldots, y_h(o_p)\}$

  3) Solve subtask h

  $\pi_h \Leftarrow \{b_h, T_h, O_h, R_h\}$


$A_{Move} = \{AskWhere, Navigate\}$
$S_{Move} = \{X', Y', Destination\}$
$\Omega_{Move} = \{o_0, \ldots, o_p\}$
$A_{Nav} = \{N, S, E, W\}$
$S_{Nav} = \{X, Y\}$
$\Omega_{Nav} = \{o_0, \ldots, o_m\}$
Move
Navigate
AskWhere
North
South
East
West

分层POMDPs执行

- Step 1 - Update belief:

$$b_t(s_i) = \frac{P(o \mid s_i, a) \sum_{s_j \in S} P(s_i \mid s_j, a) b_{t-1}(s_j)}{P(o \mid a, b)}$$

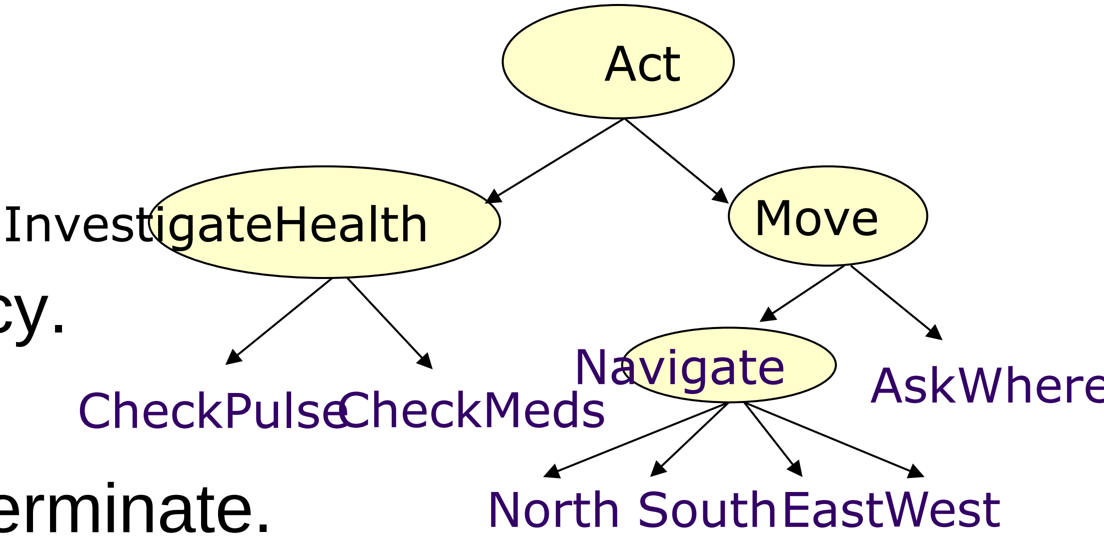
- Step 2 - Traversing hierarchy top-down, for each subtask:

  1) Get local belief.

  2) Consult local policy.

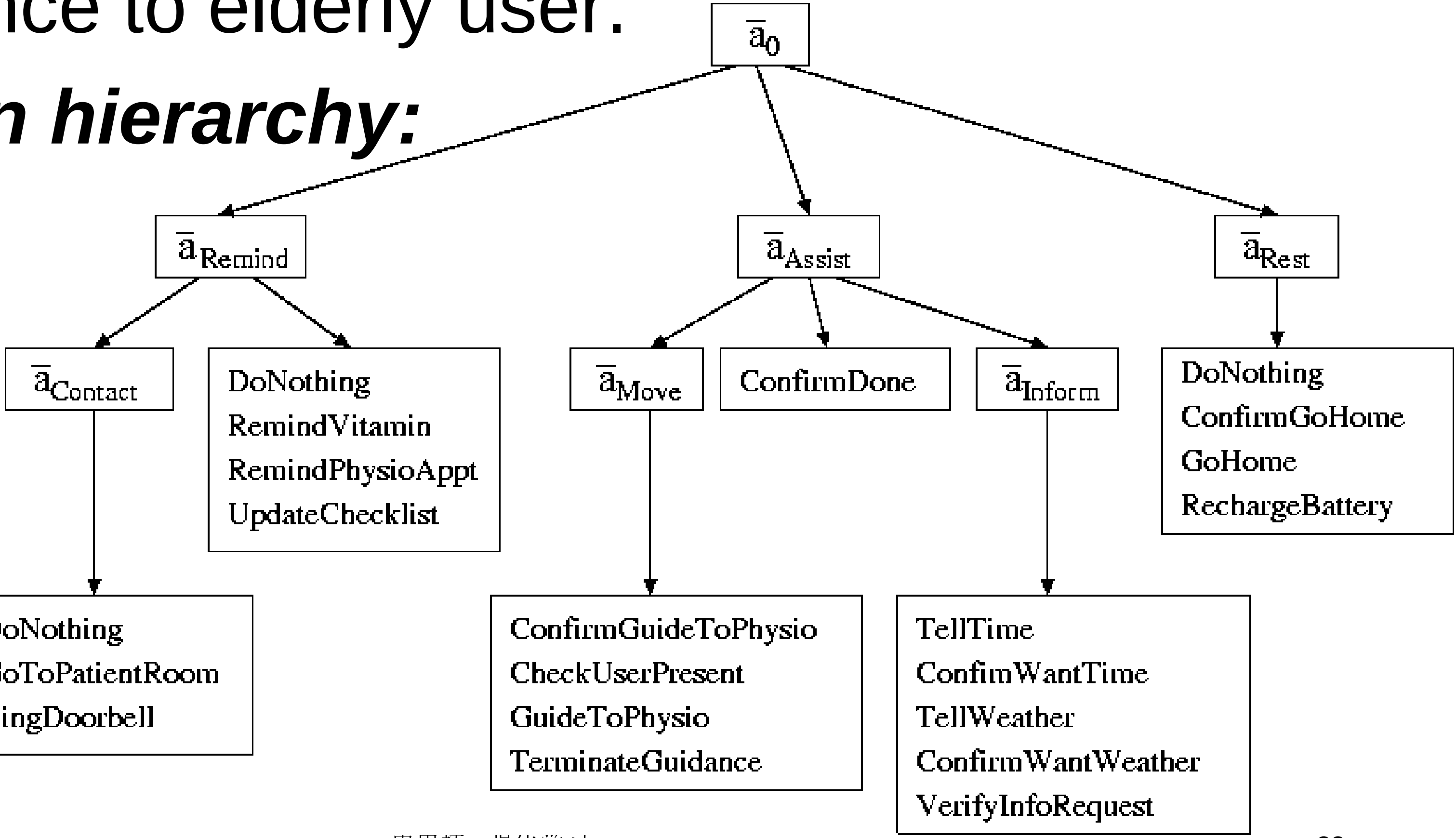
  3) If *a* is leaf node, terminate.

  Else, go to that subtask.



实验安排

- ***Task***: Robot provides reminders and guidance to elderly user.
- ***Action hierarchy:***



Applications of RL

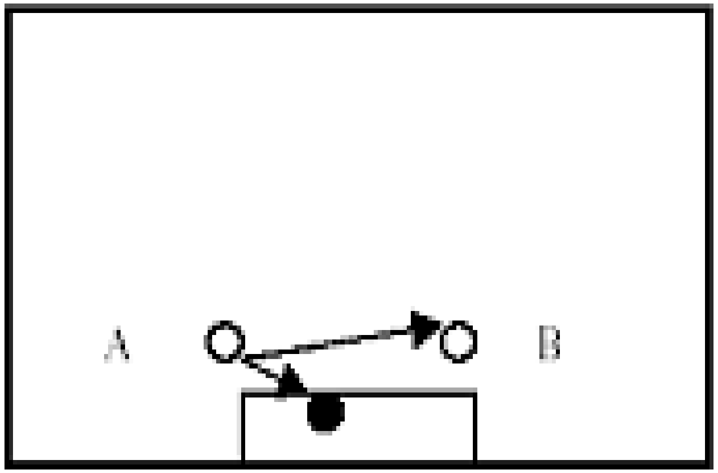
- Checker's [Samuel 59]
- TD-Gammon [Tesauro 92]
- World's best downpeak elevator dispatcher [Crites at al ~95]
- Inventory management [Bertsekas et al ~95]
  - 10-15% better than industry standard
- Dynamic channel assignment [Singh & Bertsekas, Nie&Haykin ~95]
  - Outperforms best heuristics in the literature
- Cart-pole [Michie&Chambers 68-] with bang-bang control
- Robotic manipulation [Grupen et al. 93-]
- Path planning
- Robot docking [Lin 93]
- Parking
- Football [Stone98]
- Tetris
- Multiagent RL [Tan 93, Sandholm&Crites 95, Sen 94-, Carmel&Markovitch 95-, lots of work since]
- Combinatorial optimization: maintenance & repair
  - Control of reasoning [Zhang & Dietterich IJCAI-95]

仿真机器人足球

应用Q学习算法进行仿真机器人足球2 对1 训练，训练的目的是试图使主体学习获得到一种战略上的意识，能够在进攻中进行配合[宋志伟, 2003]

仿真机器人足球

前锋A控球，并且在可射门的区域内，但是A已经没有射门角度了；队友B也处于射门区域，并且B具有良好的射门角度。A传球给B，射门由B来完成，那么这次进攻配合就会很成功。通过Q学习的方法来进行2对1的射门训练，让A掌握在这种状态情况下传球给B的动作是最优的策略；主体通过大量的学习训练（大数量级的状态量和重复相同状态）来获得策略，因此更具有适应性 。



仿真机器人足球

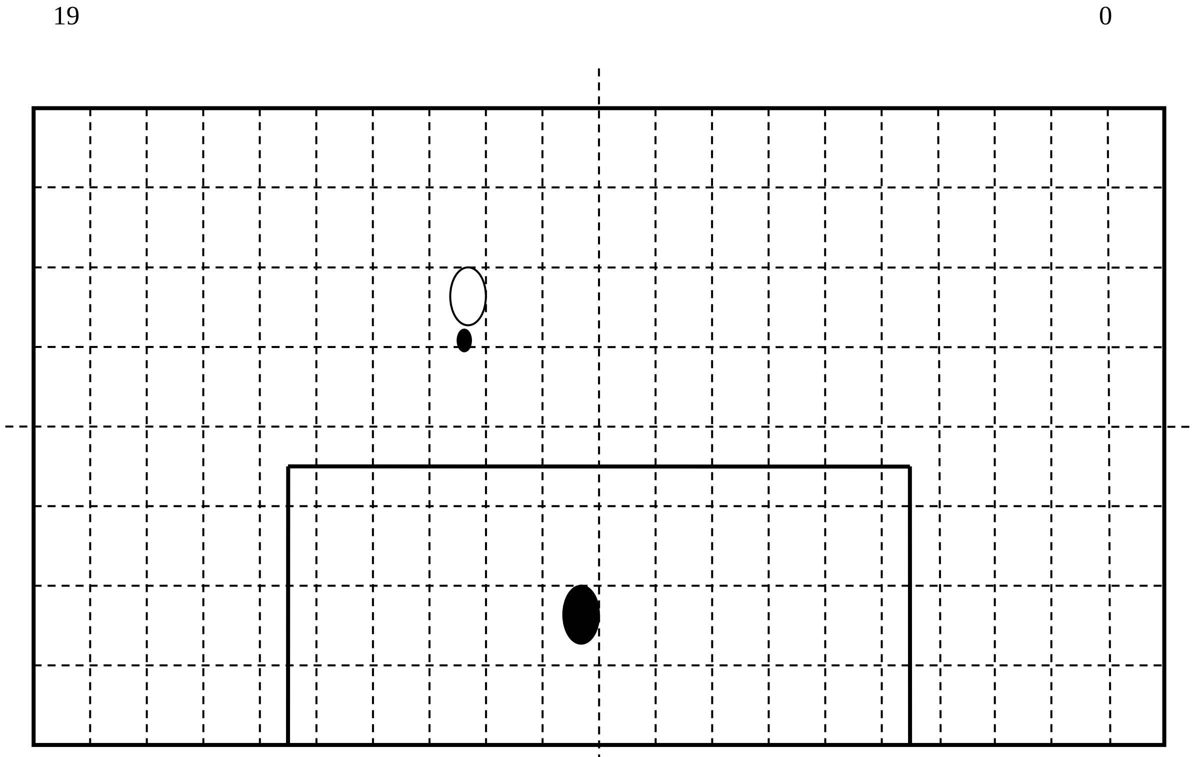


图10.11　进攻禁区内的位置划分

仿真机器人足球

状态描述，将进攻禁区划分为个小区域，每个小区域是边长为2m的正方形，一个二维数组()便可描述这个区域。使用三个Agent的位置来描述2对 1进攻时的环境状态，利用图10.11所示的划分来泛化状态。可认为主体位于同一战略区域为相似状态，这样对状态的描述虽然不精确，但设计所需的是一种战略层次的描述，可认为Agent在战略区域内是积极跑动的，这种方法满足了需求。如此，便描述了一个特定的状态；其中，是进攻队员A的区域编号，是进攻队员B的区域编号，是守门员的区域编号。区域编号计算公式为：S=ix8+j。相应地，所保存的状态值为三个区域编号组成的对。前锋A控球，并且在可射门的区域内，但是A已经没有射门角度了；队友B也处于射门区域，并且B具有良好的射门角度。A传球给B，射门由B来完成，那么这次进攻配合就会很成功。通过Q学习的方法来进行2 对1的射门训练，让A掌握在这种状态情况下传球给B的动作是最优的策略；主体通过大量的学习训练（大数量级的状态量和重复相同状态）来获得策略，因此更具有适应性 。

仿真机器人足球

- 可选动作集确定为 $\{Shoot,Pass,Dribble\}$
- Shoot 的策略通过基于概率的射门训练的学习来得到。
- Dribble 的策略是，始终向受到威胁小，并且射门成功率高的区域带球。为了实现这一策略目标，可划分进攻区域为多个战略区，在每个战略区进行射门评价，记录每个区域的射门成功率。
- Pass策略很简单，只需在两个Agent间进行传球，即不需要选择球传送的对象，也不需要判断传球路径。如果传球失败的话，则认为在这种状态下执行Pass策略是不成功的；经过此训练后，不可能的传球路径也不会被执行了。

仿真机器人足球

- 训练中的所有状态包含了四个吸收状态。假设进攻方在左半场，按照标准的Soccer server规范，这四个状态的比赛模式为play_on、goal_left、goal_kick_right和free_kick_right。当达到吸收状态时，给与主体最终奖励r。促使到达吸收状态的上一步动作获得的立即回报值为最终奖励值r，其他未直接促使到达吸收状态的动作均获得过程奖励值作为立即奖励；其中goal_left的r最大为1表示进球，其他状态下r为不同大小的负数。

仿真机器人足球

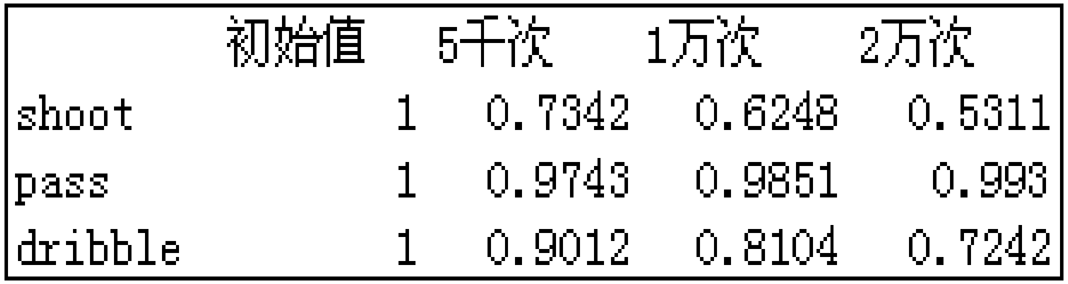
- 主体在经过一定的状态和执行多个动作后获得了终态奖励（到达了吸收状态），这时就会对这个状态-动作序列分配奖励。Q学习算法的核心就是每一个状态和动作的组合都拥有一个Q值，每次获得最终回报后通过更新等式更新这个Q值。由于Robocup仿真平台在设计的时候在状态转换的时候加入了一个较小的随机噪音，所以该模型为非确定MDP, 确定Q更新等式为：

$$Q(s,a)=(1-\alpha)Q(s,a)+\alpha\left(r+\gamma\max Q(s_{t+1},a_{t+1})\right)$$

规定=0.1，=0.95。

仿真机器人足球

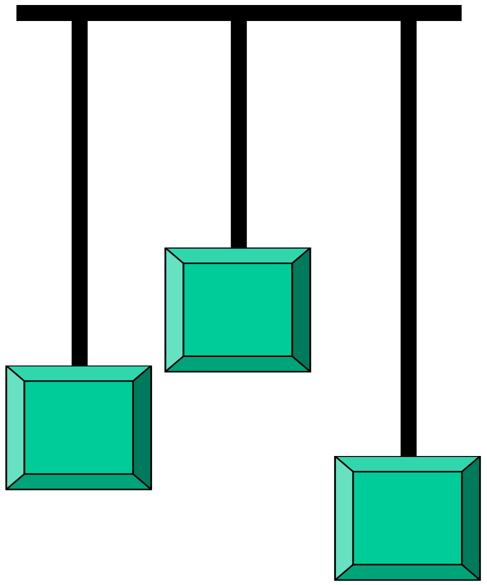
在实际的训练中，初始Q表各项的值为1。经过大约2万次的训练（到达吸收状态为一次），Agent的Q表中的绝大部分项发生了变化，并且已经区分开来。下表是某场景下的训练次数和动作选择的变化示意表。



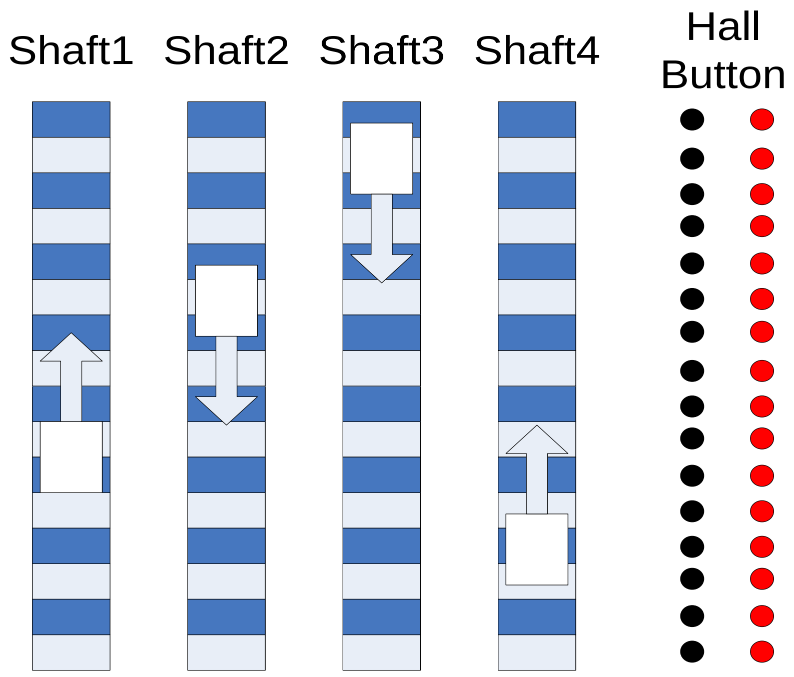
	初始值	5千次	1万次	2万次
shoot	1	0.7342	0.6248	0.5311
pass	1	0.9743	0.9851	0.993
dribble	1	0.9012	0.8104	0.7242

强化学习应用_调度

- Job shop scheduling
- Group-control elevator scheduling
- Dynamical channel allocation



应用 - 分布式电梯控制

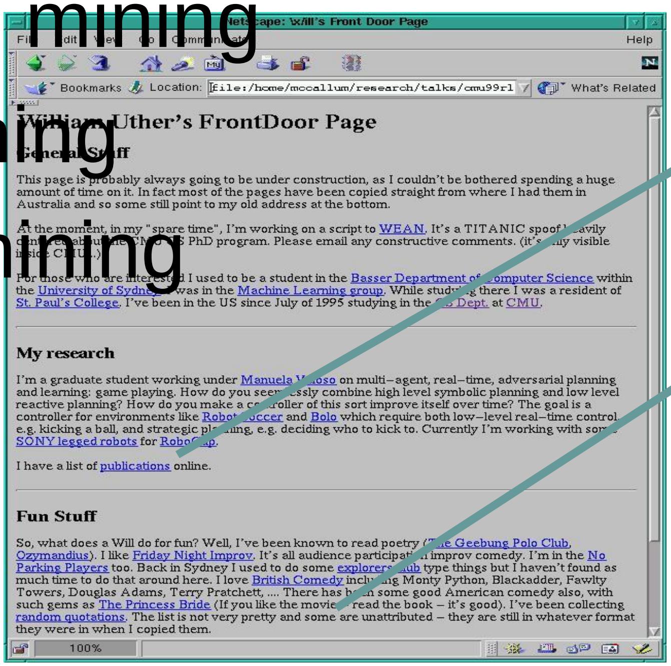


$$R_t = \int_0^\infty e^{-\beta\tau} r_{t+\tau} d\tau$$

$$Q(s,a) \rightarrow Q(s,a) + \alpha\left[\int_{t_1}^{t_2} e^{-\beta(\tau-t_1)} r_\tau d\tau + e^{-\beta(t_2-t_1)} \max_a Q(s',a') - Q(s,a)\right]$$

强化学习应用_信息检索

- Web crawling
- Domain-specific Search Engineer
- Web-log mining
- Text mining
- Image mining


Netscape: Will's Front Door Page
File Edit View Go Communicator Help
Bookmarks Location: file:/home/mccallum/research/talks/cmu99rl What's Related
100%

William Uther's FrontDoor Page

General Stuff

This page is probably always going to be under construction, as I couldn't be bothered spending a huge amount of time on it. In fact most of the pages have been copied straight from where I had them in Australia and so some still point to my old address at the bottom.

At the moment, in my "spare time", I'm working on a script to WEAN. It's a TITANIC spoof heavily centred about the CMU CS PhD program. Please email any constructive comments. (it's only visible inside CMU...)

For those who are interested I used to be a student in the Basser Department of Computer Science within the University of Sydney. I was in the Machine Learning group. While studying there I was a resident of St. Paul's College. I've been in the US since July of 1995 studying in the CS Dept. at CMU.

My research

I'm a graduate student working under Manuela Veloso on multi–agent, real–time, adversarial planning and learning: game playing. How do you seamlessly combine high level symbolic planning and low level reactive planning? How do you make a controller of this sort improve itself over time? The goal is a controller for environments like Robot Soccer and Bolo which require both low–level real–time control, e.g. kicking a ball, and strategic planning, e.g. deciding who to kick to. Currently I'm working with some SONY legged robots for RoboCup.

I have a list of publications online.

Fun Stuff

So, what does a Will do for fun? Well, I've been known to read poetry (The Geebung Polo Club, Ozymandius). I like Friday Night Improv. It's all audience participation improv comedy. I'm in the No Parking Players too. Back in Sydney I used to do some explorers club type things but I haven't found as much time to do that around here. I love British Comedy including Monty Python, Blackadder, Fawlty Towers, Douglas Adams, Terry Pratchett, ... There has been some good American comedy also, with such gems as The Princess Bride (If you like the movie – read the book – it's good). I've been collecting random quotations. The list is not very pretty and some are unattributed – they are still in whatever format they were in when I copied them.

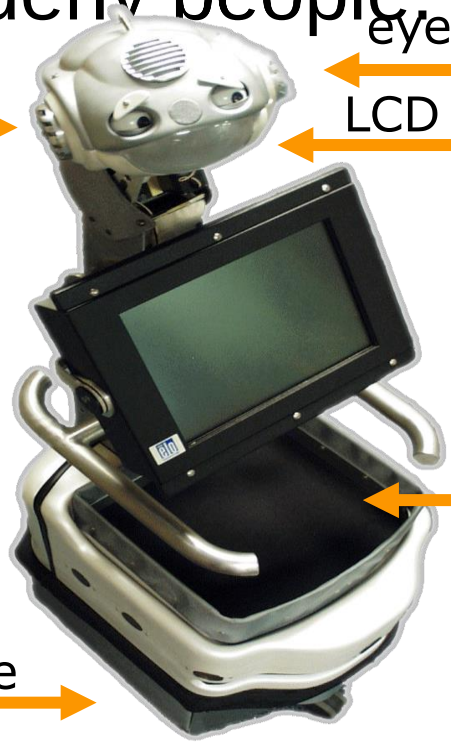
**Q("Publications",
"My research",
"Will's FrontDoor Page")**

**Q("British Comedy",
"Fun Stuff",
"Will's FrontDoor Page")**

Nursing Robot Pearl

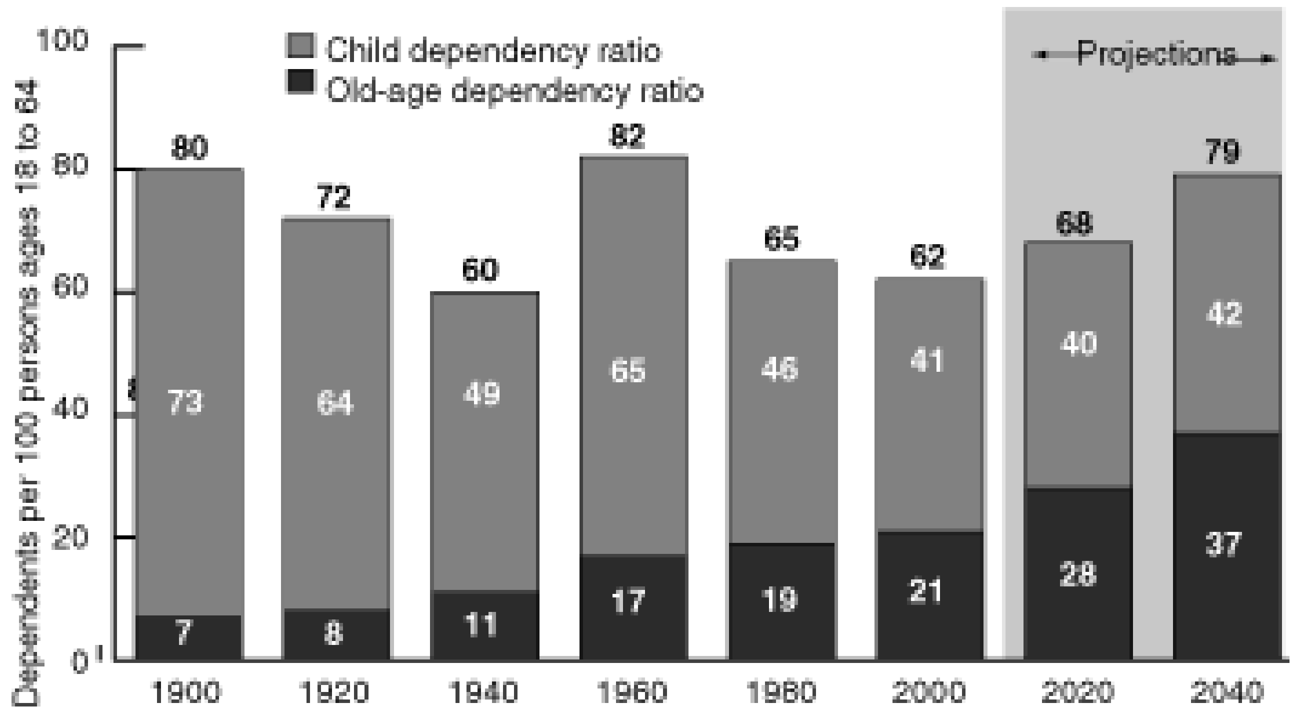
Pear l 介绍

- Pearl is a prototype nursing robot, providing assistance to both nurses and elderly people.



CogRob2002 workshop

老年人数



- 450,000 more nurses needed by 2008
- campaign to recruit and retain nurses and other health care providers

机器人助理保健

Management support of ADLs

Reminding to eat, drink, & take meds

Providing information (TV, weather)

Linking the caregiver to resources

Monitoring Rx adherence & safety

Providing physical assistance

Calling for help in emergencies

Moving things around

Supporting inter-personal communication

Enabling use of remote health services

References

- Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning- An Introduction*.
- Hastie, T., Tibshirani, R. and Friedman, J. (2001). *The Elements of Statistical Learning-Data Mining, Inference and Prediction*.
- Murphy, S.A. (2003). *Optimal Dynamic Treatment Regimes. JRSS-B.*
- Blatt, D., Murphy, S.A. and Zhu, J. (2004).
  A-Learning for Approximate Planning.
- Murphy, S.A. (2004). *A Generalization Error for Q-Learning.*
- *D. P. Bertsekas and J. N. Tsitsiklis* (1996). *Neuro-Dynamic Programming*.
- 宋志伟，陈小平，2003. 仿真机器人足球中的强化学习. 《机器人》，24(7S):761-766.
- Joelle Pineau and Sebastian Thrun. High-level robot behavior control using POMDPs. CogRob2002 workshop.

Thank You

Question!

Intelligence Science

http://www.intsci.ac.cn/