UNIT 4 PREDICTION AND CONTROL BY DYNAMIC PROGRAMMING

Ami Munshi

Syllabus

4	**Prediction and Control by Dynamic Programing** Overview of dynamic programing for MDP, definition and formulation of planning in MDPs, Banach fixed point theorem, Policy Evaluation, Policy Improvement, Policy Iteration, Value Iteration, Efficiency of Dynamic Programming	05

Course Outcomes

After completion of the course, students will be able to –

1. Apply the basics of Reinforcement Learning (RL) to compare with traditional control design
2. Correlate how RL relates and fits into the broader umbrella of machine learning, deep learning
3. Recommend value functions and appropriate algorithms for optimal decision-making
4. Design a dynamic programming approach to an industrial control problem

References

Text Books

1. Laura Graesser and Wah Loon Keng, *Foundations of Deep Reinforcement Learning: Theory and Practice in Python*, 1st Edition, Pearson India/Padmavati Publisher, 2022.
2. Richard Sutton and Andrew Barto, *Reinforcement Learning: An Introduction*, 2nd Edition, MIT Press, 2018.
3. Abhishek Nandy and Manisha Biswas, *Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python*, 1st Edition, Apress Publisher, 2017.

Reference Books

1. Nimish Sanghi, *Deep Reinforcement Learning with Python: With PyTorch, TensorFlow and OpenAI*, 2nd edition, Apress Publisher, 2021.
2. Alexander Zai and Brandon Brown, *Deep Reinforcement Learning in Action*, 1st Edition, Manning Publisher, 2020.
3. Csaba Szepesvari, *Algorithms for Reinforcement Learning*, 3rd Edition, Morgan & Claypool Publisher, 2019.

Russel and Norwig "Artificial Intelligence a Modern Approach", 4th edition

Dr Saeed Saeedvand
https://www.youtube.com/watch?v=nMIRhsgcTpY&list=PLZ_sI4f41TGvthD8dA7daahlbLV0yDW0w&index=3

Revisiting some concepts

- Policy optimization
  - Optimizing mapping state to action
  - Hence we need to estimate value function to evaluate current state of the environment
- Value of optimal policy is $v^*(s_t)$
- Hence $v^*(s_t) \geq v_\pi(s_t)$

Summary of all the equations

1) State-value function is defined as the expectation value of future total return given a current state s and a policy π.

$$v_\pi(s) \equiv \mathbb{E}_\pi[G_t|S_t = s], \text{ for all } s \in \mathcal{S}$$

2) Action-value function is defined as the expectation value of future total return give a current state s and action, plus a policy π.

$$q_\pi(s,a) \equiv \mathbb{E}_\pi[G_t|S_t = s, A_t = a]$$

3) Bellman equations (recursive relationship). With the aid of backup diagram, we can easily calculate the state-value function and the action-value function for a given policy π, in a recursive way:

$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)(r + \gamma v_\pi(s'))$$

Optimal Policies and Optimal Value Functions

- A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states
- In other words, $\pi \geq \pi'$ if and only if
  - $v_\pi(s) \geq v'_\pi(s) for\ all\ s \epsilon S$
- This is an **optimal policy**

Optimal Policies and Optimal Value Functions

- Optimal state value function for optimal policy is given by

$$v_*(s) \doteq \max_\pi v_\pi(s),$$

for all $s \in \mathcal{S}$.

- Optimal state action value function for optimal policy is given by

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a),$$

$$q_*(s,a) = \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a].$$

Bellman optimality equation for v_π

$$
\begin{aligned}
v_*(s) &= \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s,a) \\
&= \max_a \mathbb{E}_{\pi_*}[G_t \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}_{\pi_*}[R_{t+1} + \gamma G_{t+1} \mid S_t = s, A_t = a] \\
&= \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a] \\
&= \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma v_*(s')\big].
\end{aligned}
$$

- Intuitively, the Bellman optimality equation expresses the fact that the value of a state under an optimal policy must equal the expected return for the best action from that state

Bellman optimality equation for q_*

$$
\begin{aligned}
q_*(s,a) &= \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big] \\
&= \sum_{s',r} p(s', r \,|\, s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big].
\end{aligned}
$$

Dynamic Programming

- Dynamic programming (DP) refers to a collection of algorithms that
  - Can be used to compute optimal policies given a perfect model of the environment as a MDP
- Assumption
  - Environment is a finite MDP
  - That is, we assume that its state, action, and reward sets, $S, A, and\ R$, are finite
  - That its dynamics are given by a set of probabilities $p(s', r|s, a)$, for all $s \in S, a \in A(s), r \in R$
- Key idea of DP, and of reinforcement learning generally, is
  - the use of value functions to organize and structure the search for good policies

Dynamic Programming

- We can easily obtain optimal policies once we have found the optimal value functions, $v_* \; or \; q_*$, which satisfy the Bellman optimality equations

$$v_*(s) = \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s, A_t = a]$$
$$= \max_a \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma v_*(s')\Big],$$

or

$$q_*(s,a) = \mathbb{E}\Big[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \;\Big|\; S_t = s, A_t = a\Big]$$
$$= \sum_{s',r} p(s', r \mid s, a)\Big[r + \gamma \max_{a'} q_*(s', a')\Big],$$

Dynamic Programming

- Refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as an MDP
- Dynamic programming for MDP assumes full knowledge of the transition and reward models of the MDP

Dynamic Programming (Iterative Methods)

- Policy Optimization Algorithms
  - Policy Iteration
  - Value Iteration

Policy Iteration

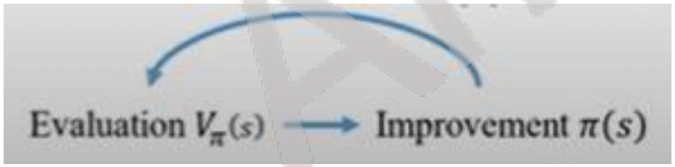
- Optimize the policy directly
- Start by choosing random or arbitrary policy
- Iteratively evaluate and improve the policy until convergence
- Two steps in Policy iteration
  - Policy Evaluation
    - to evaluate the value function for a given policy
  - Policy Improvement
    - to find a better value function and/or a better policy (eventually an optimal policy)

Policy Iteration

- Step 1: Policy Evaluation
  - Evaluate the policy π at state s to calculate the Q-value using Bellman Equation
  - $V_\pi(s) = r(s) + \gamma \sum P((s', r)|(s, \pi(s)))\ V(s')$

Policy Iteration

- Step 2: Policy Improvement
  - Searching for the action that maximized Q-value at each state
  - Update the policy by greedily performed search
  - $\pi(s) = argmax_a \sum_{s' \in S} p(s', r|s, a)\, V_\pi(s')$



Until converge to optimal ($V_\pi^*(s)$, and $\pi^*(s)$)

https://www.youtube.com/watch?v=nMlRhsgcTpY&list=PLZ_sI4f41TGvthD8dA7daahlbLV0yDW0w&index=3

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$; $V(\mathit{terminal}) \doteq 0$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \,|\, s, \pi(s)) \big[r + \gamma V(s')\big]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \,|\, s, a) \big[r + \gamma V(s')\big]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Value Iteration

- Compute optimal state value by iteratively updating $V_\pi(s)$
- Value iteration algorithm updates the state value in a single step
- $V_\pi(s) = r(s) + \gamma \max_a(\sum P(s', r|(s, a))\ V(s'))$

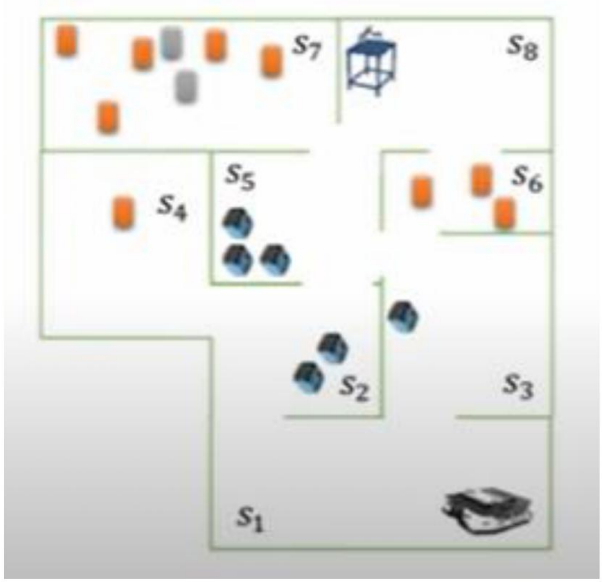
Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(\mathit{terminal}) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \arg\max_a \sum_{s',r} p(s', r \mid s, a)\big[r + \gamma V(s')\big]$

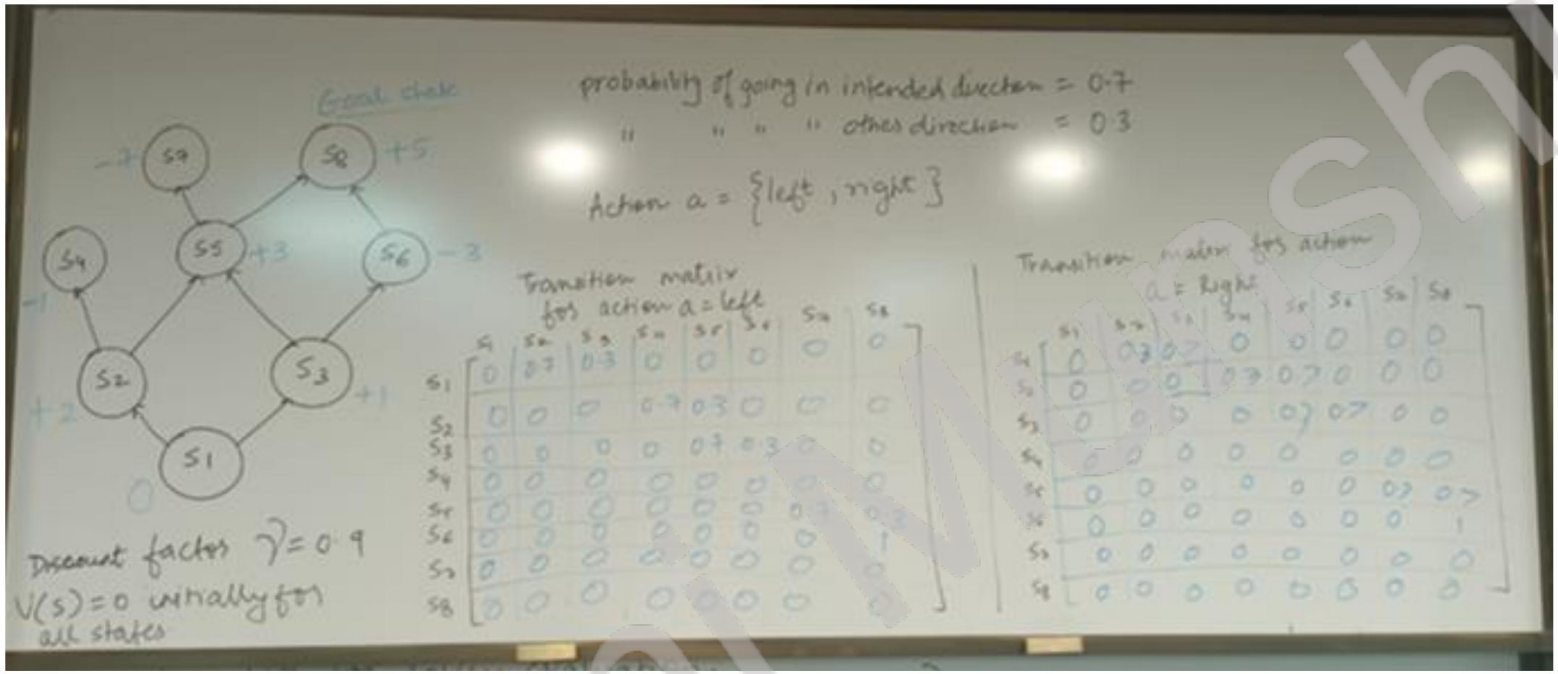
Example



s1	s2	s3	s4	s5	s6	s7	S8
R	R	R	-	R	R	-	-

- Consider the given state given maze problem
- The agent in state s1 has to reach the goal s8
- In every state there are positive rewards given by blue cubes or obstacles(negative rewards) represented by orange and grey rectangles
- Actions taken by agent= {left, right}
- Discount factor $\gamma = 0.9$
- Assumptions
  - Probability to go in the intended direction =0.7
  - Probability to go in the direction other than intended= 0.3
  - the initial policy as given in the adjacent table
  - Initial values of all the states $V(s)$=0
- Apply policy iteration algorithms for one iteration to obtain policy evaluation and policy improvement table
- Identify the change policy after the first iteration

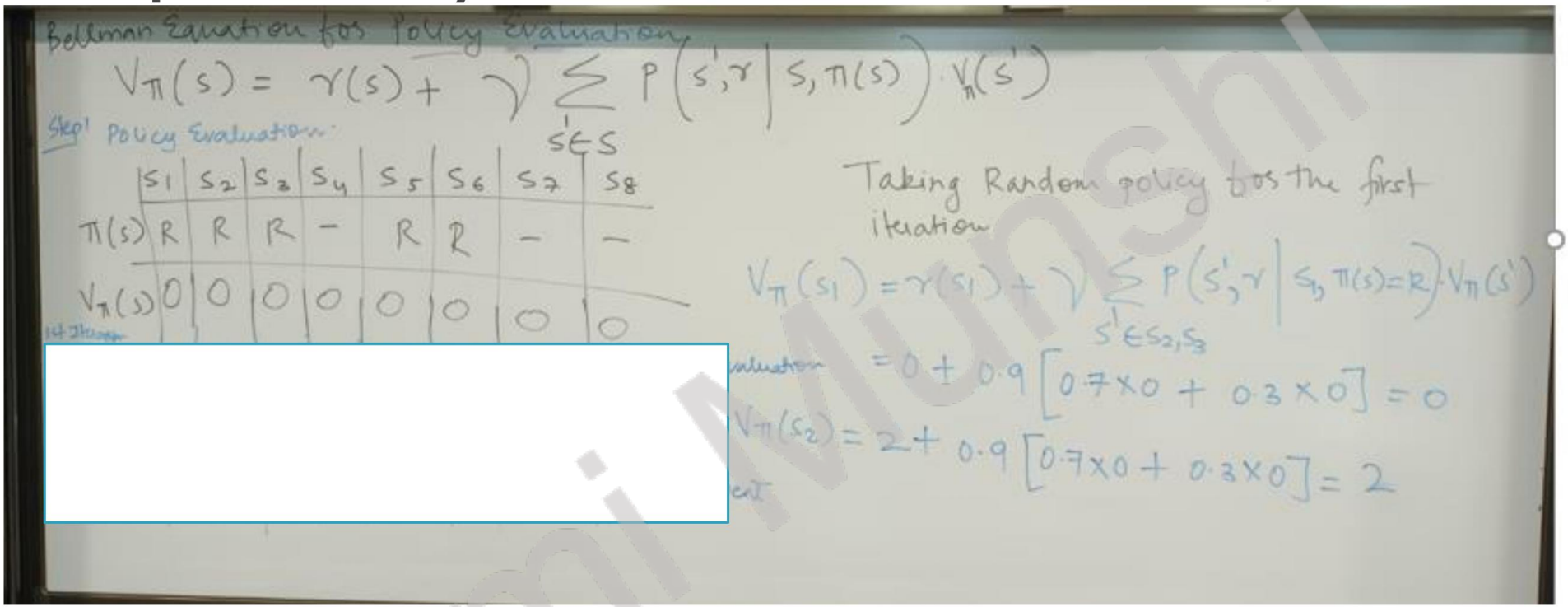
Solution



- Fig above shows the following-
  - The state diagram
  - All the assumptions
  - Transition matrix for action left and right

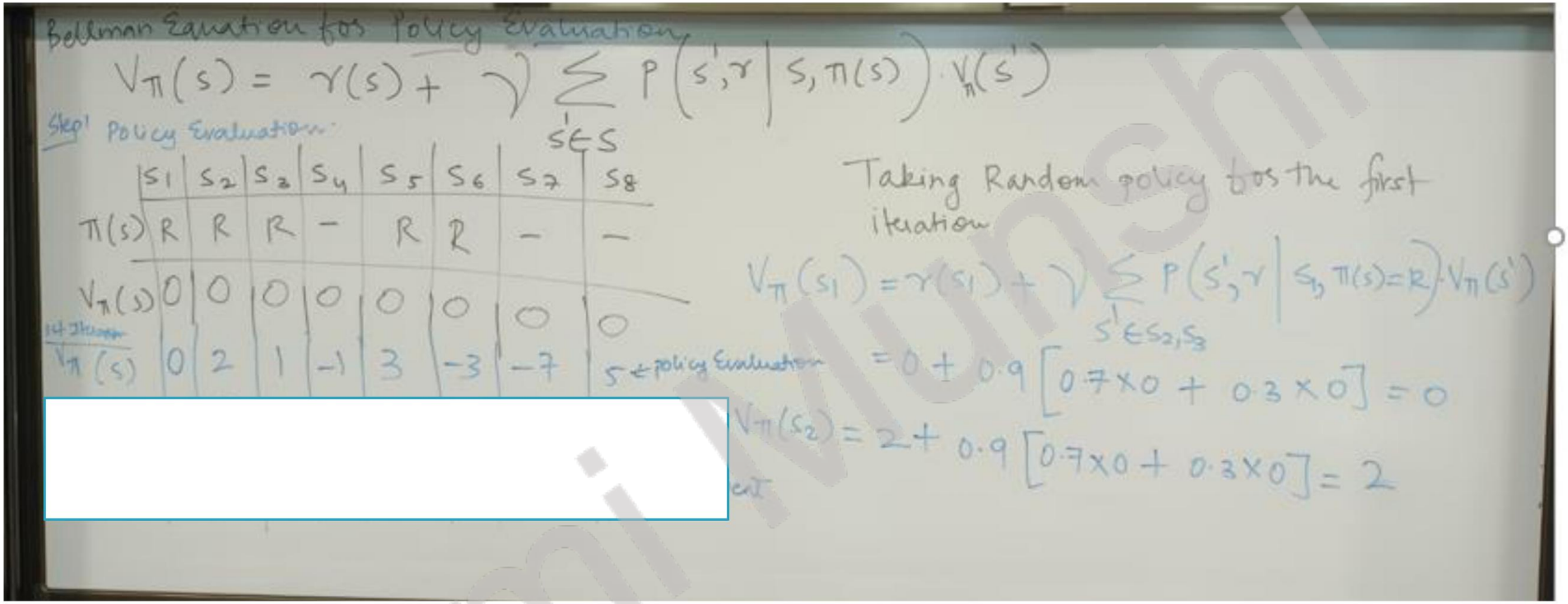
Two steps will be taken for each iteration
Step 1: Policy Evaluation
Step 2: Policy Improvement

Step1: Policy Evaluation



	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$\pi(s)$	R	R	R	-	R	R	-	-
$V_\pi(s)$	0	0	0	0	0	0	0	0

- Fig above shows the following-
  - Initial policy for each state
  - Initial value for each state
  - Bellman equation to obtain value for each state
  - Solution for value at state s1 and s2
  - Similarly, value for all the states can be obtained

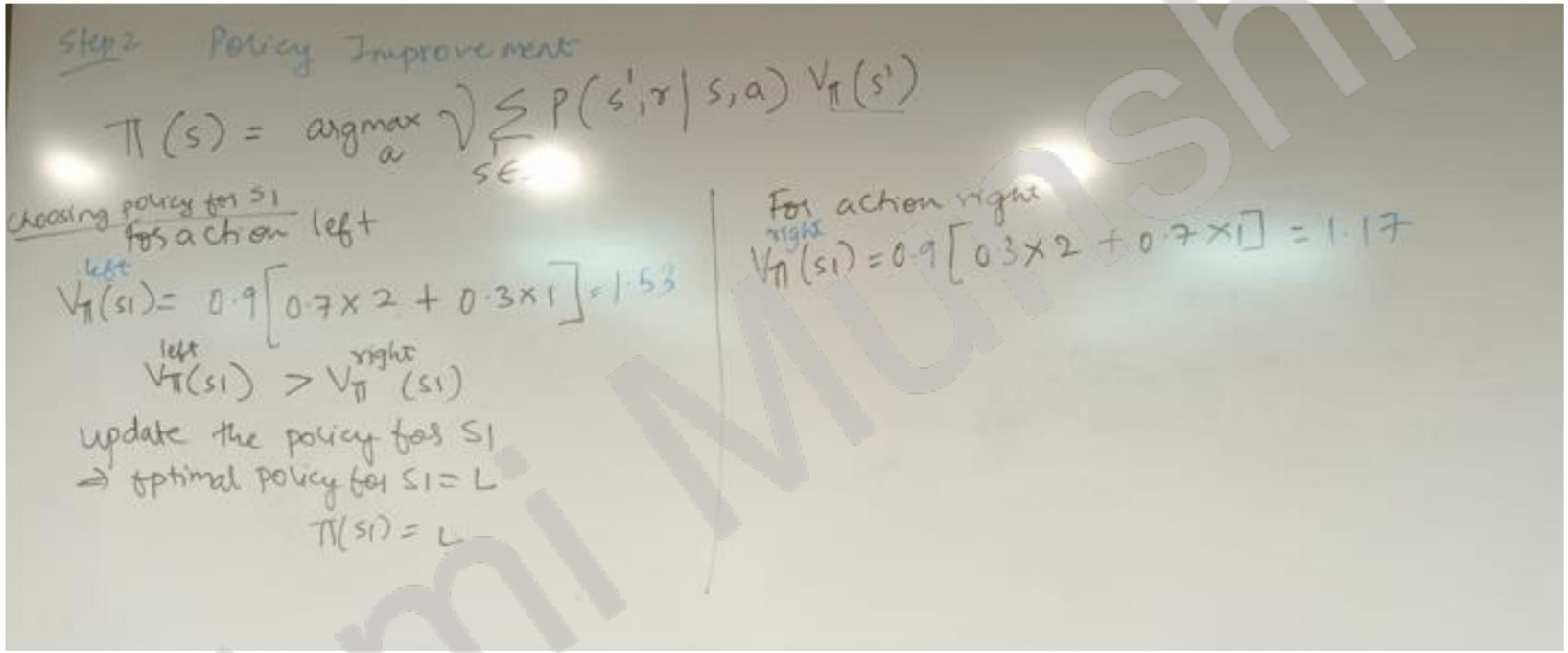


	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$\pi(s)$	R	R	R	–	R	R	–	–
$V_\pi(s)$	0	0	0	0	0	0	0	0
1st Iteration $V_\pi(s)$	0	2	1	-1	3	-3	-7	5 ← policy Evaluation

- Fig above shows the following-
  - Value after step 1 of policy evaluation

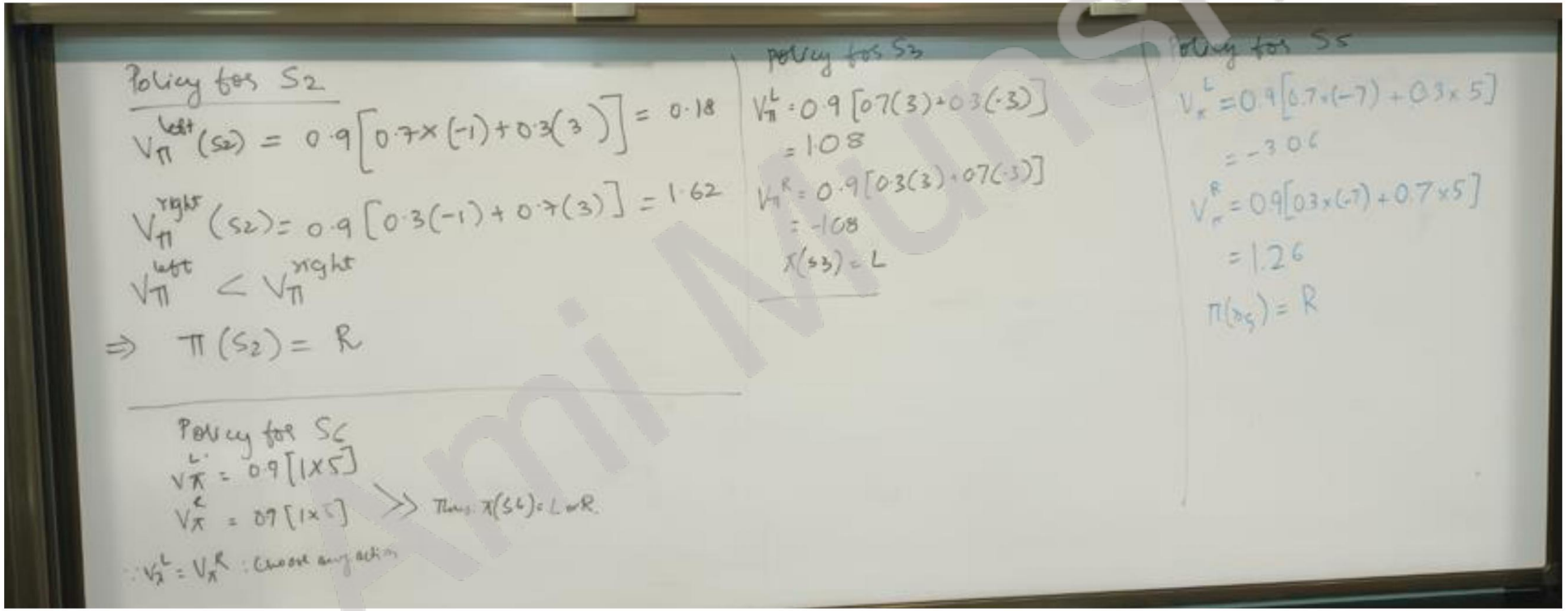
Note: If the initial values are zero, then the values after first iteration are the rewards for that state itself

Step2: Policy Improvement

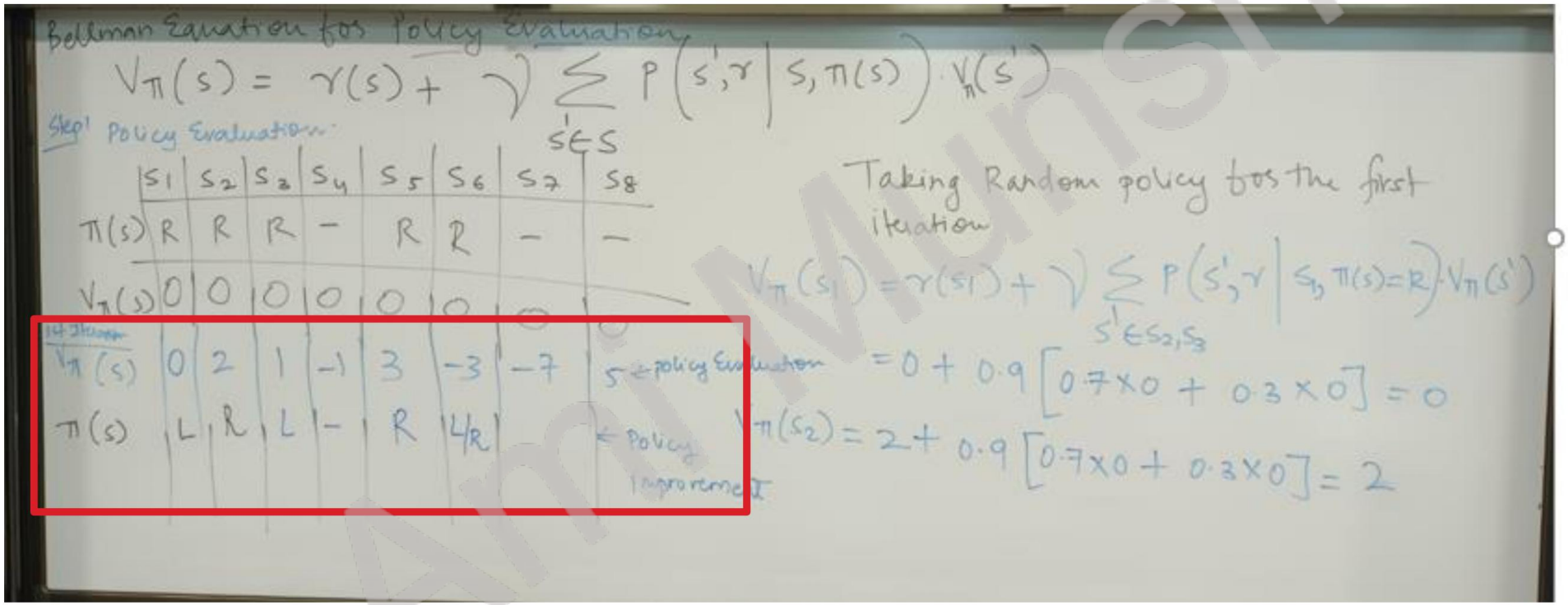


- According to the policy improvement expression, value if left action is taken or right action is taken is calculated for each state.
- Fig above shows for s1
- Which ever action gives the maximum value, that action is chosen
- Similar calculations are done for all the states which is shown in the next slide

Policy improvement calculations for states s2, s3, s5, s6



Updated values and policy after the first iteration



	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	
$\pi(s)$	R	R	R	-	R	R	-	-	
$V_\pi(s)$	0	0	0	0	0	0	0	0	
1st iteration									
$V_\pi(s)$	0	2	1	-1	3	-3	-7		← policy Evaluation
$\pi(s)$	L	R	L	-	R	L/R			← Policy Improvement

Similarly more iterations can be implemented till convergence is attained

Example

s1	s2	s3	s4	s5	s6	s7	S8
R	R	R	-	R	R	-	-

- Consider the given state given maze problem
- The agent in state s1 has to reach the goal s8
- In every state there are positive rewards given by blue cubes or obstacles(negative rewards) represented by orange and grey rectangles
- Actions taken by agent= {left, right}
- Discount factor $\gamma = 0.9$
- Assumptions
  - Probability to go in the intended direction =0.7
  - Probability to go in the direction other than intended= 0.3
  - the initial policy as given in the adjacent table
  - Initial values of all the states $V(s)$=0
- Apply value iteration algorithm
- Identify the change policy after the first iteration and the second iteration

Solution:

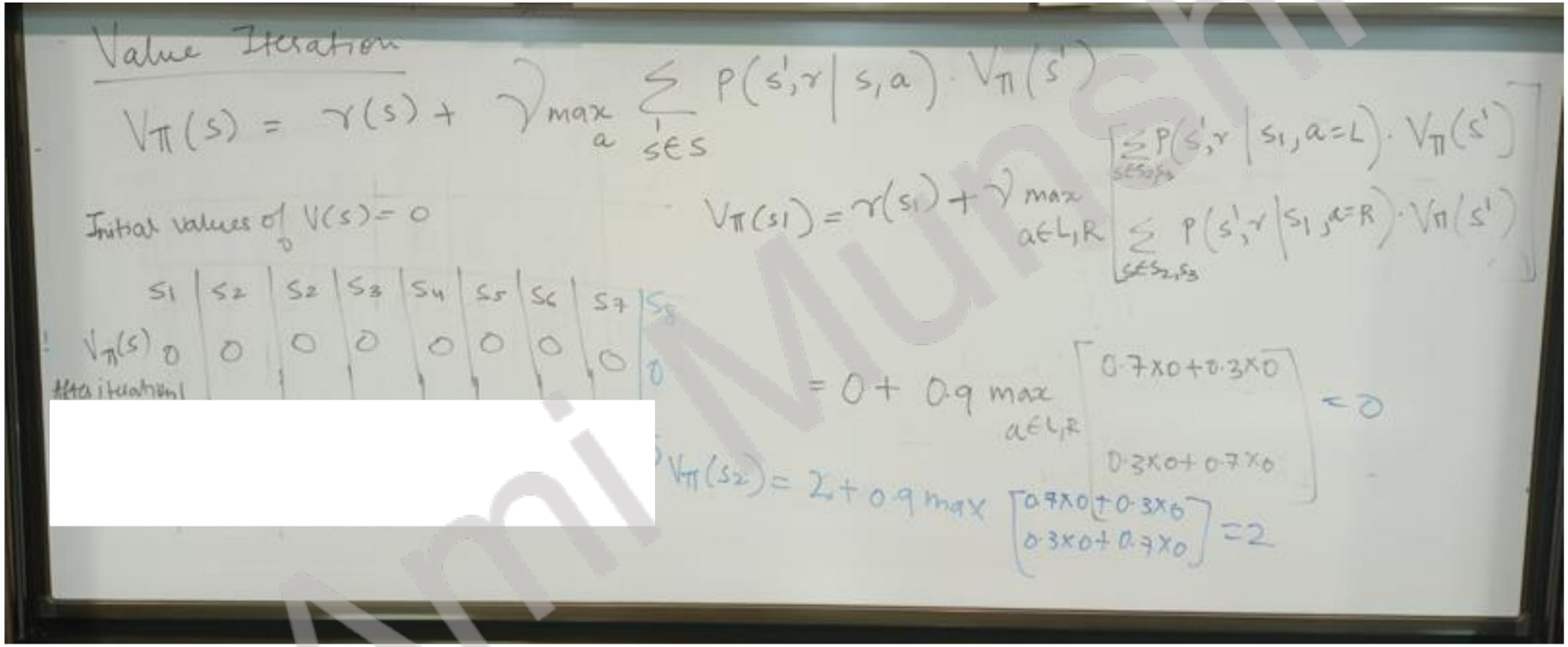
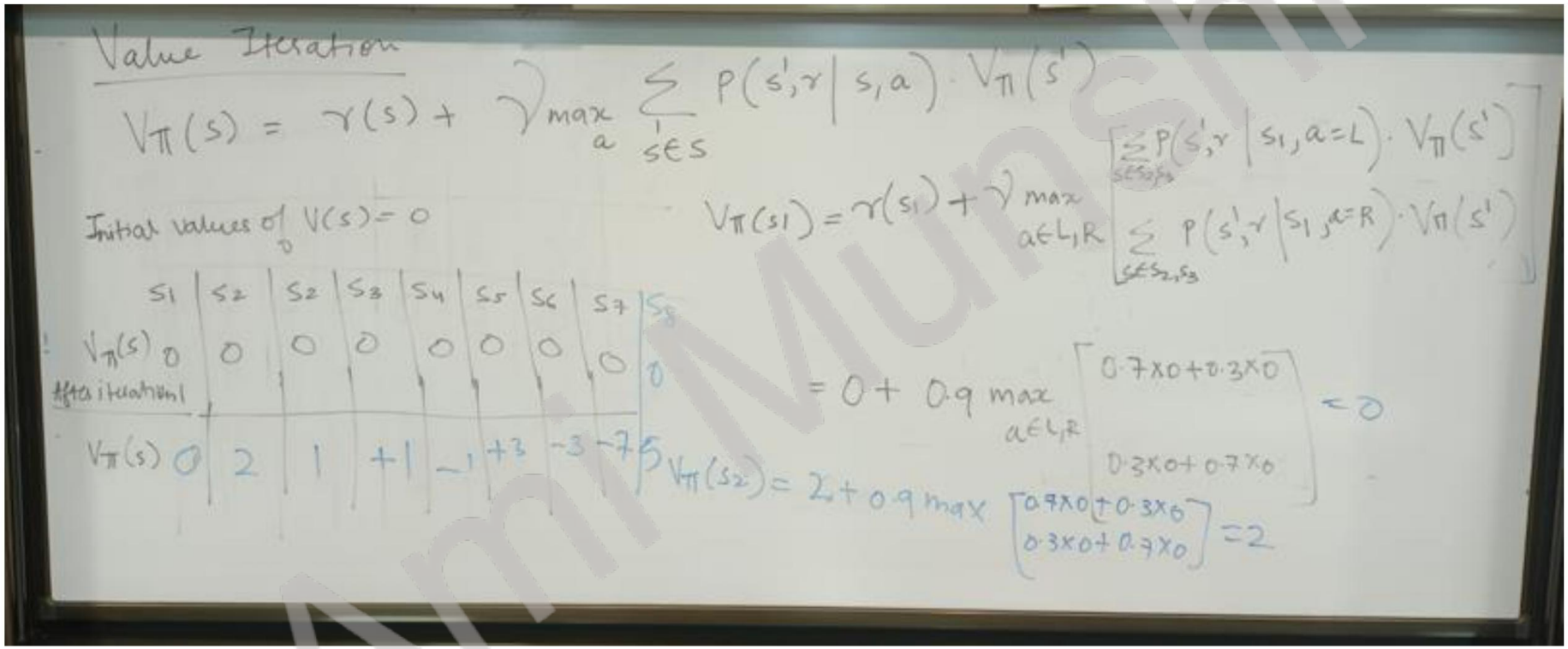


Figure above shows the calculation for values using Bellman equation for value iteration for state s1 and s2

Values after the first iteration of value iteration algorithm

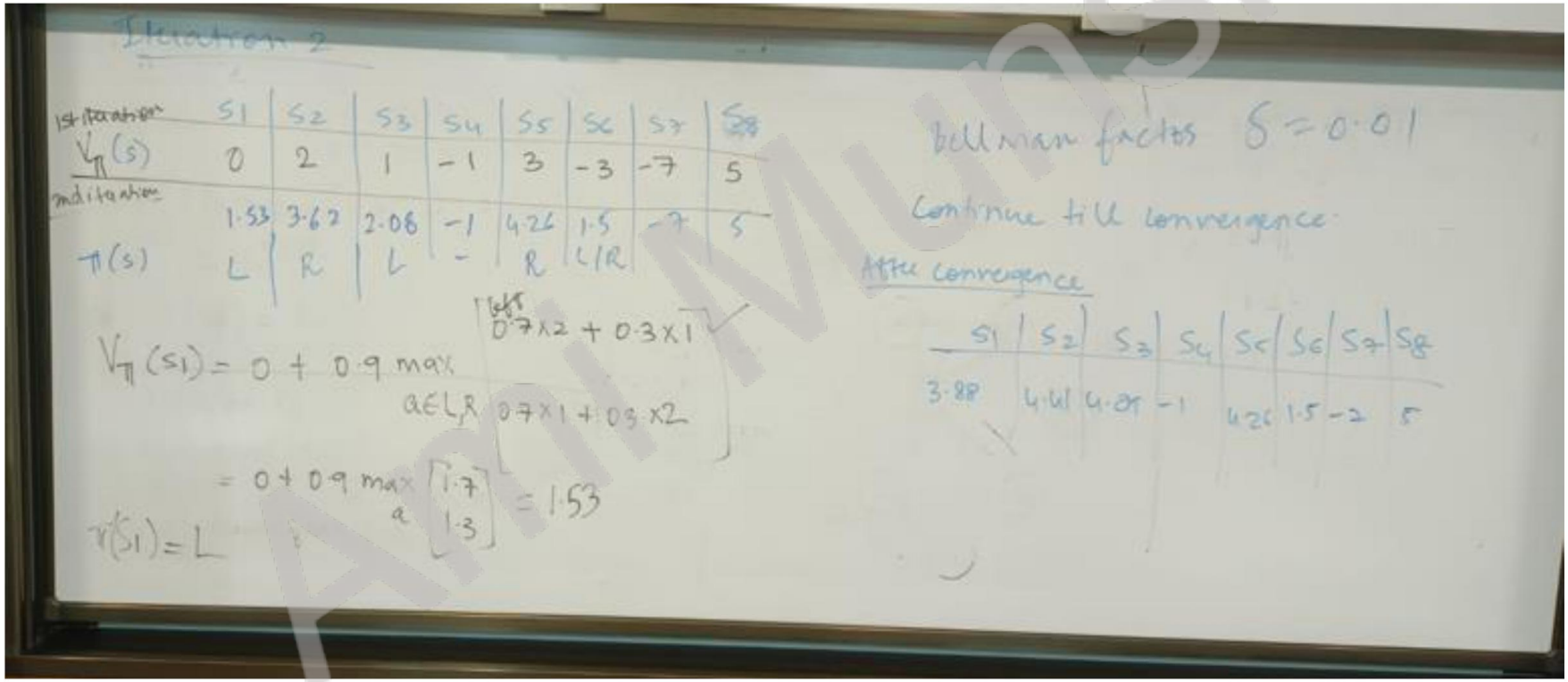


	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
$V_\pi(s)$	0	0	0	0	0	0	0	0
After iteration 1								
$V_\pi(s)$	0	2	1	+1	−1	+3	−3	−7

Similarly value for all the states is calculated

Result of values for all the states after the first iteration is shown above

Values and policy after the second iteration of value iteration algorithm



Example: Policy iteration algorithm for dice game

- At any time, the game is specified by two states: IN and END. When the game in the state IN, the agent can choose an action from the set $\mathcal{A} = \{STAY, QUIT\}$. If QUIT is selected, then the game will go to state END and the agent receive a reward \$10 with a probability of 1. If STAY is selected, the game will stay at IN state and the agent receive a reward \$4 with a probability of 2/3, or transition to state END and the agent receive a reward \$4 with a probability of 1/3.

Ref: Weidong Kuang, Fundamentals of RL, University of Texas, 2021

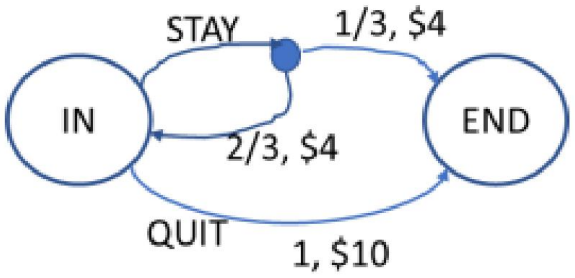


Fig.2 (a) an MDP

- The state transition diagram is drawn
- The value of state END is zero, i.e. $V(END) = 0$
- Assume that the probability of selecting STAY at state IN is 0.5

Demonstration of iterative steps for V(IN)

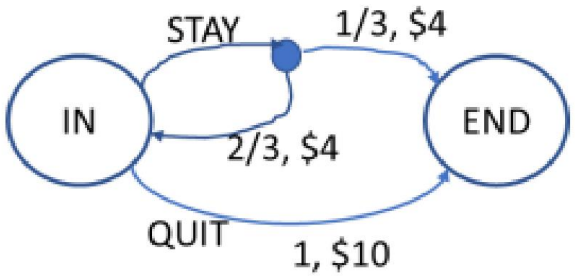


Fig.2 (a) an MDP

- k=0, $V_0(IN) = 0$
- k=1, $V_1(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_0(IN))\right) = 7$
- k=2, $V_2(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_1(IN))\right) = 9.33$
- k=3, $V_3(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_2(IN))\right) = 10.11$
- k=4, $V_4(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_3(IN))\right) = 10.37$
- k=5, $V_5(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_4(IN))\right) = 10.46$
- k=5, $V_6(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_5(IN))\right) = 10.49$
- k=6, $V_6(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_5(IN))\right) = 10.50$
- k=7, $V_7(IN) = 0.5 \times (10 + V(END)) + 0.5\left(\frac{1}{3}(4 + V(END)) + \frac{2}{3}(4 + V_6(IN))\right) = 10.50$

$V_6(IN) = V_7(IN) = 10.5$ This is same as example solved in Unit 3 pdf Page 100. See the snapshot of it below.

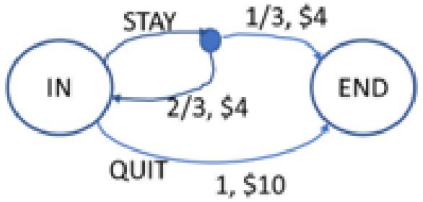


Fig.2 (a) an MDP

The Bellman equations for state-value function can be written as

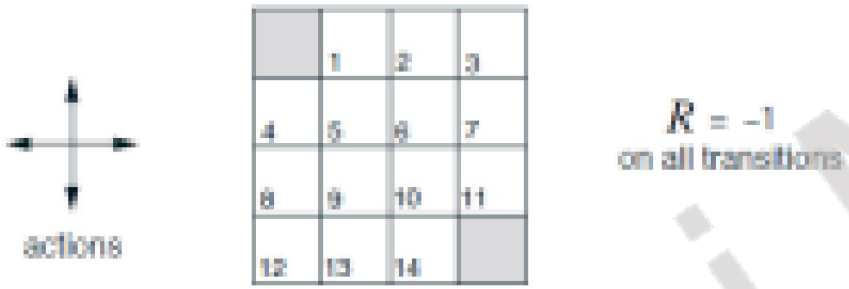
$$\begin{cases} v(END) = 0 \\ v(IN) = \frac{1}{2}(10 + v(END)) + \frac{1}{2}\left(\frac{1}{3}(4 + v(END)) + \frac{2}{3}(4 + v(IN))\right) \end{cases}$$

By solving the equations, we get the state-value function for a random policy,

$$\begin{cases} v(END) = 0 \\ v(IN) = 10.5 \end{cases}$$

Snapshot from example solved in Unit 3 pdf Page 100.

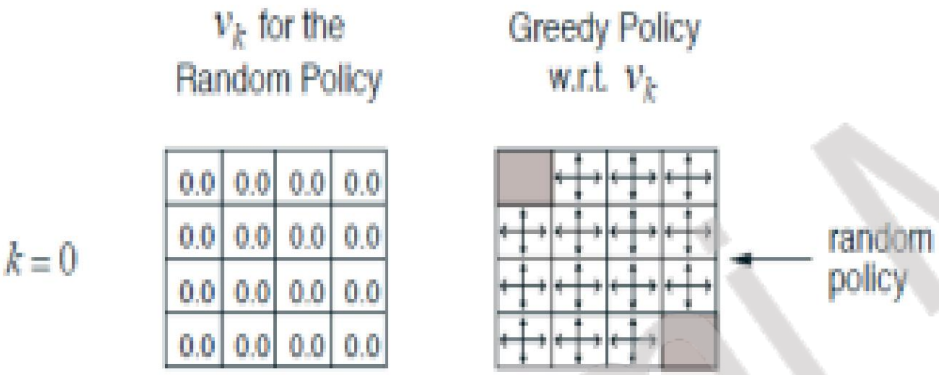
Example Grid World: Policy iteration algorithm



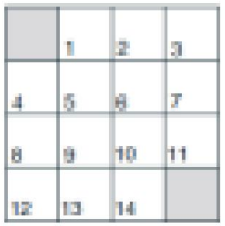
- Consider a 4x4 gridworld with a random policy as shown
- The agent moves on the grid randomly (p=0.25)
- It receives a reward of -1 for each move
- The episode ends at the up-left or bottom-right corners

Ref: Weidong Kuang, Fundamentals of RL, University of Texas, 2021

Example Grid World-Policy iteration algorithm



- Consider a 4x4 gridworld with a random policy as shown
- The agent moves on the grid randomly (p=0.25)
- It receives a reward of -1 for each move
- The episode ends at the up-left or bottom-right corners
- Step k =0, all V(s) are initialized to zero.



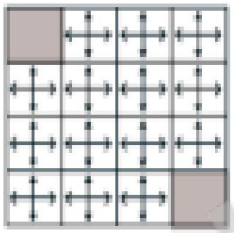
$R = -1$
on all transitions

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

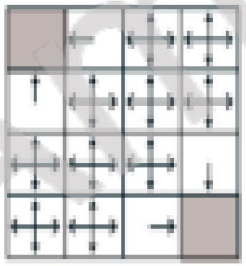
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



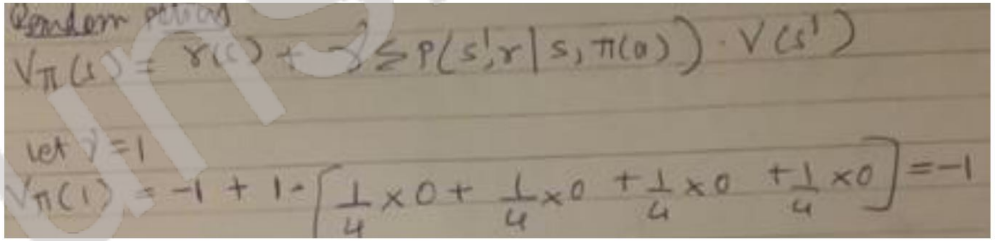
random policy

$k = 1$

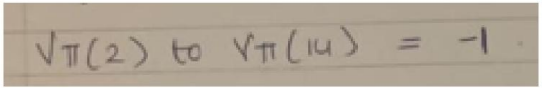
0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

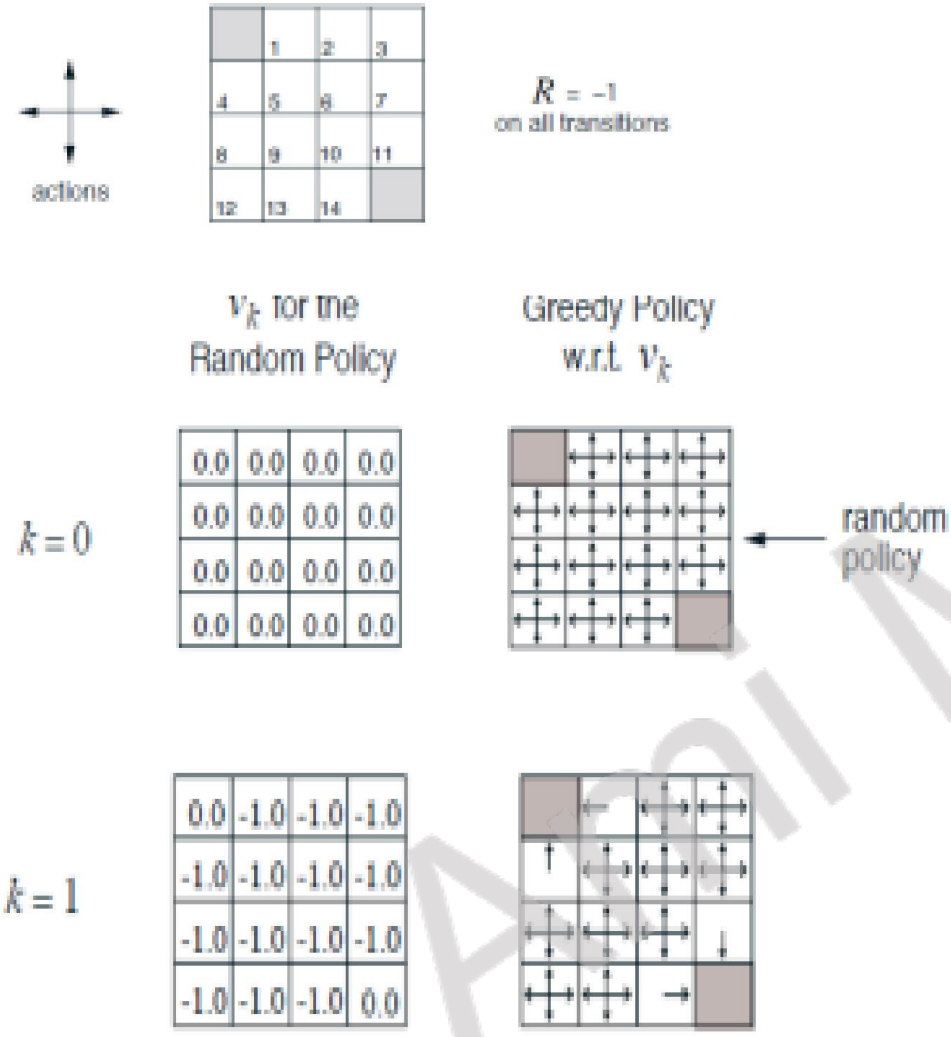


- Iteration1
- Step1 Policy Evaluation
  - Value for State1

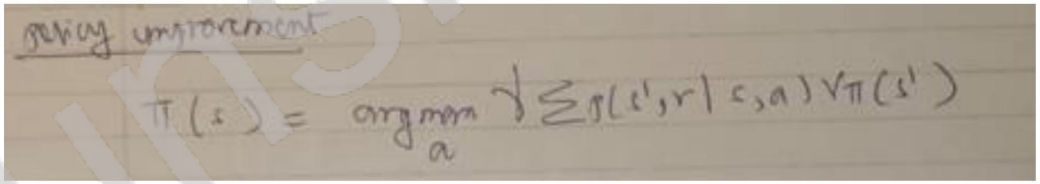


  - Similarly we have values for states 2 to 14

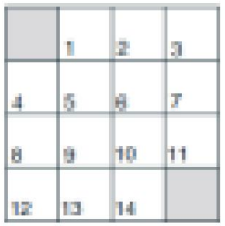




- Iteration1
- Step2 Policy Improvement



- To identify policy for state s1
  - We calculate values for all the actions (U,D,L,R)
  - We will choose the action which have maximum value
- Calculations of values for all the actions are given below



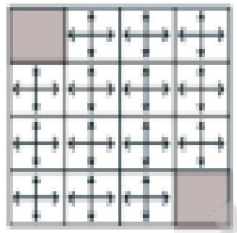
$R = -1$
on all transitions

v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

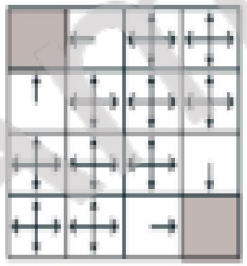
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0



random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0



For action U

$$V_\pi^U(s_1) = 1[0.25\times(-1) + \frac{0.25}{4}(0) + \frac{0.25}{4}(-1) + \frac{0.25}{4}(-1) + \frac{0.25}{4}(-1)]$$

$$= [-0.25 + \frac{0.75}{4}(0-1-1-1)] = -0.8125$$

For action R

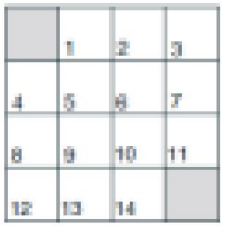
$$V_\pi^R(s_1) = 1[0.25\times(0) + \frac{0.75}{4}[0-1-1-1]] = -0.5625$$

For action L

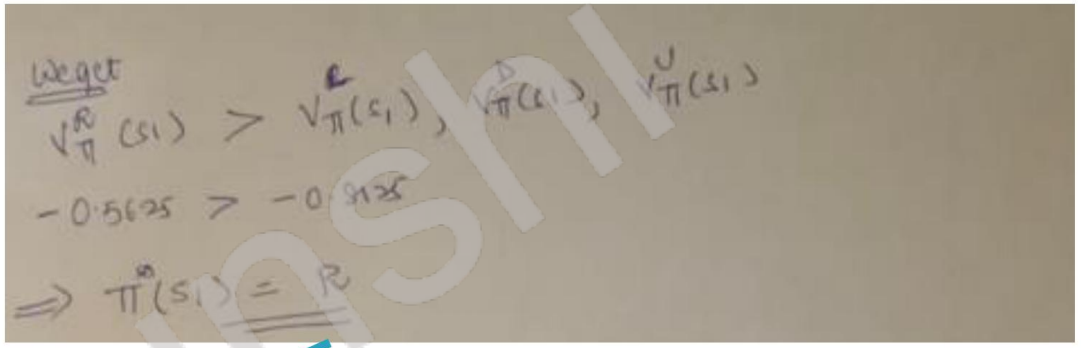
$$V_\pi^L(s_1) = 1[0.25(-1) + \frac{0.75}{4}(0-1-1-1)] = -0.8125$$

For action D

$$V_\pi^D(s_1) = 1[0.25(-1) + \frac{0.75}{4}(0-1-1-1)] = -0.8125$$



$R = -1$
on all transitions

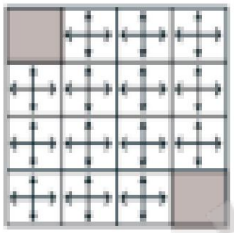


v_k for the Random Policy

Greedy Policy w.r.t. v_k

$k = 0$

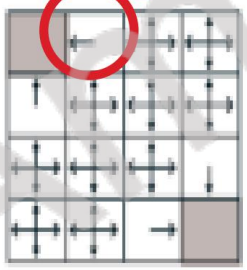
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0

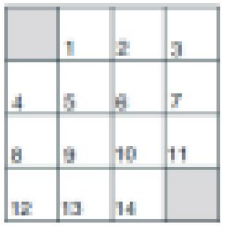


random policy

$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

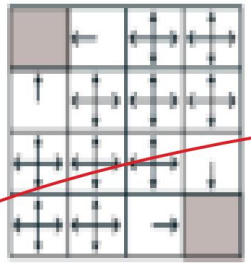




$R = -1$
on all transitions

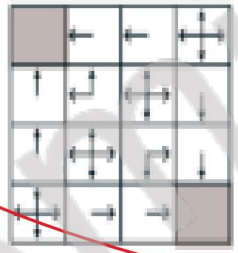
$k = 1$

0.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	0.0

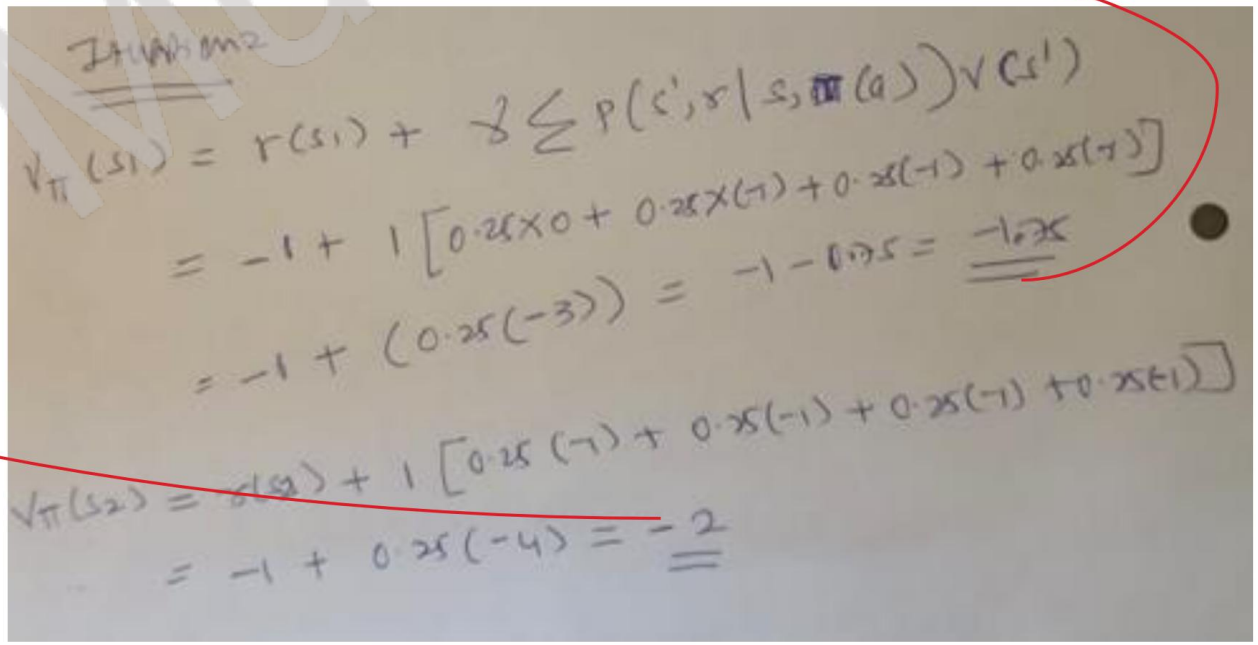


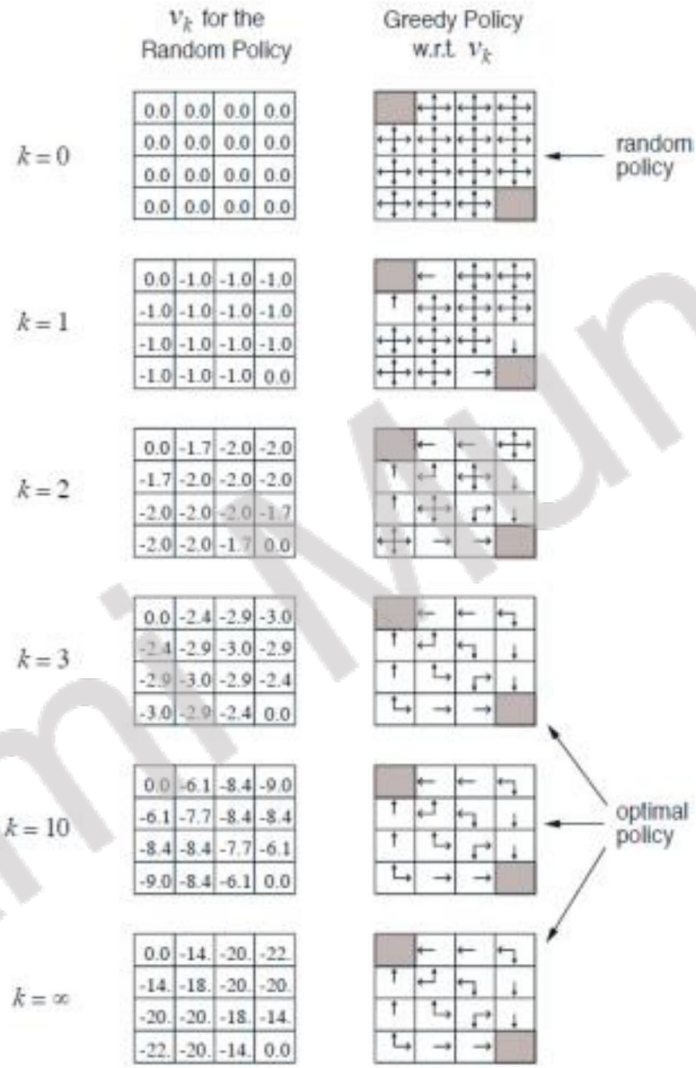
$k = 2$

0.0	-1.7	-2.0	-2.0
-1.7	-2.0	-2.0	-2.0
-2.0	-2.0	-2.0	-1.7
-2.0	-2.0	-1.7	0.0



- Iteration 2
- Step1 Policy Evaluation
  - Given below is calculation of values for state 1 and 2




v_k for the Random Policy
Greedy Policy w.r.t. v_k
$k = 0$
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
0.0 0.0 0.0 0.0
random policy
$k = 1$
0.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 -1.0
-1.0 -1.0 -1.0 0.0
$k = 2$
0.0 -1.7 -2.0 -2.0
-1.7 -2.0 -2.0 -2.0
-2.0 -2.0 -2.0 -1.7
-2.0 -2.0 -1.7 0.0
$k = 3$
0.0 -2.4 -2.9 -3.0
-2.4 -2.9 -3.0 -2.9
-2.9 -3.0 -2.9 -2.4
-3.0 -2.9 -2.4 0.0
$k = 10$
0.0 -6.1 -8.4 -9.0
-6.1 -7.7 -8.4 -8.4
-8.4 -8.4 -7.7 -6.1
-9.0 -8.4 -6.1 0.0
optimal policy
$k = \infty$
0.0 -14. -20. -22.
-14. -18. -20. -20.
-20. -20. -18. -14.
-22. -20. -14. 0.0

Homework- DIY

- Go through the policy evaluation for volcano example in the following book
- Weidong Kuang, Fundamentals of RL, University of Texas, 2021

Homework

- Compare Policy Iteration an Value Iteration Algorithms
- Significance or Efficiency of dynamic programming (Sutton chap4)