Artificial Intelligence

B.Tech(AIDS)_V Semester

Academic Year: 2024-2025

Dr D. L. Sreenivasa Reddy

Email: dlsrinivasareddy_aids@cbit.ac.in

Syllabus

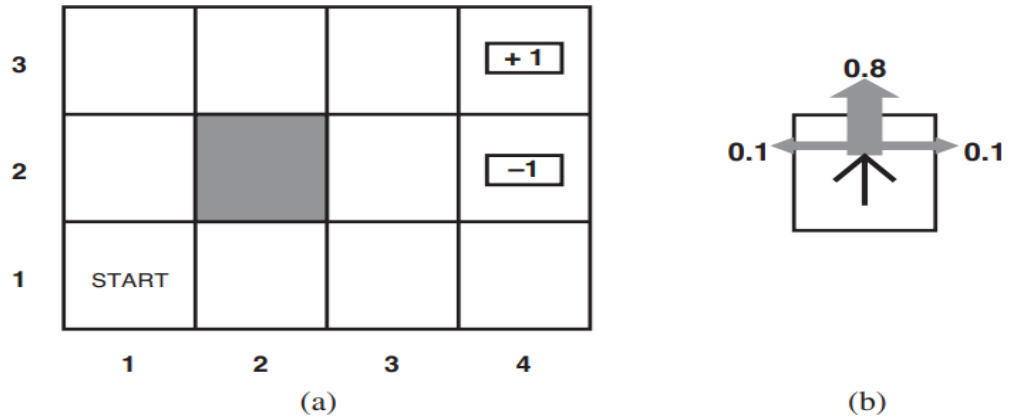
UNIT-V

Reinforcement Learning: Introduction, Passive reinforcement learning, Active Reinforcement Learning, Generalization in reinforcement learning, adaptive dynamic programming, temporal difference learning, active reinforcement learning- Q learning.

SEQUENTIAL DECISION PROBLEMS

Actions available to the agent in each state are given by ACTIONS(s)=A(s)

Ex:



Actions in every state = **Up, Down, Left, and Right,**

Goal states are +1 or –

Transition model of the environment: The “intended” outcome occurs with probability 0.8,

The agent moves at right angles to the intended direction probability 0.2.

The two terminal states have reward +1 and –1,

All other states have a reward of –0.04.

Solution : [Up, Up, Right, Right, Right], probability (0.8)**5= 0.32768, 0.1 4 × 0.8, for a grand total of 0.32776

example, if the agent reaches the +1 state after 10 steps, its total utility will be 0.6. The negative reward of –0.04 gives the agent an incentive to reach (4,3) quickly

- **Transition model:** Describes the outcome of each action in each state
- **Example:** P(s ’ | s, a) to denote the probability of reaching state s ’ if action **a** is done in state s.
- **Transitions in Markovian(HMM):** Probability of reaching **s ’** from s depends only on **s** and not on the history of earlier states
- **P(s ’ | s, a)** as a big three-dimensional table containing probabilities.
- **Transition model** can be represented as a dynamic Bayesian network
- **Utility function** will depend on a sequence of states(an environment history)rather than on a single state
- **Reward** is denoted by **R(s):**which may be positive or negative, but must be bounded
- The utility of an environment history is just the **sum** of the rewards received.

MARKOV DECISION PROCESS

MDP: A sequential decision problem for a fully observable, stochastic environment with a Markovian transition model and additive rewards is called a Markov decision process.

MDP consists

1. A set of states (S)
2. An initial state s0)
3. A set ACTIONS(s) of actions in each state
4. A transition model P(s ’ | s, a)
5. A reward function R(s).

Policy: A solution must specify what the agent should do for any state, that the agent might reach. A solution of this kind is called a **policy**

- A policy is denoted by π
- The action recommended by the policy π for state s is denoted by π(s) .
- If the agent has a complete policy, then no matter what the outcome of any action, the agent will always know what to do next.

Optimal policy:

- The quality of a policy is therefore measured by the **expected utility** of the possible environment histories generated by the policy.
- An optimal policy is a policy that yields the **highest expected utility**
- Optimal policy is denoted by π^*
- For a given Optimal **Policy** π^*, the agent decides what to do by consulting its current percept, which tells it the current state s, and then executing the action $\pi^*(s)$

Utility Function over time:

- Performance of the agent was measured by a **sum of rewards** for the states visited.
- This choice of performance measure is not arbitrary.
- But it is not the only possibility for the utility function on environment histories.
- Utility function is denoted by Uh([s0, s1, . . . , sn]).
- **Finite horizon** : Finite horizon means that there is a fixed time N after which nothing matters

  Uh([s0, s1, . . . , sN+k])= Uh([s0, s1, . . . , sN]) for all k>0
- The **optimal policy** for a finite horizon is **nonstationary**

STATIONARY POLICY:

With no fixed time limit

Additive rewards: The utility of a state sequence is

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \cdots .$$

Discounted rewards: The utility of a state sequence is

$$U_h([s_0, s_1, s_2, \ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \cdots$$

where the discount factor γ is a number between 0 and 1,
Discounting appears to be a good model of both animal and human preferences over time.
A discount factor of γ is equivalent to an interest rate of $(1/\gamma) - 1$.

With discounted rewards, the utility of an infinite sequence is *finite*. In fact, if $\gamma < 1$ and rewards are bounded by $\pm R_{\max}$, we have

$$U_h([s_0, s_1, s_2, \ldots]) = \sum_{t=0}^{\infty} \gamma^t R(s_t) \leq \sum_{t=0}^{\infty} \gamma^t R_{\max} = R_{\max}/(1-\gamma)$$

Optimal policies and the utilities of states

- Utility of a given state sequence is the sum of discounted rewards obtained during the sequence.
- Compare policies by comparing the expected utilities obtained when executing them.
- The agent is in some initial state s and define St (a random variable) to be the state the agent reaches at time t when executing a particular policy π.
- The probability distribution over state sequences S1, S2, . . . , is determined by the initial state s, the policy π, and the transition model for the environment.
- where the expectation is with respect to the probability distribution over state sequences determined by s and π
- out of all the policies the agent could choose to execute starting in s, one (or more) will have higher expected utilities than all the others.

The expected utility obtained by executing π starting in s is given by

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] ,$$

$$\pi_s^* = \operatorname*{argmax}_{\pi} U^{\pi}(s) \qquad \pi^*(s) = \operatorname*{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

VALUE ITERATION

- For calculating an optimal policy, value iteration is used.
- The basic idea is to calculate the utility of each state and then use the state utilities to select an optimal action in each state.

The Bellman equation for utilities:

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action.

$$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

This is called the Bellman equation. The Solution for this bellman equation above is

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

$$\begin{aligned} U(1,1) = -0.04 + \gamma \max[\ & 0.8U(1,2) + 0.1U(2,1) + 0.1U(1,1), \\ & 0.9U(1,1) + 0.1U(1,2), \\ & 0.9U(1,1) + 0.1U(2,1), \\ & 0.8U(2,1) + 0.1U(1,2) + 0.1U(1,1)\ ] \end{aligned}$$

3	0.812	0.868	0.918	+1
2	0.762		0.660	−1
1	0.705	0.655	0.611	0.388
	1	2	3	4

The utilities of the states in the 4 × 3 world, calculated with γ =1 and R(s)= − 0.04 for nonterminal states

VALUE ITERATION

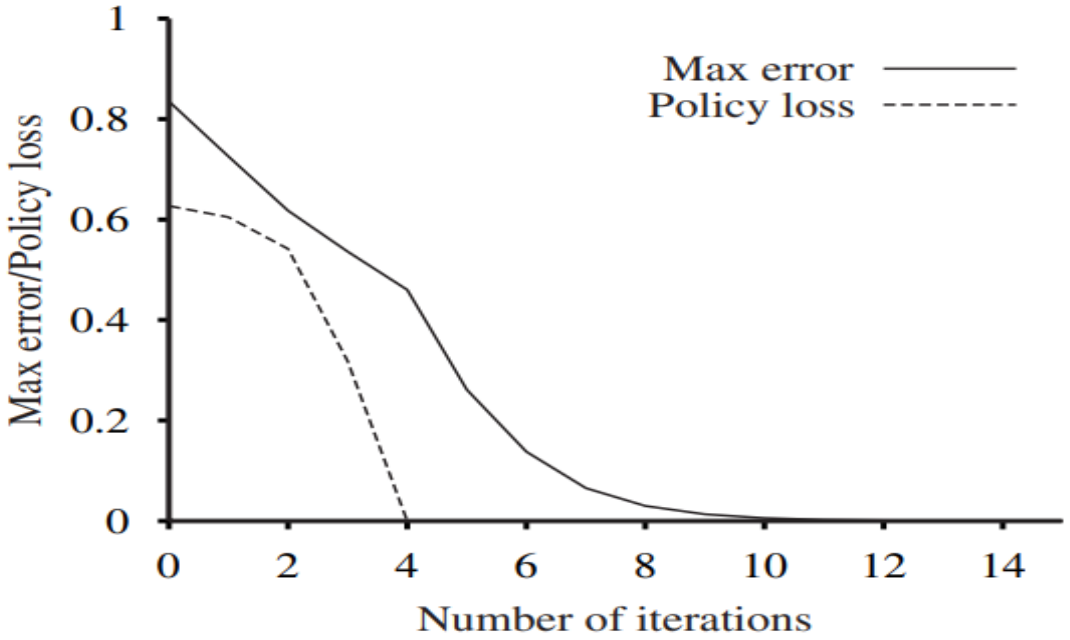
Bellman update Algorithm: $$U_{i+1}(s) \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U_i(s')$$

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

POLICY ITERATION

- The policy iteration algorithm alternates the following two steps, beginning from some initial policy π0:
- **Policy evaluation:** Given a policy πi , calculate $U_i = U^{\pi i}$, the utility of each state if π_i were to be executed.
- **Policy improvement:** Calculate a new MEU(Maximum Error Utility) policy π_{i+1}, using one-step look-ahead based on U_i
- The algorithm terminates when the policy improvement step yields no change in the utilities.
- The utility function Ui is a fixed point of the Bellman update, so it is a solution to the Bellman equations, and πi must be an optimal policy.
- Because there are only finitely many policies for a finite state space, and each iteration can be shown to yield a better policy, policy iteration must terminate.



The maximum error ||Ui − U|| of the utility estimates and the policy loss ||U πi − U||, as a function of the number of iterations of value iteration.

POLICY ITERATION Algm For Calculating Optimal Policy

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

VALUE ITERATION

- For calculating an optimal policy, it is to calculate the utility of each state and then use the state utilities to select an optimal action in each state.

The Bellman equation for utilities:

The utility of a state is the immediate reward for that state plus the expected discounted utility of the next state, assuming that the agent chooses the optimal action

The utility of a state is given by $$U(s) = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

This is called the Bellman equation

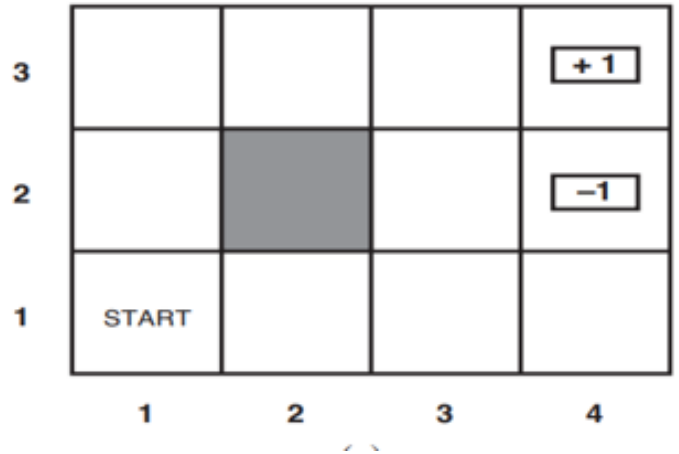
PARTIALLY OBSERVABLE MDPS(POMDP)

- The description of Markov decision processes assumed that the environment was fully observable.
- With this assumption, the agent always knows which state it is in.
- This, combined with the Markov assumption for the transition model
- I.e the optimal policy depends only on the current state. When the environment is only partially observable, the situation is, one might say, much less clear.
- The agent does not necessarily know which state it is in, so it cannot execute the action π(s) recommended for that state.
- Furthermore, the utility of a state s and the optimal action in s depend not just on s, but also on how much the agent knows when it is in s.
- For these reasons, partially observable MDPs or POMDPs are usually viewed as much more difficult than ordinary MDPs. However, the real world is in partially Observable environment.

PARTIALLY OBSERVABLE MDPS(POMDP)

- A POMDP has the same elements as an MDP—the transition model P(s ’ | s, a), actions A(s), and reward function R(s)—but, like the partially observable search problems.
- nondeterministic and partially observable planning problems and identified as the belief state
- The set of actual states the agent might be in—as a key concept for describing and calculating solutions.
- In **POMDPs**, the belief state b becomes a probability distribution over all possible states

- The initial belief state for the 4 × 3 POMDP could be the uniform distribution over the nine nonterminal states, i.e., h<1/9 , 1/9 , 1/9 , 1/9 , 1/9 , 1/9 , 1/9 , 1/9 , 1/9 , 0, 0>
- write b(s) for the probability assigned to the actual state s by belief state b.



The agent can calculate its current belief state as the conditional probability distribution over the actual states given the sequence of percepts and actions so far.

This is essentially called filtering task.

- If b(s) was the previous belief state, and the agent does action a and then perceives evidence e, then the new belief state is given by

$$b'(s') = \alpha\, P(e \mid s') \sum P(s' \mid s, a) b(s)$$

- where α is a normalizing constant that makes the belief state sum to 1
- Update operator for filtering $b' = \text{FORWARD}(b, a, e)$
- The fundamental insight required to understand POMDPs is: the optimal action depends only on the agent's current belief state.
- i.e, the optimal policy can be described by a mapping π ∗ (b) from belief states to actions.
- It does not depend on the actual state the agent is in.Because the agent does not know its actual state; all it knows is the belief state.
- Hence, the decision cycle of a POMDP agent can be broken down into the following three steps:

1. Given the current belief state b, execute the action $a = \pi^*(b)$
2. Receive percept e.
3. Set the current belief state to $\text{FORWARD}(b, a, e)$ and repeat.

- The belief state: b ′ = FORWARD(b, a, e)
- Though, the subsequent percept is not yet known, so the agent might arrive in one of several possible belief states b ′, depending on the percept that is received.
- The probability of perceiving e, given that a was performed starting in belief state b, is given by summing over all the actual states s ′ that the agent might reach:

$$P(e|a,b) = \sum_{s'} P(e|a,s',b)P(s'|a,b)$$
$$= \sum_{s'} P(e\,|\,s')P(s'|a,b)$$
$$= \sum_{s'} P(e\,|\,s') \sum P(s'\,|\,s,a)b(s)\ .$$

$$P(b'\,|\,b,a) = P(b'|a,b) = \sum_{e} P(b'|e,a,b)P(e|a,b)$$
$$= \sum_{e} P(b'|e,a,b) \sum_{s'} P(e\,|\,s') \sum_{s} P(s'\,|\,s,a)b(s)$$

where $P(b'|e,a,b)$ is 1 if $b' = \text{FORWARD}(b,a,e)$ and 0 otherwise.

Reward function for belief states $\rho(b) = \sum_{s} b(s)R(s)$

Value iteration for POMDPs:

Consider an optimal policy $\pi *$ and its application in a specific belief state b: the policy generates an action, then, for each subsequent percept, the belief state is updated and a new action is generated, and so on.

For this specific b, the policy is exactly equivalent to a **conditional plan** for nondeterministic and partially observable problems.

There are two observations:

1. Let the utility of executing a *fixed* conditional plan p starting in physical state s be $\alpha_p(s)$. Then the expected utility of executing p in belief state b is just $\sum_s b(s)\alpha_p(s)$, or $b \cdot \alpha_p$ if we think of them both as vectors. Hence, the expected utility of a fixed conditional plan varies *linearly* with b; that is, it corresponds to a hyperplane in belief space.
2. At any given belief state b, the optimal policy will choose to execute the conditional plan with highest expected utility; and the expected utility of b under the optimal policy is just the utility of that conditional plan:

$$U(b) = U^{\pi^*}(b) = \max_p b \cdot \alpha_p \ .$$

If the optimal policy π^* chooses to execute p starting at b, then it is reasonable to expect that it might choose to execute p in belief states that are very close to b; in fact, if we bound the depth of the conditional plans, then there are only finitely many such plans and the continuous space of belief states will generally be divided into *regions*, each corresponding to a particular conditional plan that is optimal in that region.

DOMINATED PLAN:

Conditional plan whose initial action is a and whose depth-d − 1 subplan for percept e is p.e; then

Conditional plan is: $$\alpha_p(s) = R(s) + \gamma \left(\sum_{s'} P(s' \mid s, a) \sum_{e} P(e \mid s') \alpha_{p.e}(s') \right)$$

Dominated Plan Algm is:

function POMDP-VALUE-ITERATION($pomdp, \epsilon$) **returns** a utility function
  inputs: $pomdp$, a POMDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    sensor model $P(e \mid s)$, rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', sets of plans p with associated utility vectors α_p

  $U' \leftarrow$ a set containing just the empty plan $[\,]$, with $\alpha_{[]}(s) = R(s)$
  repeat
    $U \leftarrow U'$
    $U' \leftarrow$ the set of all plans consisting of an action and, for each possible next percept,
      a plan in U with utility vectors computed according to Equation (17.13)
    $U' \leftarrow$ REMOVE-DOMINATED-PLANS(U')
  until MAX-DIFFERENCE(U, U') $< \epsilon(1-\gamma)/\gamma$
  return U

Reinforcement Learning

How agents can learn what to do in the absence of labeled examples

Consider the problem of learning to play chess.

A **supervised learning** agent needs to be told the correct move for each position it encounters.

- but such feedback is seldom available

In the absence of feedback, an agent can learn a transition model for its own moves and and can perhaps learn to predict the opponent's moves.

- without some **feedback** about what is good and what is bad, the agent will have no grounds for deciding which move to make

A typical kind of feedback is called a **reward**, or **reinforcement**

- in games like chess, the reinforcement is received only at the end of the game
- in other problems, the rewards come more frequently (in ping-pong, each point scored can be considered a reward)

The reward is part of the **input percept**, but the agent must be "hardwired" to recognize that part as a reward

- pain and hunger are negative rewards
- pleasure and food intake are positive rewards

Reinforcement learning

Reinforcement learning might be considered to encompass all of artificial intelligence:

- an agent is placed in an environment and must learn to behave successfully therein
- in many complex domains, reinforcement learning is the only feasible way to train a program to perform at high levels

We will consider three of the agent designs

- a **utility-based agent** learns a utility function and uses it to select actions that maximize the expected outcome utility
  - must also have a model of the environment, because it must know the states to which its actions will lead
- a **Q-learning** agent learns an action-utility function (Q-function) giving the expected utility of taking a given action in a given state
- a **reflex agent** learns a policy that maps directly from states to actions

The task of **passive learning** is to learn the utilities of the states, where the agent's policy is fixed.

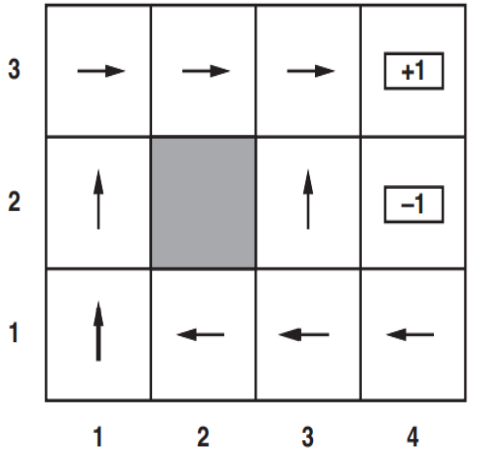
In **active learning** the agent must also learn what to do.

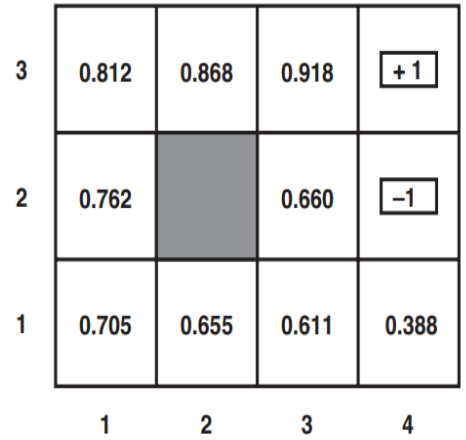
- It involves some form of **exploration**: an agent must experience as much as possible of its environment in order to learn how to behave in it.

Passive reinforcement learning

The agent's **policy is fixed** (in state s, it always executes the action $\pi(s)$).

The goal is to learn how good the policy is , that is, to **learn the utility function** $U^{\pi}(s) = E[\Sigma_{t=0,\ldots,\infty} \gamma^{t}.R(s_t)]$





The agent does not know the **transition model** P(s'|s,a) nor does it know the **reward function** R(s).

where $R(s)$ is the reward for a state, S_t (a random variable) is the state reached at time t when executing policy π, and $S_0 = s$. We will include a **discount factor** γ in all of our equations,

A core approach:

- the agent executes a set of trials in the environment using its policy π
- its percept supply both the current state and the reward received at that state

$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
$(1,1)_{-0.04} \rightarrow (2,1)_{-0.04} \rightarrow (3,1)_{-0.04} \rightarrow (3,2)_{-0.04} \rightarrow (4,2)_{-1}$

Direct utility estimation

The idea is that the utility of a state is the expected total reward from that state onward (expected **reward-to-go**).

- for state (1,1) we get a sample total reward 0.72 in the first trial
- for state (1,2) we have two samples 0.76 and 0.84 in the first trial

The same state may appear in more trials (or even in the same trial) so we keep running average for each state.

Direct utility estimation is just an instance of supervised learning (input = state, output = reward-to-go)

Major inefficiency:

- The utilities of states are not independent!
- The utility values obey the **Bellman equations** for a fixed policy
  $U^{\pi}(s) = R(s) + \gamma \Sigma_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s')$
- We search for U in a hypothesis space that is much larger than it needs to be (it includes many functions that violate the Bellman equations); for this reason, the algorithm often converges very slowly.

An **adaptive dynamic programming (ADP)** agent takes advantage of the Bellman equations.

The agent learns:

- **the transition model** $P(s' | s, \pi(s))$
  - Using the frequency with which s is reached when executing a in s. For example $P((2,3) | (1,3), \text{Right}) = 2/3$.
- **rewards** $R(s)$
  - directly observed

The **utility of states** is calculated from the Bellman equations, for example using the modified policy iteration.

ADP algorithm

function PASSIVE-ADP-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    mdp, an MDP with model P, rewards R, discount γ
    U, a table of utilities, initially empty
    N_{sa}, a table of frequencies for state–action pairs, initially zero
    $N_{s'|sa}$, a table of outcome frequencies given state–action pairs, initially zero
    s, a, the previous state and action, initially null

  if s' is new **then** $U[s'] \leftarrow r'$; $R[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_{sa}[s, a]$ and $N_{s'|sa}[s', s, a]$
    for each t such that $N_{s'|sa}[t, s, a]$ is nonzero **do**
      $P(t \mid s, a) \leftarrow N_{s'|sa}[t, s, a] \,/\, N_{sa}[s, a]$
  $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
  if s'.TERMINAL? **then** $s, a \leftarrow$ null **else** $s, a \leftarrow s', \pi[s']$
  return a

Temporal-difference learning

We can use the observed transitions to adjust utilities of the states so that they agree with the constraint equations.

Example:

- consider the transitions from (1,3) to (2,3)
- suppose that, as a result of the first trial, the utility estimates are
  $U^{\pi}(1,3) = 0.84$ and $U^{\pi}(2,3) = 0.92$
- if this transition occurred all the time, we would expect the utility to obey the equations (if $\gamma = 1$)
  $U^{\pi}(1,3) = -0.04 + U^{\pi}(2,3)$
- so the utility would be
  $U^{\pi}(1,3) = 0.88$
- hence the current estimate $U^{\pi}(1,3)$ might be a little low and should be increased

Discount factor = γ

In general, we apply the following update (α is the **learning rate** parameter):

$$U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha.(R(s) + \gamma.U^{\pi}(s') - U^{\pi}(s))$$

The above formula is often called the **temporal-difference** (TD) equation.

TD algorithm

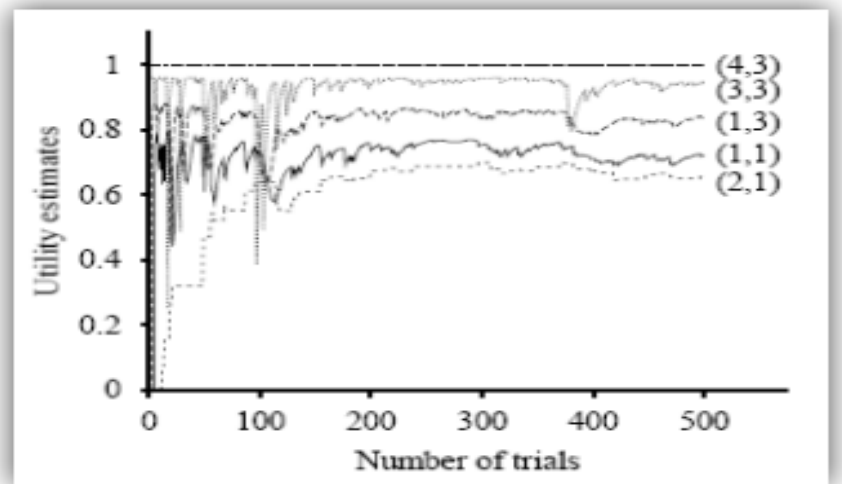
function PASSIVE-TD-AGENT($percept$) **returns** an action
  inputs: $percept$, a percept indicating the current state s' and reward signal r'
  persistent: π, a fixed policy
    U, a table of utilities, initially empty
    N_s, a table of frequencies for states, initially zero
    s, a, r, the previous state, action, and reward, initially null

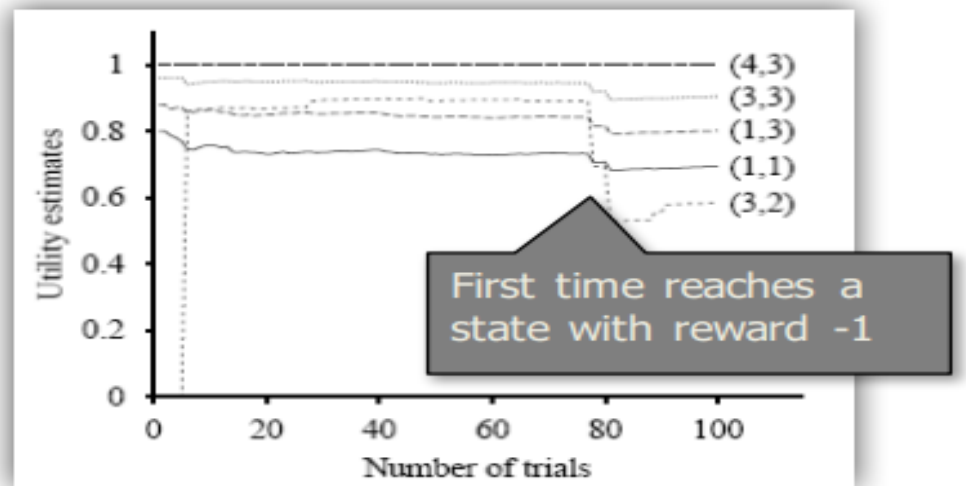
  if s' is new **then** $U[s'] \leftarrow r'$
  if s is not null **then**
    increment $N_s[s]$
    $U[s] \leftarrow U[s] + \alpha(N_s[s])(r + \gamma U[s'] - U[s])$
  if s'.TERMINAL? **then** $s, a, r \leftarrow$ null **else** $s, a, r \leftarrow s', \pi[s'], r'$
  return a

Comparison of ADP and TD

Both ADP and TD approaches try to make **local adjustments** to the utility estimates in order to make each state „agree“ with its successors.

- **Temporal difference**
  - does not need a transition model to perform updates
  - adjusts a state to agree with its *observed* successor
  - a single adjustment per observed transition
- **Adaptive dynamic programming**
  - adjusts a state to agree with *all* of the successors
  - makes as many adjustments as it needs to restore consistency between the utility estimates





Active reinforcement learning

A passive learning agent has a fixed policy that determines its behavior.

An **active agent** must decide what actions to take.

Let us begin with the adaptive dynamic programming agent

- the utilities it needs to learn are defined by the optimal policy; they obey the Bellman equations
  $U^{\pi}(s) = R(s) + \gamma \max_a \Sigma_{s'} P(s' | s, a) U^{\pi}(s')$
- these equations can be solved to obtain the utility function

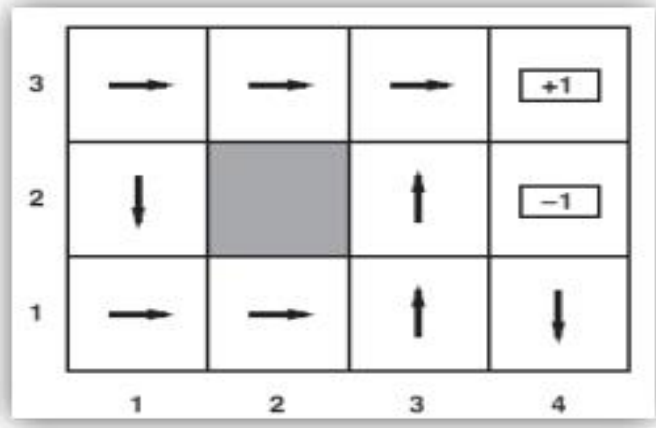
What to do at each step?

- the agent can extract an optimal action to maximize the expected utility
- then it should simply execute the action the optimal policy recommends
- Or should it?

Greedy agent

An example of policy found by the **active ADP agent**.

This is not an optimal policy!
What did happen?



The agent found a route (2,1), (3,1), (3,2), (3,3) to the goal with reward +1.

After experimenting with minor variations, it sticks to that policy.

As it does not learn utilities of the other states, it never finds the optimal route via (1,2), (1,3), (2,3), (3,3).

We call this agent the **greedy agent**.

How can it be that **choosing the optimal action leads to suboptimal results**?

- the learned model is not the same as the true environment; what is optimal in the learned model can therefore be suboptimal in the true environment
- *actions do more than provide rewards; they also contribute to learning the true model by affecting the percepts that are received*
- by improving the model, the agent will receive greater rewards in the future

An agent therefore must made tradeoff between **exploitation** to maximize its reward and **exploration** to maximize its long-term well-being.

What is the right trade-off between exploration and exploitation?

- **pure exploration** is of no use if one never puts that knowledge in practice
- **pure exploitation** risks getting stuck in a rut

Basic idea

- at the beginning striking out into the unknown in the hopes of discovering a new and better life
- with greater understanding less exploration is necessary

An n-armed bandit

- a slot machine with n-levers (or n one-armed slot machines)

Which lever to play?

- The one that has paid off best, or maybe one that has not been tried?

Exploration policies

The agent chooses a random action a fraction 1/t of the time and follows the greedy policy otherwise

- it does eventually converge to an optimal policy, but it can be extremely slow

A more sensible approach would give some **weight to actions** that the agent has **not tried very often**, while tending to **avoid actions** that are believed to be of **low utility.**

- assign a higher utility estimate to relatively unexplored state-action pairs
- value iteration may use the following update rule

$$U^+(s) \leftarrow R(s) + \gamma \max_a f(\Sigma_{s'} P(s'|s, a)\, U^+(s'), N(s,a))$$

  - N(*s*,*a*) is the number of times action *a* has been tried in state *s*
  - $U^+(s)$ denotes the optimistic estimate of the utility
  - f(u,n) is called the **exploration function;** it determines how greed is traded off against curiosity (should be increasing in *u* and decreasing in *n*)
    - for example $f(u,n) = R^+$ if $n<N_e$, otherwise u
      (R^+ is an optimistic estimate of the best possible reward obtainable in any state)

The fact that U^+ rather than U appears in the right-hand side is very important.

- As exploration proceeds, the states and actions near the start might well be tried a large number of times.
- If we used U, the more pessimistic utility estimate, then the agent would soon become disinclined to explore further afield.
- The benefits of exploration are propagated back from the edges of unexplored regions so that actions that lead toward unexplored regions are weighted more highly.

Q-learning

Let us now consider how to construct an **active temporal-difference learning agent**.

- The update rule remains unchanged:
  $U(s) \leftarrow U(s) + \alpha.(R(s) + \gamma.U(s') - U(s))$
- The model acquisition problem for the TD agent is identical to that for the ADP agent.

There is an alternative TD method, called **Q-learning**

- Q(s,a) denotes the value for doing action *a* in state *s*
- the q-values are directly related to utility values as follows:
  - $U(s) = \max_a Q(s,a)$
- the TD-agent that learns a Q-function does not need a model form $P(s' | s, a)$
  - Q-learning is called a **model-free** method
- we can write a constraint equation that must hold at equilibrium:
  - $Q(s,a) = R(s) + \gamma \sum_{s'} P(s' | s, a) \max_{a'} Q(s',a')$
  - This does require that a model $P(s' | s, a)$ also be learned!
- The TD approach requires no model of state transitions – all it needs are the Q values
  - **$Q(s,a) \leftarrow Q(s,a) + \alpha.(R(s) + \gamma \max_{a'} Q(s',a') - Q(s,a))$**
  - it is calculated whenever action *a* is executed in state *s* leading to state *s'*

Q-learning algorithm

function Q-LEARNING-AGENT(*percept*) **returns** an action

inputs: *percept*, a percept indicating the current state s' and reward signal r'

persistent: Q, a table of action values indexed by state and action, initially zero
N_{sa}, a table of frequencies for state–action pairs, initially zero
s, a, r, the previous state, action, and reward, initially null

if TERMINAL?(s) **then** $Q[s, None] \leftarrow r'$

if s is not null **then**

increment $N_{sa}[s, a]$

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

$s, a, r \leftarrow s', \text{argmax}_{a'} f(Q[s', a'], N_{sa}[s', a']), r'$

return a

State-Action-Reward-State-Action

- a close relative to Q-learning with the following update rule

$$Q(s,a) \leftarrow Q(s,a) + \alpha.(R(s) + \gamma.Q(s',a') - Q(s,a))$$

- the rule is applied at the end of each s,a,r,s',a' quintuplet, i.e. after applying action a'

Comparison of SARSA and Q-learning:

- for a greedy agent the two algorithms are identical (the action a' maximizing Q(s',a') is always selected)
- When exploration is assumed there is a subtle difference
  - Q-learning pays no attention to the actual policy being followed – it is an off-policy learning algorithm (can learn how to behave well even when guided by a random or adversarial exploration policy)
  - SARSA is more realistic: it is better to learn a Q-function for what actually happen rather than what the agent would like to happen
    - works if the overall policy is even partly controlled by other agents

Final notes

Both Q-learning and SARSA learn the optimal policy, but do so at much slower rate than the ADP agent.

- the local updates do not enforce consistency among all the Q-values via the model

Is it better to learn a model and a utility function (ADP) or to learn an action-utility function with no model (Q-learning, SARSA)?

- One of the key historical characteristics of much of AI research is its adherence to the **knowledge-based** approach; this adheres to assumption that the best way to represent the agent function is to build a representation of some aspects of the environment in which the agent is situated.
- Availability of model-free methods such as Q-learning means that the knowledge-based approach is unnecessary.
- Intuition is that as the environment becomes more complex, the advantages of knowledge-based approach become more apparent.