Module 1
CSE3011 Reinforcement Learning
Credit Structure : 2-2-3

Slides Prepared by Dr J Alamelu Mangai

PRESIDENCY UNIVERSITY
GAIN MORE KNOWLEDGE REACH GREATER HEIGHTS
Private University Estd. in Karnataka State by Act No. 41 of 2013
PRESIDENCY GROUP OVER 40 YEARS OF ACADEMIC WISDOM

Module 1 : Introduction to RL

Topics : Elements of RL - Agent, environment Interface, Goals and rewards, RL platforms, Applications of RL, Markov decision process (MDP), RL environment as a MDP, Maths essentials of RL, Policy and its types, episodic and continuous tasks, return and discount factor, fundamental functions of RL – value and Q functions, model-based and model-free learning, types of RL environments, Solving MDP using Bellman Equation, Algorithms for optimal policy using Dynamic Programming -Value iteration and policy iteration, Example : Frozen Lake problem, Limitations and Scope.

Slides Prepared by Dr J Alamelu Mangai

Introduction to RL

- Reinforcement Learning(RL) is one of the areas of Machine Learning(ML).
- It is a feedback-based Machine learning technique in which an agent learns to behave in an environment by performing the actions and seeing the results of actions.
- For each good action, the agent gets positive feedback, and for each bad action, the agent gets negative feedback or penalty.
- The agent interacts with the environment and explores it by itself. The primary goal of an agent in reinforcement learning is to improve the performance by getting the maximum positive rewards.
- It is one of the most active research areas in AI.
- It has evolved and capable of building a recommendation system to self-driving cars.
- Reason for this evolution is deep RL, a combination of DL and RL.
- https://neptune.ai/blog/reinforcement-learning-applications

Introduction to RL..

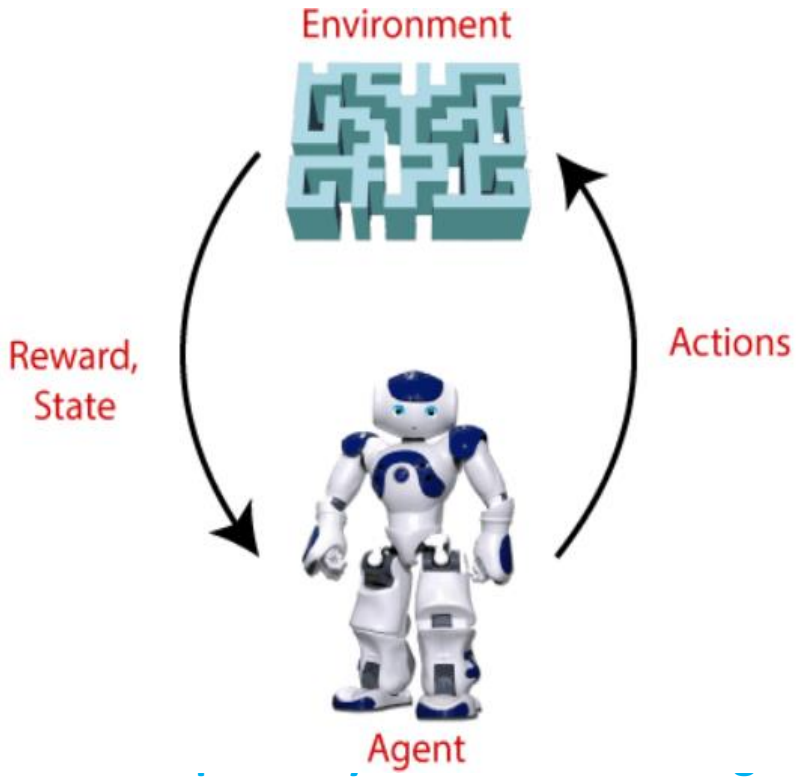
- In Reinforcement Learning, the agent learns automatically using feedbacks without any labeled data, unlike supervised learning.
- Since there is no labeled data, the agent is bound to learn by its experience only.
- RL solves a specific type of problem where decision making is sequential, and the goal is long-term, such as **game-playing, robotics**, etc.

RL agents

- The goal of reinforcement learning is to train an agent to complete a task within an uncertain environment.
- At each time interval, the agent receives observations and a reward from the environment and sends an action to the environment.
- The reward is a measure of how successful the previous action (taken from the previous state) was with respect to completing the task goal.

Example

- **Goal of an AI agent has to find the diamond present within a maze environment**
- The agent interacts with the environment by performing some actions, and based on those actions, the state of the agent gets changed, and it also receives a reward or penalty as feedback for its actions.
- The agent continues doing these three things (**take action, change state/remain in the same state, and get feedback**), and by doing these actions, he learns and explores the environment.
- The agent learns that what actions lead to positive feedback or rewards and what actions lead to negative feedback penalty. As a positive reward, the agent gets a positive point, and as a penalty, it gets a negative point.



Elements of RL

- LO1: Define the key elements of RL
- LO2: Explain the steps involved in a typical RL algorithm
- LO3: Differentiate the three machine learning paradigms namely, supervised, unsupervised and reinforcement learning.

Elements of RL

1. Agent:

- It is a software that learns to make intelligent decisions.
- In an RL setting, an agent is a learner.
- Ex1: a chess-player is an agent, the player learns to make the best moves (decisions) to win the game.
- Ex2 : Mario in a Super Mario Bros Video Game

Elements of RL

2. **Environment :**

- It is the world of the agent, within which the agent stays, takes actions and interacts.
- Ex1: chess-board in a chess game.
- The chess player(agent) stays in the chess board to learn how to play the game.

Elements of RL

3. **State and Action:**

- In an RL setup, the environment has many positions where the agent can be in.
- Each such position is a state.
- A state is denoted by **s**
- Ex: in a chess-board environment, each position is a state.

Elements of RL

3. **State and Action:**

- The agent interacts with the environment and moves from one state to another state by performing an action.
- Ex: In a chess-game environment, the action is the move performed by the player(agent).
- An action is denoted by **a**

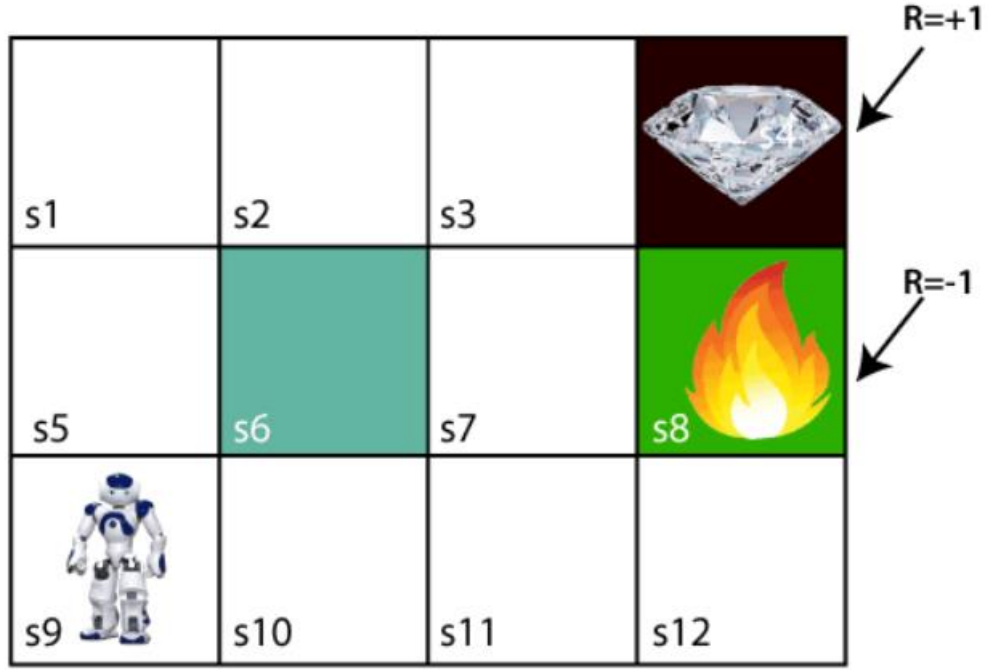
Elements of RL

4. Reward :

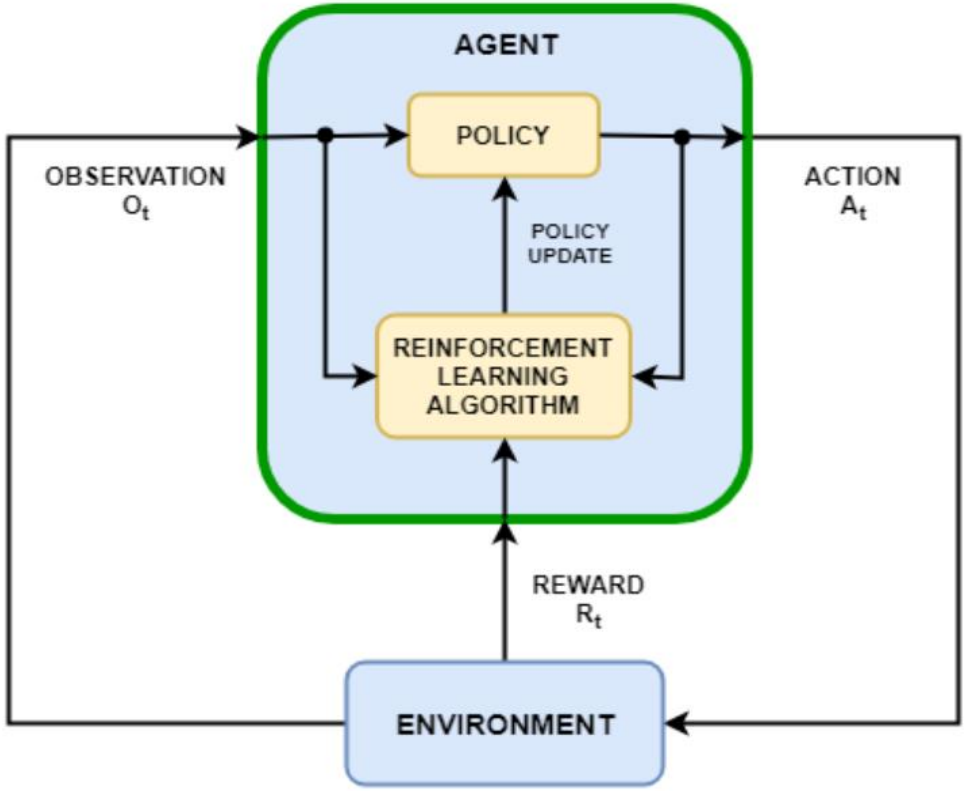
- The agent interacts with the environment by performing an action and moves from one state to another.
- Based on the action the agent receives a reward.
- Reward is a numerical value, ex: +1 for a good action, -1 for a bad action.
- Ex: in a chess-game, **good action** – the agent's move which takes one of the opponent's chess piece; **bad action** – the agent's move loosing one chess piece to the opponent

Basic idea of RL

- To understand the working process of the RL, we need to consider two main things:
- **Environment:** It can be anything such as a room, maze, football ground, etc.
- **Agent:** An intelligent agent such as AI robot.
- Let's take an example of a maze environment that the goal of the agent is to explore and find the path to the diamond in few steps.
- **States** : S1 to S12, where S6 is a wall, S8 is a fire pit and S4 has diamond.
- **Actions** : move left, right, up and down



A typical RL setup



- The agent has two components : policy and the RL algorithm.
- **Policy :**
  - It is a mapping from the current environment's observation to a probability distribution of the actions to be taken.
  - Within an agent, the policy is implemented by a function approximator with tunable parameters and a specific approximation model, such as a deep neural network.
- **RL algorithm:**
  - The learning algorithm continuously updates the policy parameters based on the actions, observations, and rewards.
  - The goal of the learning algorithm is to find an optimal policy that maximizes the expected cumulative long-term reward received during the task.**Slides Prepared by Dr J Alamelu Mangai**

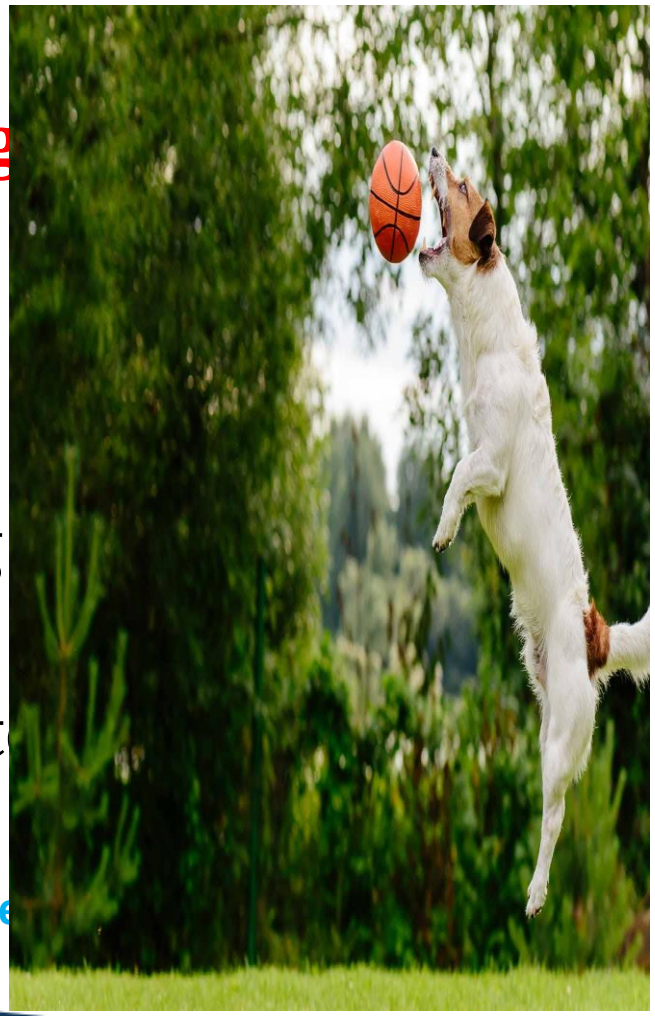
How RL differs from other ML paradigms?

Task : train a dog to catch a ball

- **Difference between RL and Supervised learning**
- Supervised learning, we train the dog explicitly

with training data : turn left, go right, move forward seven steps, catch the ball and so on.

- In RL, we simply throw the ball, every time the dog catches the ball, we give it a cookie(reward).
- So the dog will learn to catch the ball while trying t maximize the cookies(rewards) it can get.



Slides Pre

How RL differs from other ML paradigms?

- **Difference between RL and unsupervised learning?**
- **Task : Movie recommendation system-** recommend a new movie to the user
- Unsupervised learning: the model will recommend a new movie based on the similar movies the user has viewed before.
- In RL, each time the user watches a movie, the agent receives feedback from the user.
- Feedbacks are rewards(ratings given by the user to this movie, time spent watching the movie, etc).
- Based on these rewards, the RL agent will understand the movie preference of the user, then suggests new movies accordingly.

How RL differs from other ML paradigms?

- Hence, the supervised and unsupervised the models learn from the training data set.
- In RL, the agent learns by continuously interacting with the environment.
- Hence entire RL is about the interaction between the agent and the environment.

A typical RL algorithm

- The steps involved in a typical RL algorithm are:

1. First, the agent interacts with the environment by performing an action.
2. By performing an action, the agent moves from one state to another.
3. Then the agent will receive a reward based on the action it performed.
4. Based on the reward, the agent will understand whether the action is good or bad.
5. If the action was good, that is, if the agent received a positive reward, then the agent will prefer performing that action, else the agent will try performing other actions in search of a positive reward.

The goal of the agent is to maximize the reward it gets. If the agent receives a good reward, then it means it has performed a good action. If the agent performs a good action, then it implies that it can win the game. Thus, the agent learns to win the game by maximizing the reward

RL agent in the grid world environment

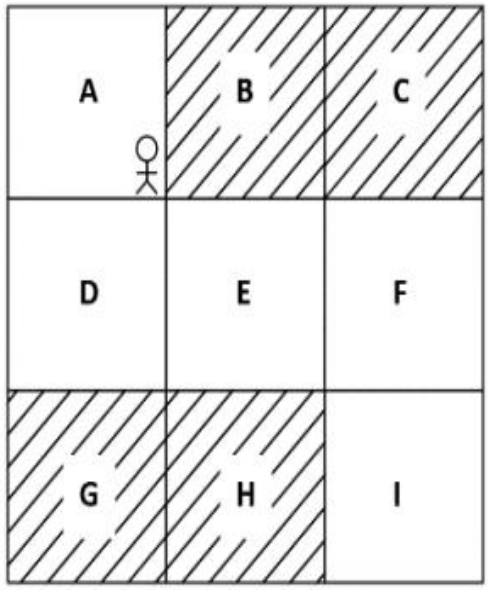


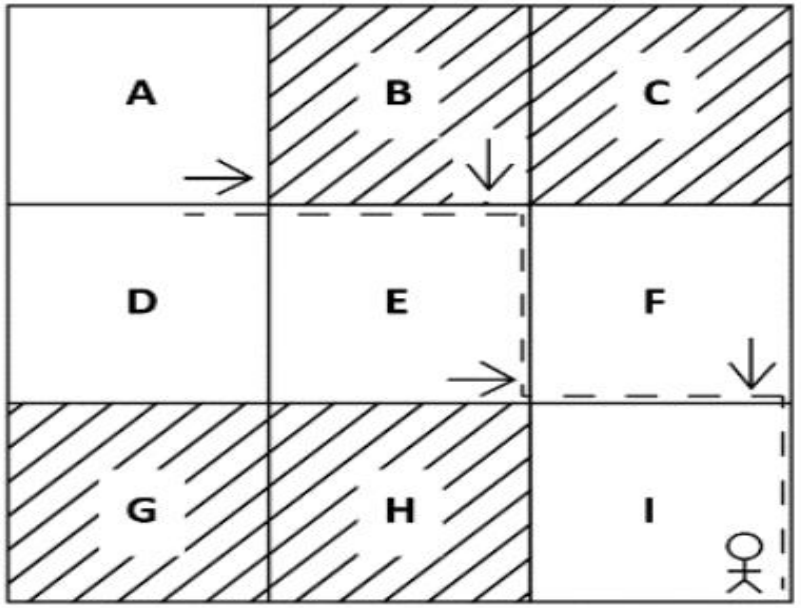
Figure 1.4: Grid world environment

- **Environment** : Grid World environment
- **States** : A,B,C,D,E,F,G,H and I. Shaded states are hole states.
- **Goal of the agent** : reach state I from state A.
- **Actions** : move up, down, left and right.
- Every time the agent reaches one of the shaded states it receives a **negative reward**(-1).
- Every time the agent reaches one of the unshaded states it receives a **positive reward**(+1).
- First time when the agent interacts with the envt (first iteration), it performs a random action in each state, and mostly ends up with negative rewards.
- But, over a series of iterations, it learns to perform the correct action in each state, based on the rewards it has obtained in that state in the previous iterations and hence reaches the goal.

Slides Prepared by Dr J Alamelu Mangai

RL agent in the Grid World Environment

- Iteration 1:



igure 1.5: Actions taken by the agent in iteration 1

RL agent in the Grid World Environment

- Iteration 2:

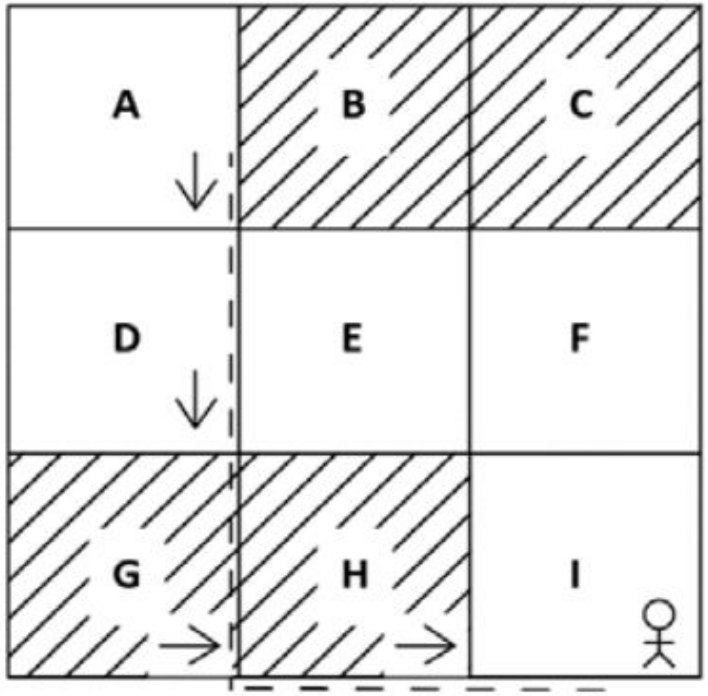


Figure 1.6: Actions taken by the agent in iteration 2

RL agent in the Grid World Environment

- Iteration 3:

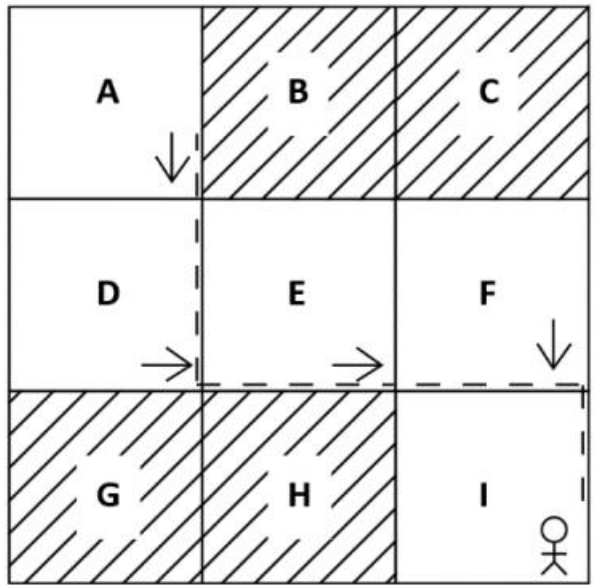


Figure 1.7: Actions taken by the agent in iteration 3

RL agent in the Grid World Environment

- Result of Iteration 3, the agent reaches the goal state without reaching the shaded states.
- The agent has successfully learnt to reach the goal state I from state A, without visiting the shaded states based on the rewards.
- The goal of the agent is to maximize the rewards and ultimately achieve the goal
- Each iteration known as an episode in RL terms.

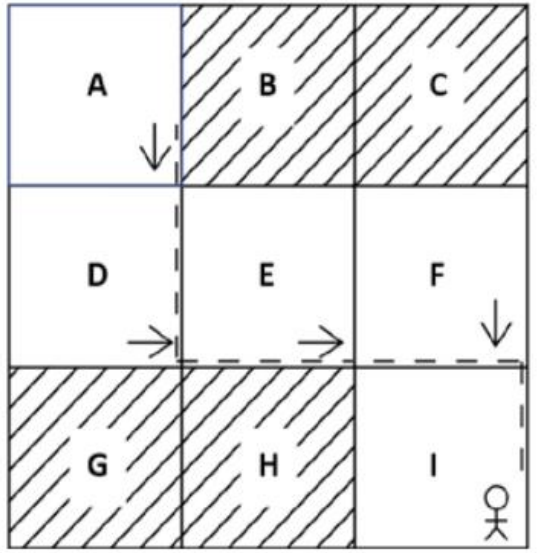


Figure 1.8: The agent reaches the goal state without visiting the shaded states

Types of RL environments

- LO1: List the types of RL environments
- LO2: Differentiate the different types of RL environments
- LO3: Explain the different types of RL environments with an example

Slides Prepared by Dr J Alamelu Mangai

Types of RL environments

- **Deterministic environment** : It is certain that, when an agent in state s, performs an action a, then it always reaches state s'.

Ex: **Chess –** there would be only a few possible moves for a coin at the current state and these moves can be determined.

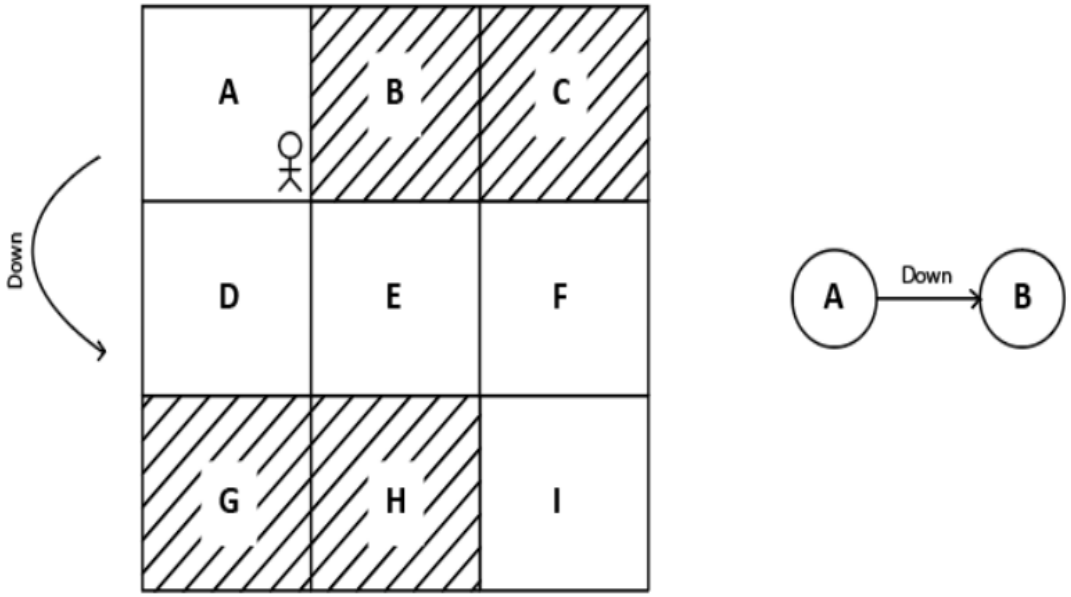


Figure 1.30: Deterministic environment

Types of RL environments

- **Stochastic environment** : When an agent in state s, performs an action a, then we cannot say that, it always reaches state s'.
- We cannot determine the outcome of the action in the current state
- This is due to the randomness in the stochastic environment.

Ex1: **Self-Driving Cars-** the actions of a self-driving car are not unique, it varies time to time.

Ex2: The radio station is a stochastic environment where the listener is not aware about the next song.

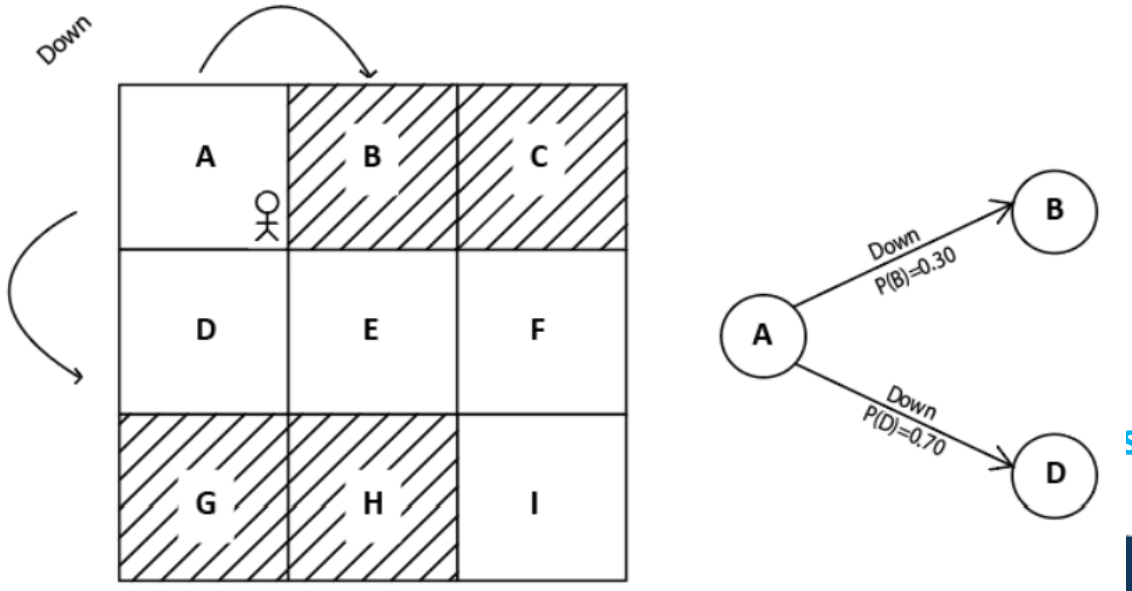


Figure 1.31: Stochastic environment

Types of RL environments

3. **Discrete Environment:** The action space of the environment is discrete.

Ex: action space of the grid world environment is [up, down, left, right].

4. **Continuous environment:** The action space of the environment is continuous

Ex1: To train an agent to drive a car, then the action space will involve multiple actions like [changing the car's speed, the no. of degrees to rotate the wheel, etc..]

Ex2: In a basketball game, the position of players (**Environment**) keeps changing continuously and hitting (**Action**) the ball towards the basket can have different angles and speed so infinite possibilities.

Types of RL environments

5. **Episodic/Non-Sequential Environment**: the agent's current action will not affect the future actions

Ex: A support bot (agent) answering to a question and then answering to another question and so on. So each question-answer is a single episode.

6. **Non-Episodic/Sequential Envt**: the agent's current action will affect its future actions.

Ex: a chess-board is a sequential environment since the agent's current action will affect its future actions in a chess match.

7. Single and Multi-agent environments:

- **Single agent environment** where an environment is explored by a single agent. All actions are performed by a single agent in the environment.

- **Real-life Example**: Playing tennis against the ball is a single agent environment where there is only one player.

- If two or more agents are taking actions in the environment, it is known as a multi-agent environment.

- **Real-life Example**: Playing a soccer match is a multi-agent environment.

Markov decision process (MDP)

- LO1: State the Markov property, Markov chain, Markov reward process and the Markov decision process.
- LO2: Explain the Markov chain with a suitable example
- LO3: Recognize the mathematical essentials of RL
- LO4: Interpret the grid world RL environment as a MDP

Slides Prepared by Dr J Alamelu Mangai

Markov Decision Process(MDP)

- MDP is mainly used to study optimization problems via dynamic programming.
- **A Markov decision process (MDP) refers to a stochastic decision-making process that uses a mathematical framework to model the decision-making of a dynamic system.**
- **It is used in scenarios where the results are either random or controlled by a decision maker, which makes sequential decisions over time.**
- **MDPs evaluate which actions the decision maker should take considering the current state and environment of the system.**
- Almost all RL problems can be modelled as a MDP.
- In artificial intelligence, MDPs model sequential decision-making scenarios with probabilistic dynamics.
- They are used to design intelligent machines or agents that need to function longer in an environment where actions can yield uncertain results.

Slides Prepared by Dr J Alamelu Mangai

MDP

- MDP uses two entities namely Markov property and Markov chain.
- **Markov property** : **says the future depends only on the present and not on the past.**
- **Markov chain/Markov process** : has a sequence of states that strictly obey the Markov property.
- Markov chain is a probabilistic model that solely depends on the current state to predict the next state and not the previous states.
- **The future is conditionally independent of the past.**
- Ex : if the current state of weather is cloudy, we can predict the next state to be rainy.
- We made this prediction only based on the current state(cloudy) and not on the previous states which might be sunny, windy, etc.
- **Markov property is not valid for all processes.**
- Ex: while throwing a dice(the next state), doesn't depend on the previous number that showed up on the dice(the current state)

MDP

- MDP uses states and state transition probabilities.
- $P(s'|s)$ is the probability of moving from current state s to next state s'
- State transition probabilities can be represented using Markov table, state diagram or transition matrix.

Current State	Next State	Transition Probability
Cloudy	Rainy	0.7
Cloudy	Windy	0.3
Rainy	Rainy	0.8
Rainy	Cloudy	0.2
Windy	Rainy	1.0

Table 1.1: An example of a Markov table

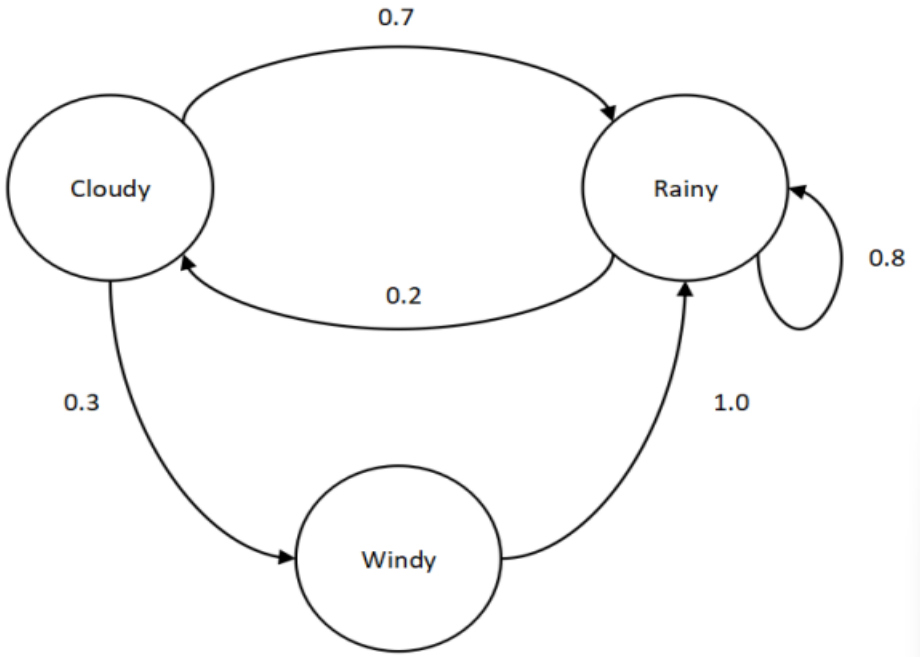


Figure 1.9: A state diagram of a Markov chain

	Cloudy	Rainy	Windy
Cloudy	0.0	0.7	0.3
Rainy	0.2	0.8	0.0
Windy	0.0	1.0	0.0

Figure 1.10: A transition matrix

- **Hence, a Markov process consists of a set of states along with their transition probabilities**

MDP…

- **Markov Reward process (MRP) :**
  - An extension of the Markov chain with the reward function.
  - A reward function says the reward we obtain in each state $R(s)$.
  - The MRP consists of states s, transition probabilities $P(s'|s)$and a reward function $R(s)$.
- **Markov Decision Process(MDP):**
  - An extension of the MRP with states s, actions a, transition probabilities $P(s'|s)$and a reward function $R(s)$.
- In a RL setup, the agent makes decisions only based on the current state and not based on the past states.
- Hence we can model a RL problem as a MDP

Grid World as MDP

- Goal : the agent has to move from state A to state I, without visiting the shaded states.

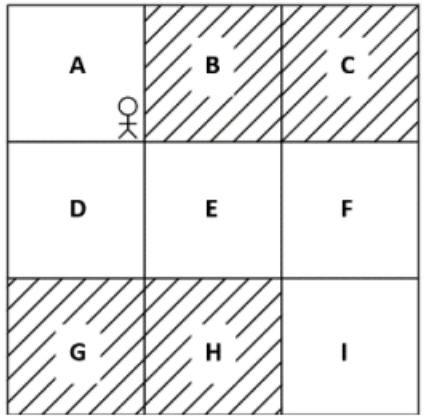


Figure 1.11: Grid world environment

- **States** : set of states, from A to I
- **Actions** : a set of actions that our agent can perform in each state as in up, down, left, right
- **Transition probability:** The probability of moving from the current state s to the next state s',
  while performing an action a is denoted by $P(s'|s,a)$
- Ex: $P(B|A,right) = 1.0$

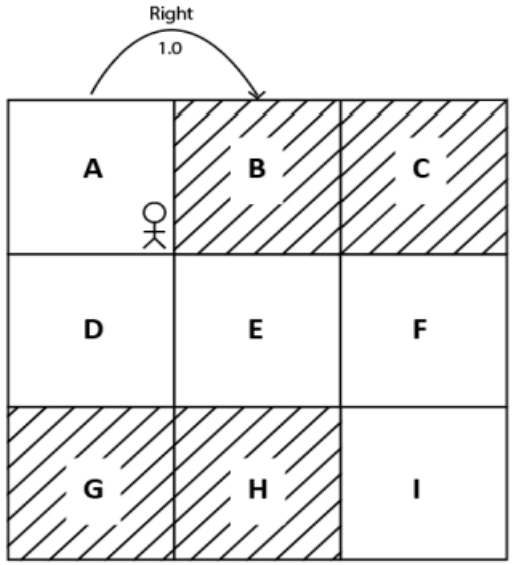
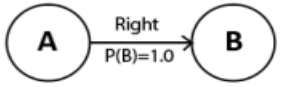


Figure 1.12: Transition probability of moving right from A to B



Grid World as MDP

- **Reward function**: the reward the agent receives while moving from state s to state s' while performing action a. Denoted by $R(s, a, s')$.
- Ex: $R(A, right, B) = -1$. $R(C, down, F) = +1$

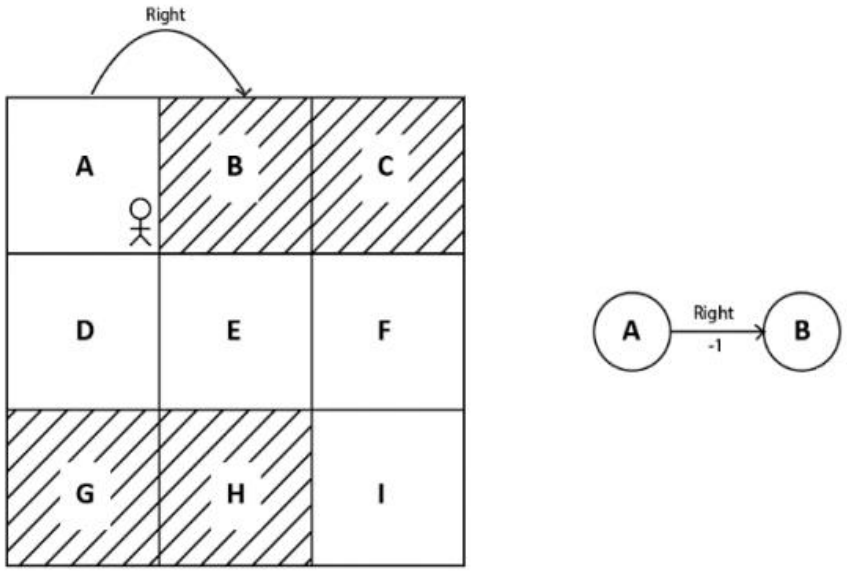


Figure 1.14: Reward of moving right from A to B

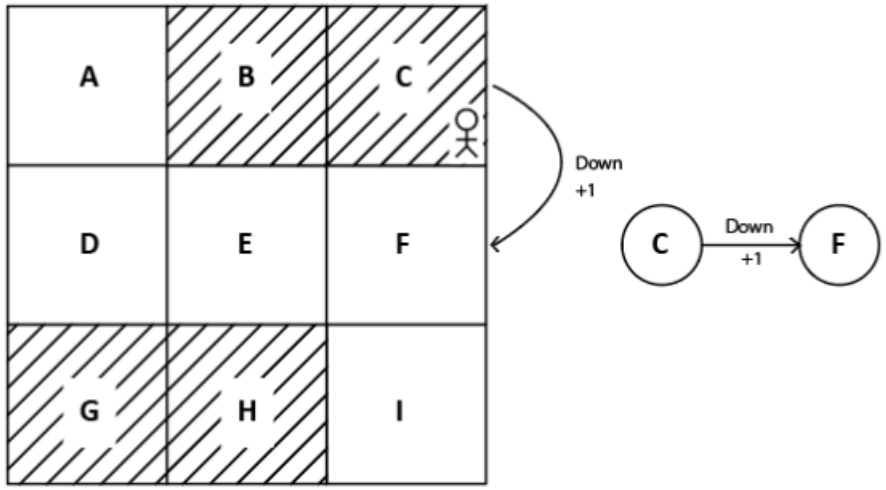


Figure 1.15: Reward of moving down from C to F

Slides Prepared by Dr J Alamelu Mangai

Fundamental concepts of RL

- **Maths essentials : Expectation of a random variable X**
- A random variable takes values from a random experiment such as throwing a dice, tossing a coin, etc.
- Ex : if we are throwing a fair dice, then the possible outcomes(X) are 1,2,3,4,5,and 6.
- The probability of occurrence of each of these outcomes are 1/6

X	1	2	3	4	5	6
P(x)	1/6	1/6	1/6	1/6	1/6	1/6

Table 1.2: Probabilities of throwing a dice

- Find the average value of the random variable X?

  Ans : take the weighted average of X

$$E(X) = \sum_{i=1}^{N} x_i p(x_i)$$

Thus, the expectation of the random variable X is $E(X) = 1(1/6) + 2(1/6) + 3(1/6) + 4(1/6) + 5\ (1/6) + 6(1/6) = 3.5$.

Fundamental concepts of RL

- **Expectation of a function of a random variable X**
- Ex: Let $f(X) = X^2$. *Find the expected value of* $f(X)$

X	1	2	3	4	5	6
f(x)	1	4	9	16	25	36
P(x)	1/6	1/6	1/6	1/6	1/6	1/6

Table 1.3: Probabilities of throwing a dice

The expectation of a function of a random variable can be computed as:

$$\mathbb{E}_{x\sim p(x)}[f(X)] = \sum_{i=1}^{N} f(x_i)p(x_i)$$

Thus, the expected value of $f(X)$ is given as $E(f(X)) = 1(1/6) + 4(1/6) + 9(1/6) + 16(1/6) + 25(1/6) + 36(1/6) = 15.1$.

Slides Prepared by Dr J Alamelu Mangai

Fundamental concepts of RL

- **Action Space** : the set of all possible actions in the environment
  - Ex: for the grid world environment, the action space is [up,down,left,right]
- **Types of Action Space**: discrete and continuous
- **A Discrete action space** has actions that are discrete.
  - Ex: the action space of the grid world environment
- A **continuous action space** has action that are continuous.
  - Ex: training an agent to drive a car, actions are continuous in nature such as speed, degrees to rotate the wheel, etc..

Fundamental concepts of RL

- **A policy** defines the agent's behaviour in an environment.
- It tells the agent what action to perform in each state.
- Ex: in the grid world envt with states from A to I, and 4 actions, a policy may tell the agent to move **down** in state A, move **right** in state D, so on.
- In the first iteration, the agent starts with a random policy, taking a random action in each state.
- Learns whether the actions taken in each state are good or bad based on the reward it gets.
- Over a series of iterations, the agent learns a good policy that gets a positive reward.
- This **good(optimal) policy** is the policy that gets the agent a good reward and helps the agent to reach the goal state.

Fundamental concepts of RL..

- Ex of an optimal policy

Optimal Policy

State	Action
A	Down
D	Right
E	Right
F	Down

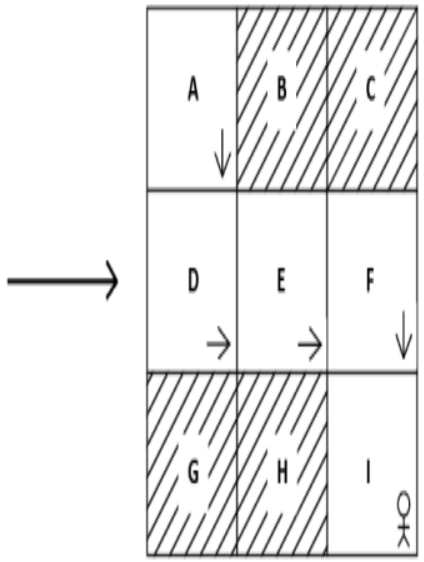


Figure 1.17: The optimal policy in the grid world environment

- **Types of policy : deterministic and stochastic**
- **Deterministic Policy:**
  - this policy tells the agent to perform one particular action in a state.
  - Denoted by μ
  - If the agent is in state 's' at time 't', the deterministic policy tells the agent to perform action 'a', expressed by
    $a_t = \mu(s_t)$
    Ex : $\mu(A) = down$
- **Stochastic policy** :
  - This policy doesn't map a state to one particular action.
  - It maps a state to a probability distribution over an action space
  - Denoted by π, $a_t \sim \pi(s_t)$ or $\pi(a_t|s_t)$
  - Ex: if the stochastic policy for state A over the 4 action space [up,down,left, right] is [0.10,0.70,0.10,0.10] respectively, when the agent in state A, it chooses action 'up' 10% of the time, 'down' 70% of the time, left 10% of the time and right 10% of the time.

Fundamental concepts of RL..

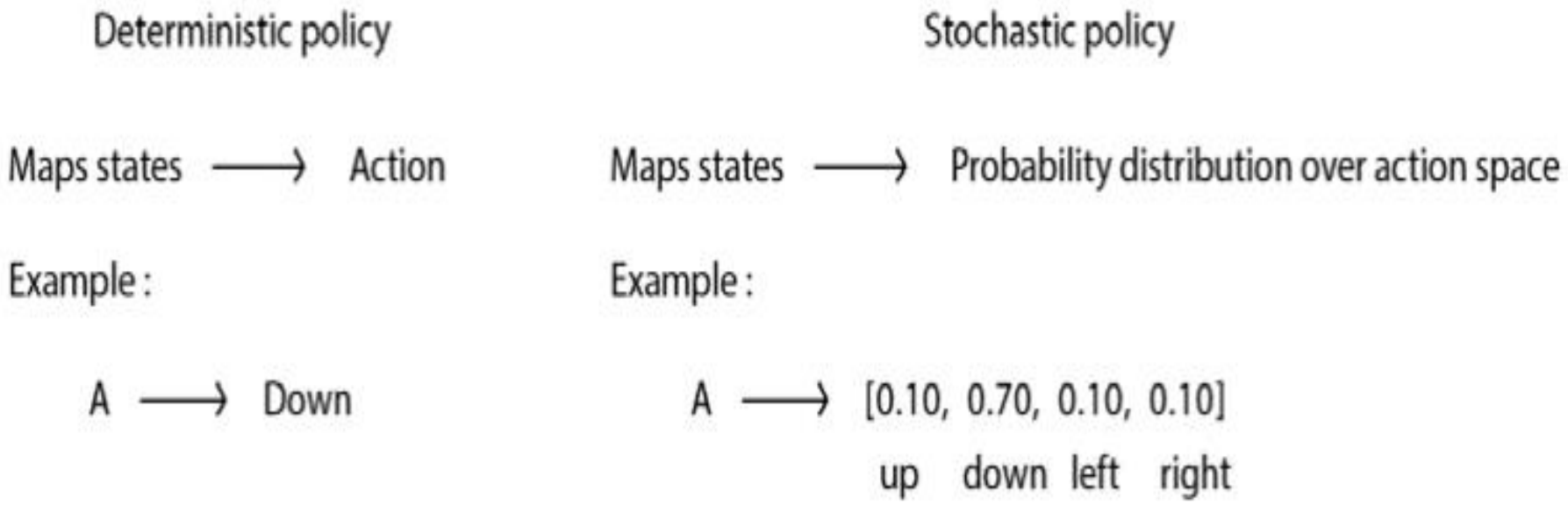


Figure 1.18: The difference between deterministic and stochastic policies

- **Types of Stochastic policy : Categorical and Gaussian**
- **Categorical policy:**
- If the action space of a stochastic policy is discrete, then it is a categorical policy
- Prob distributions are taken over a discrete action space

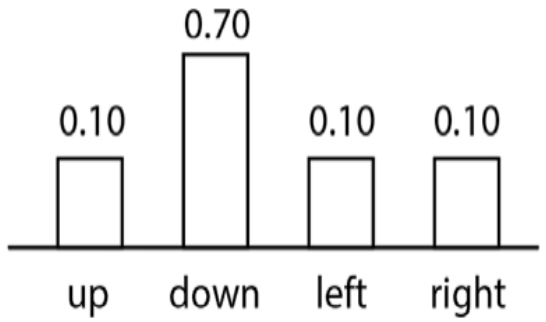


Figure 1.19: Probability of next move from state A for a discrete action space

- **Gaussian policy**
- A stochastic policy whose action space is continuous
- Its uses a Gaussian prob distribution over an action space
- Ex: if we are training an agent to drive a car, there is a continuous action in our action space – speed of the car whose value ranges from 0 to 150kmph.
- The stochastic policy uses the Gaussian distribution over the action space to select an action

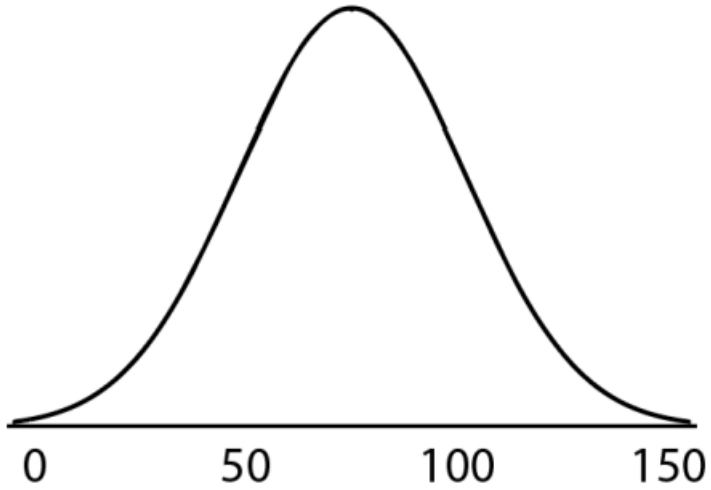
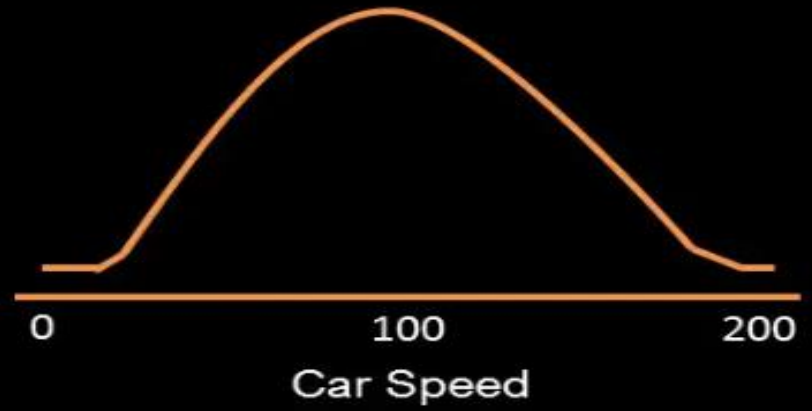


Figure 1.20: Gaussian distribution

Categorical Policy vs Gaussian Policy in Reinforcement Learning



Categorical Policy is used when action space is discrete. It selects the actions from the categorical distribution of discrete action space.

Gaussian Policy is used when action space is continuous. It selects the action from the Gaussian distribution of continuous action space.

Slides Prepared by Dr J Alamelu Mangai

- **Episode**: the agent-environment interaction starting from the initial state until the final state is called an episode
- Often known as a trajectory(the path taken by the agent)
- Denoted by τ
- An agent can play a game any no. of episodes, each episode is independent of the other.
- What is the use of playing the same game for multiple episodes?
  - To learn the optimal policy, that is, the policy that tells the agent to perform the correct action in each state
- The episode information is of the form state, action, reward starting from the initial state to the final state, i.e $(s_0, a_0, r_0, s_1, a_1, r_1, \ldots\ldots s_T)$

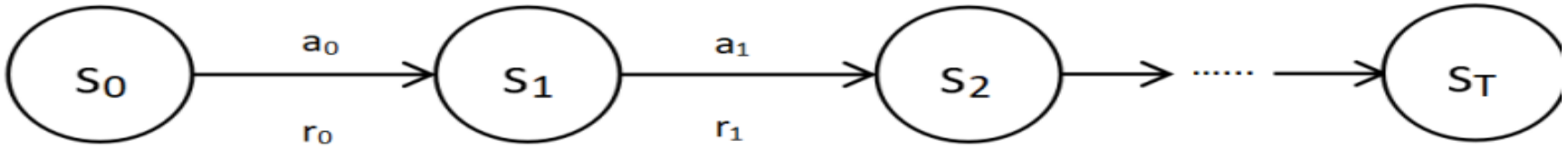


Figure 1.21: An example of an episode

Alamelu Mangai

Episode and optimal policy in Grid world Envt

- The agent generates the first episode using a random policy
- Explores the envt over several episodes to learn an optimal policy
- **Episode 1**

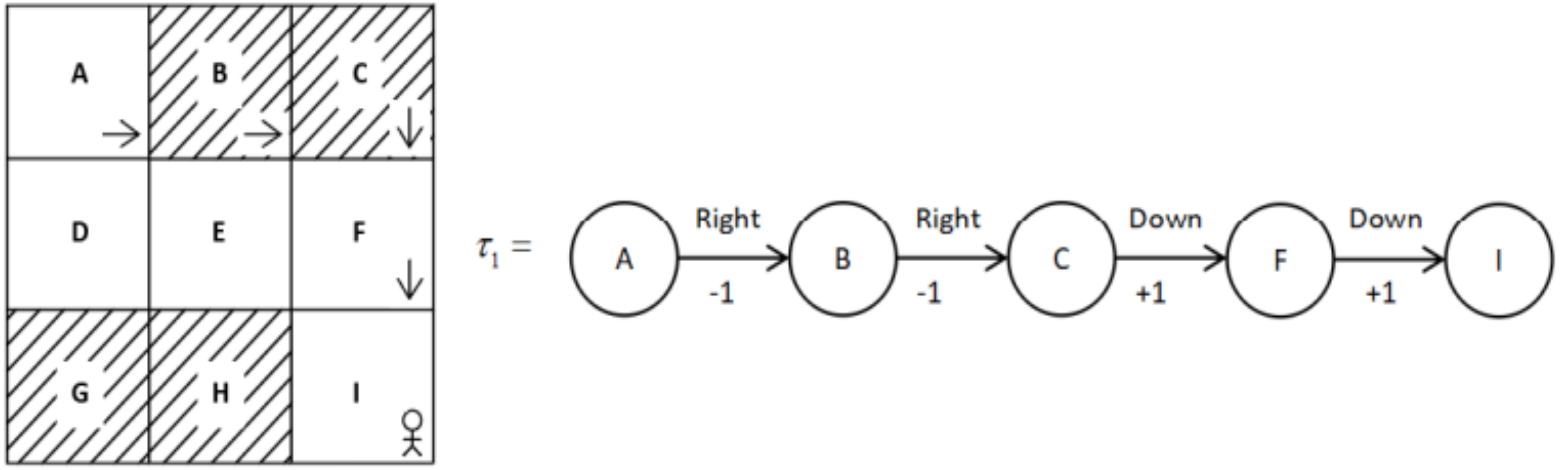


Figure 1.22: Episode 1

- **Episode 2** : the agent tries a different policy to avoid the negative rewards it got in the previous episode

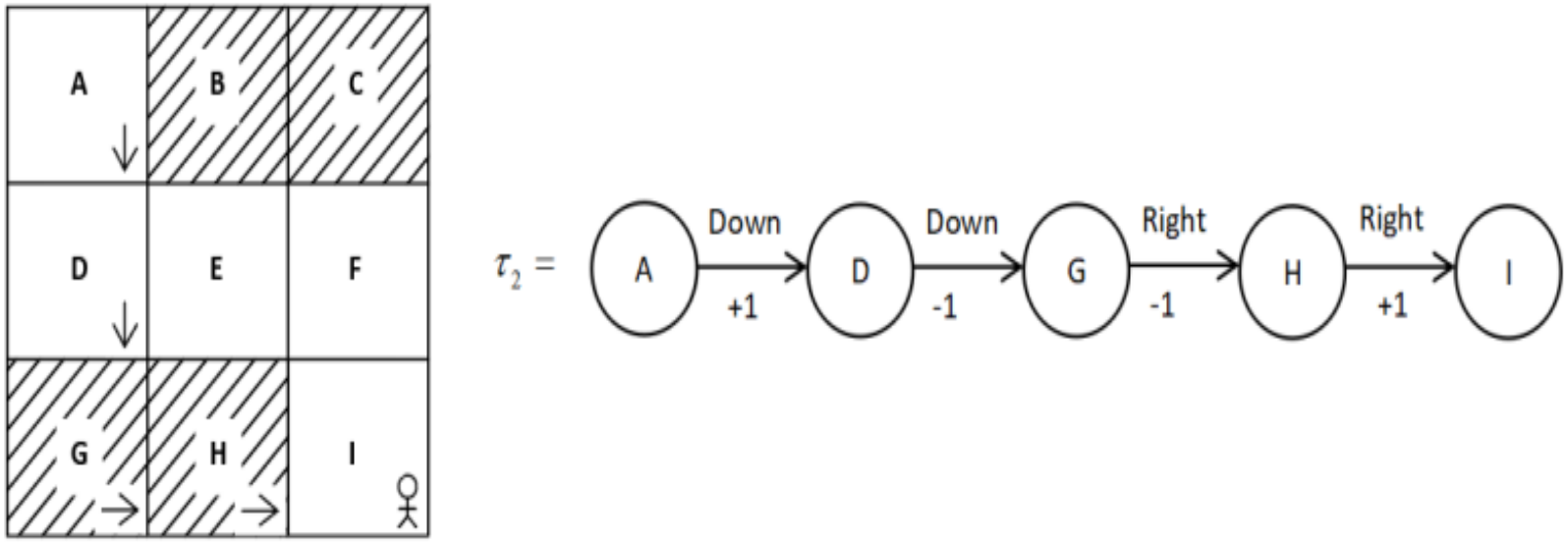


Figure 1.23: Episode 2

- **Episode n:** Over a series of episodes, the agent learns the optimal policy, the policy that takes the agent from state A to state I, without visiting the shaded states and also maximising the rewards.
-

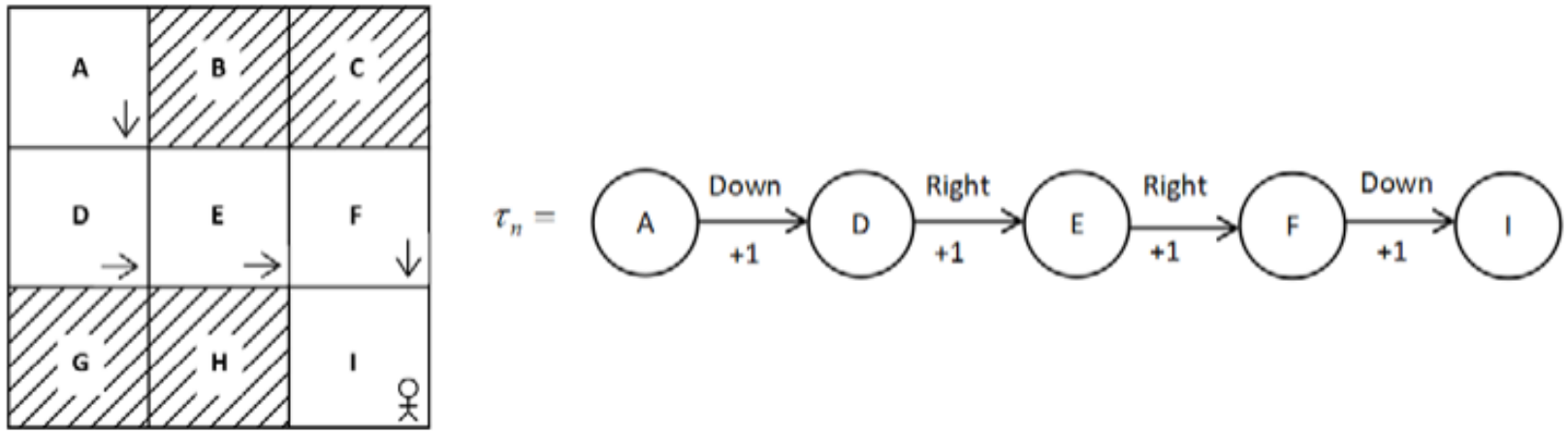


Figure 1.24: Episode n

Episodic and continuous tasks

- **Episodic tasks**: a task made up of episodes and thus they have a terminal state
  - Ex: car racing game
- **Continuous tasks**: do not have any episodes and so don't have any terminal state.
  - Ex: a personal assistance robot does not have a terminal state
- **Horizon :** the time step until which the agent interacts with the envt.
- **Types : finite and infinite horizon**
- **Finite horizon** : the agent-envt interaction stops at a particular time step.
  - Ex: in an episodic task, the agent-envt interaction stops after the agent reaches the final state T.
- **Infinite horizon**: the agent-envt interaction never stops
  - Ex: a continuous task without final state has an infinite horizon.

Return and discount factor

- **Return** : sum of the rewards obtained by an agent in an episode.
- Denoted by R or G.
- Ex: if the agent starts at initial state at time step t=0 and reaches the final state at time step T, then the return by the agent is

$$R(\tau) = r_0 + r_1 + r_2 + \ldots + r_T$$

$$R(\tau) = \sum_{t=0}^{T} r_t$$

- Ex: for the trajectory below:

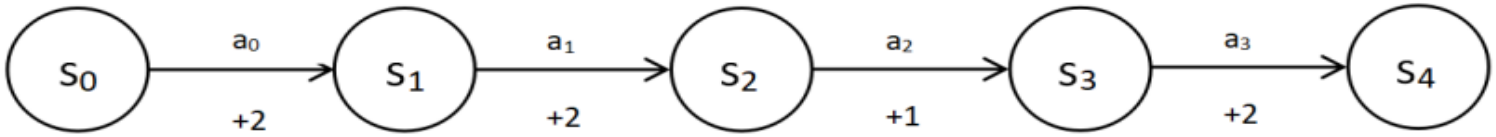


Figure 1.25: Trajectory/episode τ

The return of the trajectory is the sum of the rewards, that is,
$R(\tau) = 2 + 2 + 1 + 2 = 7$.

Dr J Alamelu Mangai

Return and discount factor..

- So the goal of the agent is to maximise the return, i.e, maximise the sum of the rewards obtained over an episode.
- How can we maximise this return? How can we perform the correct action in each state?
- By using the **optimal policy – the policy that gets our agent the maximum return (sum of the rewards) by performing the correct action in each state**.
- How to define return for continuous tasks, where there is no terminal state?
- Return for continuous tasks – sum of the rewards upto infinity.

$$R(\tau) = r_0 + r_1 + r_2 + \ldots + r_\infty$$

Return and discount factor..

- **How to maximise the return that sums to infinity?**
  - Using discount factor

$$R(\tau) = \gamma^0 r_0 + \gamma^1 r_1 + \gamma^2 r_2 + \ldots + \gamma^n r_\infty$$

$$R(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t$$

- Discount factor helps us by preventing the return in reaching infinity, by deciding how much importance we should give to immediate rewards and future rewards.
- Its value ranges from 0 to 1

Return and discount factor..

- If the discount factor is too small(close to 0), it means we give more importance to immediate rewards than future rewards.
- If the discount factor is set to a large value(close to 1), it means we give more importance to future rewards than immediate rewards.

Return and discount factor..

- **What happens when the discount factor is small? γ =** 0.2?

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\
&= (0.2)^0 r_0 + (0.2)^1 r_1 + (0.2)^2 r_2 + \ldots \\
&= (1) r_0 + (0.2) r_1 + (0.04) r_2 + \ldots
\end{aligned}
$$

- At time step 0, the reward r_0 is weighted by a discount factor of 1.
- At time step 1, the reward r_1 is weighted by a heavily decreased discount factor of 0.2.
- At time step 2, the reward r_2 is weighted by a heavily decreased discount factor of 0.04.

- **So, when we set discount factor to a small value, we give more importance to immediate rewards than future rewards.**

Slides Prepared by Dr J Alamelu Mangai

Return and discount factor..

- **What happens when the discount factor is large? γ =** 0.9?

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\
&= (0.9)^0 r_0 + (0.9)^1 r_1 + (0.9)^2 r_2 + \ldots \\
&= (1) r_0 + (0.9) r_1 + (0.81) r_2 + \ldots
\end{aligned}
$$

- At time step 0, the reward r_0 is weighted by a discount factor of 1.
- At time step 1, the reward r_1 is weighted by a slightly decreased discount factor of 0.9.
- At time step 2, the reward r_2 is weighted by a slightly decreased discount factor of 0.81.

As we can observe, the discount factor is decreased for subsequent time steps but unlike the previous case, the discount factor is not decreased heavily. Thus, when we set the discount factor to a high value, we give more importance to future rewards than the immediate reward.

Slides Prepared by Dr J Alamelu Mangai

Immediate or future rewards?

- It depends on the task that we want to train an agent for.
- Suppose, in a chess game, the goal is to defeat the opponent's king.
- If we give importance to the immediate rewards like a reward on pawn defeat any opponent player then the agent will learn to perform these sub-goals no matter if his players are also defeated.
- So, in this task future rewards are more important

Return and discount factor..

- **What happens when the discount factor is set to 0? γ =** 0?

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \ldots \\
&= (0)^0 r_0 + (0)^1 r_1 + (0)^2 r_2 + \ldots \\
&= (1) r_0 + (0) r_1 + (0) r_2 + \ldots \\
&= r_0
\end{aligned}
$$

- **Our return is just the immediate reward**

Return and discount factor..

- **What happens when the discount factor is set to 1? γ =** 1?

$$
\begin{aligned}
R &= (\gamma)^0 r_0 + (\gamma)^1 r_1 + (\gamma)^2 r_2 + \dots \\
&= (1)^0 r_0 + (1)^1 r_1 + (1)^2 r_2 + \dots \\
&= r_0 + r_1 + r_2 + \dots
\end{aligned}
$$

- **Our return is just the sum of the rewards upto infinity**

Return and discount factor..

- If the discount factor is set to 0, the agent never learns, as it considers only the immediate reward.
- If the discount factor is set to 1, the agent will learn forever, looking for the future rewards that lead to infinity.
- So, the optimal value of discount factor is between 0.2 to 0.8
- For certain tasks future rewards are more important than immediate rewards and vice versa.

Value function

- **Value function** also called the state value function gives the value of a state.
- The value of a state ‘s’ is the return of the trajectory τ starting from that state to the final state following a policy π.

$$V^{\pi}(s) = [R(\tau)|s_0 = s]$$

- The policy could be deterministic or stochastic
- A deterministic policy maps each state to a one particular action
- A stochastic policy selects action for a state based on a probability distribution of the action space.

Value function for a deterministic policy

- If the trajectory τ, for the grid world environment, using some policy deterministic policy π is :

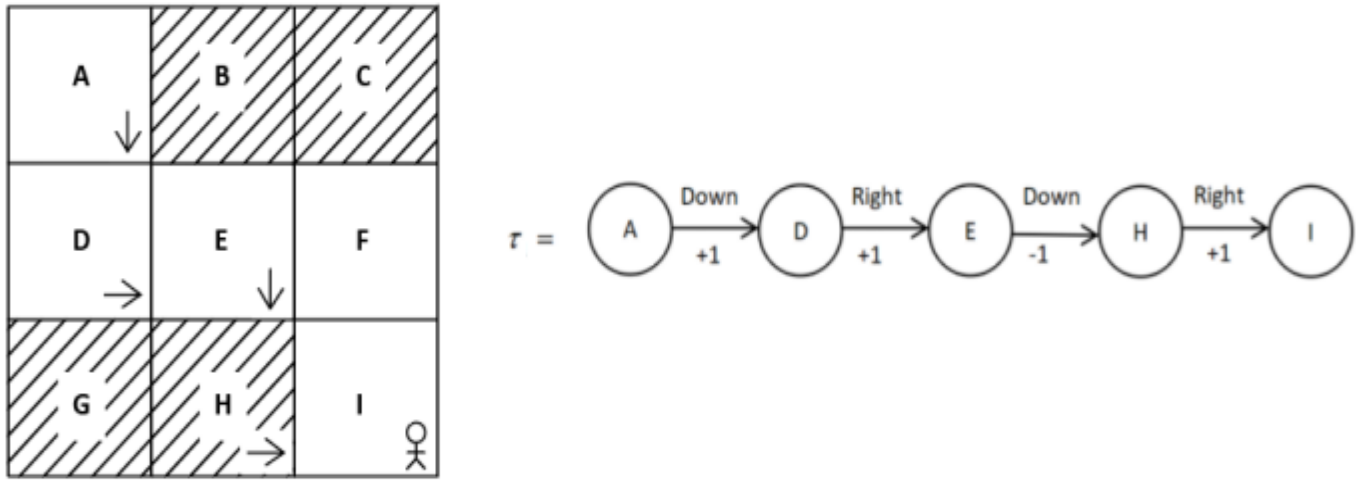


Figure 1.26: A value function example

Value function for a deterministic policy…

- The value function can be calculated for each state as the return(sum of the rewards) of the trajectory starting from that state :
  - The value of state **A** is the return of the trajectory starting from state **A**. Thus, $V(A) = 1+1+ -1+1 = 2$.
  - The value of state **D** is the return of the trajectory starting from state **D**. Thus, $V(D) = 1-1+1= 1$.
  - The value of state **E** is the return of the trajectory starting from state **E**. Thus, $V(E) = -1+1 = 0$.
  - The value of state **H** is the return of the trajectory starting from state **H**. Thus, $V(H) = 1$.
- The value function of the finals state is zero, since a reward is associated only with a state that has a transition.

Value function for stochastic policy

- (Expected)value function of a state with a stochastic policy is the expected return that the agent would get starting from that state s and following a stochastic policy π.
- Return of a state in a trajectory τ, following a stochastic policy π, is a random variable.
- It takes different values with some probability in each trajectory.
- It is expressed as :

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau) | s_0 = s]$$

Value function for stochastic policy…

- Ex: In state A, the stochastic policy gives a prob distribution over the action space [up,down,left,right] as [0.0, 0.8, 0.0, 0.2], i.e

  perform the action down 80% of the time, that is, $\pi(\text{down}|A) = 0.8$, and the action right 20% of the time, that is $\pi(\text{right}|A) = 0.20$

- This gives two trajectories from state A.
- Assume the stochastic policy selects "right" in states D and E and "down" in B and F 100% of the time.

- $\pi(\text{down}|A) = 0.8$ $\pi(\text{right}|A) = 0.20$
- $\pi(\text{right}|\text{D}) = 1$ $\pi(\text{right}|\text{E}) = 1$
- $\pi(\text{down}|\text{B}) = 1$ $\pi(\text{down}|\text{F}) = 1$

- $\pi(\text{right}|\text{D}) = 0.8$ $\pi(\text{down}|\text{D}) = 0.2$
- $(\text{right}|\text{E}) = 1$ $\pi(\text{down}|\text{B}) = 1$ $\pi(\text{down}|\text{F}) = 1$
- $\pi(\text{right}|\text{G}) = 1$ $\pi(\text{right}|\text{H}) = 1$

Value function for stochastic policy…

- **The first trajectory τ_1 with $\pi(\text{down}|A) = 0.8$ is :**

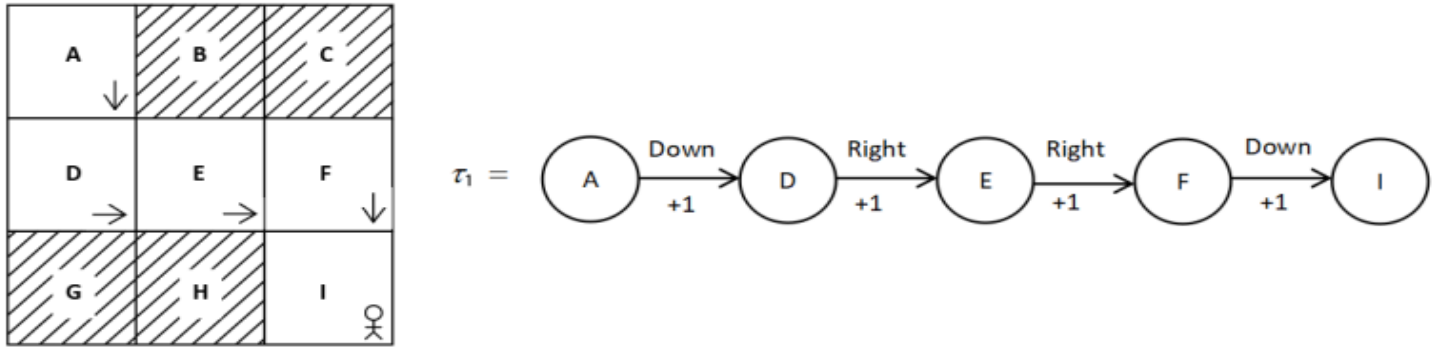


Figure 1.27: Episode τ_1

- Value of state A is, the return(sum of the rewards) of the trajectory starting from state A.
- Thus, $V(A) = R(\tau_1)$ =1+1+1+1=4.

Value function for stochastic policy…

- **The second trajectory τ_2 with $\pi(\text{right}|A) = 0.20$ is :**

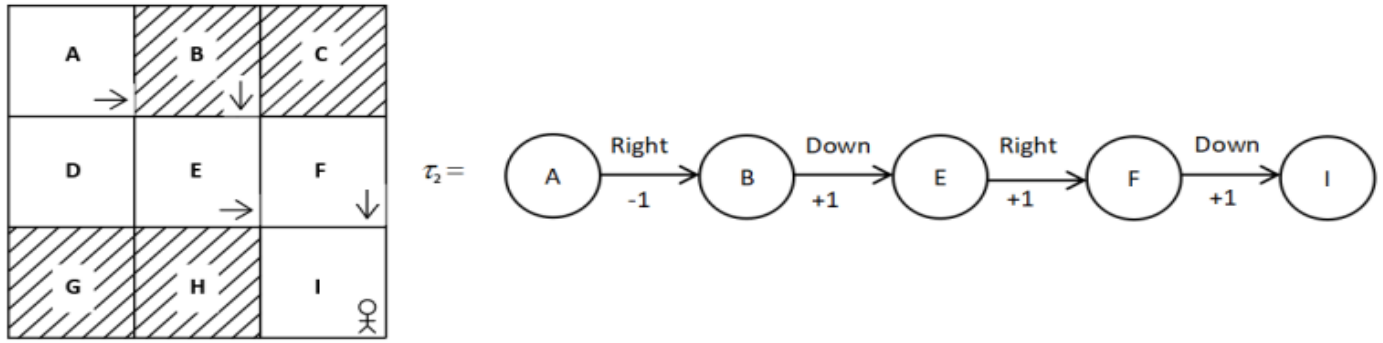


Figure 1.28: Episode τ_2

- Value of state A is, the return(sum of the rewards) of the trajectory $\boldsymbol{\tau_2}$ starting from state A.
- Thus, $V(A) = R(\tau_2)$ =-1+1+1+1=2.
- Also, it's the same policy, V(A) differs with the trajectories.
- For this policy, return is 4, 80% of the time and 2, for 20% of the time.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)|s_0 = s]$$

Value function for stochastic policy…

- Value of a state for a stochastic policy is the expected return of the trajectory starting from that state.

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = s]$$

- Expected return is the weighted average, sum of the returns, multiplied by their probabilities.
- So, V(A) is :

$$V^{\pi}(A) = \mathop{\mathbb{E}}_{\tau \sim \pi}[R(\tau)|s_0 = A]$$

$$= \sum_i R(\tau_i)\pi(a_i|A)$$

$$= R(\tau_1)\pi(\text{down}|A) + R(\tau_2)\pi(\text{right}|A)$$
$$= 4(0.8) + 2(0.2)$$
$$= 3.6$$

Value function for stochastic policy…

- Thus, the value of a state is the expected return of the trajectory starting from that state.
- Value function depends on the policy.
- There can be many value functions for a state, according to different policies.
- The optimal value function V*(s) is the maximum value of the state, among all its value functions

$$V^*(s) = \max_{\pi} V^{\pi}(s)$$

Ex: Text book Pg.33 . We can find the optimal state from a Value table.

State	Value
s_0	7
s_1	11

Table 1.4: Value table

Slides Prepared by Dr J Alamelu Mangai

Q function

- Q function denotes the value of a state-action pair for a particular state, s.
- It is the return that the agent will obtain starting from a state s, and performing an action a, following a policy π
- It is also known as state-action value function

$$Q^{\pi}(s,a) = [R(\tau)|s_0 = s, a_0 = a]$$

- Ex: Given trajectory τ, :

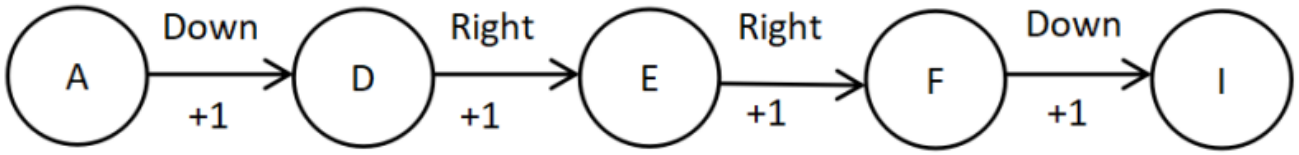


Figure 1.29: A trajectory/episode example

Q function…

- Find the q-function of **A-down, D-right :**
- **Ans 1:**

$$Q^{\pi}(A, \text{down}) = [R(\tau)|s_0 = A, a_0 = \text{down}]$$

$$Q(A, \text{down}) = 1 + 1 + 1 + 1 = 4$$

- **Ans 2: ? 3**
- **Since return is a random variable taking a different value with some probability, instead of taking the return directly, we take the expected return.**

$$Q^{\pi}(s, a) = \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau)|s_0 = s, a_0 = a]$$

It implies that the Q value is the expected return the agent would obtain starting from state s and performing action a following policy π.

Q function….

- Q function depends on the policy.
- There will be Q values for a (s,a) pair depending on the policy.
- The optimal policy for a (s,a) pair is :

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a)$$

- The optimal policy π^* is the policy that gives the maximum Q value for a (s,a).
- Given a Q table, we can find the optimal policy π^*

Q Table

State	Action	Value
s_0	0	9
s_0	1	11
s_1	0	17
s_1	1	13

Optimal policy

State	Action
s_0	1
s_1	0

Table 1.6: Optimal policy extracted from the Q table

Model-based and model-free learning

- **Model-based learning** : the agent learns the optimal policy by using the model dynamics of the environment.
- Model dynamics of the environment is defined using
  - 1. state transition probabilities and 2. reward function

- **Model-free learning** : the agent tries to learns the optimal policy without using the model dynamics of the environment

Bellman Equation and Dynamic Programming

- LO1: State the Bellman equation of the value function and Q function
- LO2: Solve MDP using Bellman equation
- LO3: Apply Bellman equation to find the value function of a state for a given deterministic environment

Slides Prepared by Dr J Alamelu Mangai

Bellman Equation and Dynamic Programming

- In RL, the agent has to learn an optimal policy to perform a particular task
- An optimal policy selects the correct action for an agent in each state, so that the agent can get the maximum return and achieve its goal.
- Two classical RL algorithms – **value and policy iteration**, helps the agent to learn an optimal policy.
- These algos are model based and use Dynamic Programming
- Bellman equation is used in RL to find the optimal value function and optimal Q functions recursively.
- These are then used to find the optimal policy

Bellman Equation?

- Bellman equation?
- Bellman optimality equation?
- Relationship between value and Q function?
- Dynamic Programming – value and policy iteration methods?
- Solving the Frozen Lake problem using value and policy iteration methods.

Bellman equation for the value function

- As per Bellman Equation, the value of a state is the sum of the immediate reward $R(s, a, s')$and the discounted value of the next state $\gamma V(s')$.

$$V(s) = R(s, a, s') + \gamma V(s')$$

- $R(s, a, s')$ implies the immediate reward obtained while performing an action a in state s and moving to the next state s'
- γ is the discount factor
- $V(s')$ implies the value of the next state

Bellman equation for the value function

- **In a deterministic environment:**
- Ex: Given a trajectory τ using some policy π as :

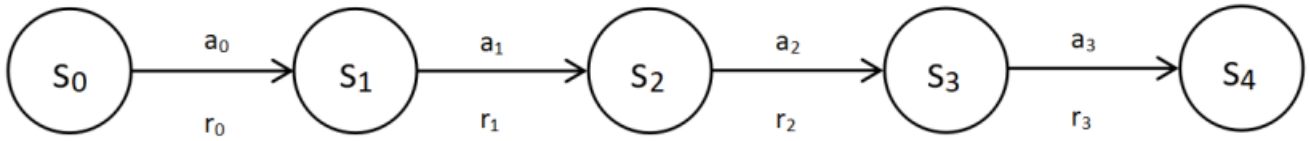


Figure 3.1: Trajectory

- Using Bellman equation find V(s_2)?
- Ans: $V(s_2) = R(s_2, a_2, s_3) + \gamma V(s_3)$

  $= r_2 + \gamma V(s_3)$
- Hence the Bellman equation of the value function for a deterministic environment associated with a policy π is :

$$V^{\pi}(s) = R(s, a, s') + \gamma V^{\pi}(s')$$

- Where the rhs term is known as **Bellman backup**

Bellman equation for the value function

- **In a stochastic environment:**

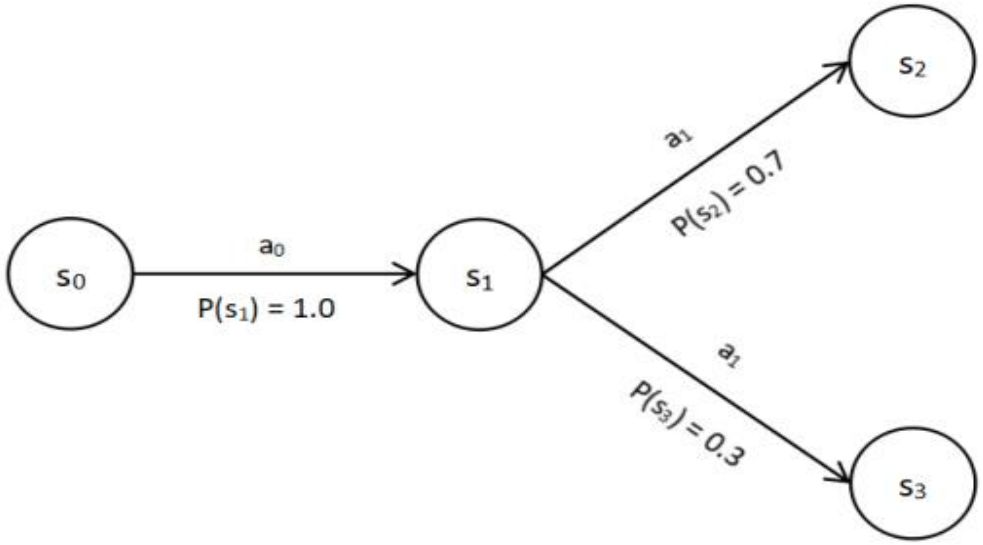


Figure 3.2: Transition probability of performing action a_1 in state s_1

Bellman equation for the value function

- Modify the Bellman equation of the value function with expectations (weighted average)
- Bellman backup multiplied with the transition probability of the next state.

$$V^{\pi}(s) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

In the preceding equation, the following applies:

- $P(s'|s,a)$ denotes the transition probability of reaching s' by performing an action a in state s
- $[R(s,a,s') + \gamma V^{\pi}(s')]$ denotes the Bellman backup

Bellman equation for the value function

- Ex: same trajectory, find V(s_1)?
- Ans:

$$V(s_1) = P(s_2|s_1,a_1)[R(s_1,a_1,s_2) + V(s_2)] + P(s_3|s_1,a_1)[R(s_1,a_1,s_3) + V(s_3)]$$

$$V(s_1) = 0.70[R(s_1,a_1,s_2) + V(s_2)] + 0.30[R(s_1,a_1,s_3) + V(s_3)]$$

- **Hence the Bellman equation for the value function for a stochastic environment for a policy π is :**

$$V^{\pi}(s) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

Bellman equation for the value function

- **What if the policy itself is stochastic?** Instead of performing the same action in a state, we select an action based on the prob distbn over the action space.
- Ex:

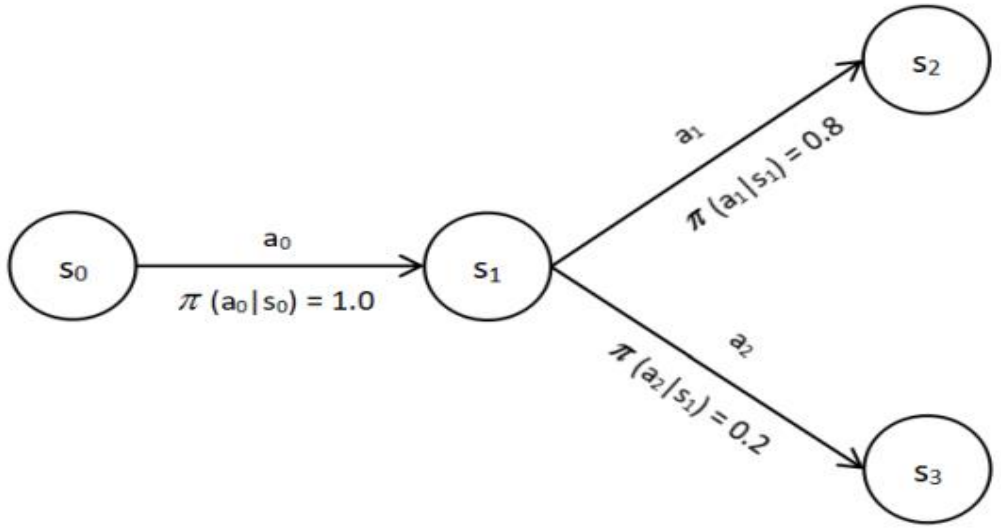


Figure 3.3: Trajectory using a stochastic policy

Bellman eqn for value function…

- to include the **stochasticity present in the environment** in the Bellman equation, we took the expectation (the weighted average), that is, a sum of the Bellman backup multiplied by the corresponding transition probability of the next state. •
- Similarly, to include the **stochastic nature of the policy** in the Bellman equation, we can use the expectation (the weighted average), that is, a sum of the Bellman backup multiplied by the corresponding probabilitv of action.

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^{\pi}(s')]$$

- Using expectations :

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{a \sim \pi \\ s' \sim P}} [R(s,a,s') + \gamma V^{\pi}(s')]$$

Slides Prepared by Dr J Alamelu Mangai

Bellman Eqn of the Q function

- **For deterministic envt** : Bellman eqn of the Q function says that, the Q value of a state-action pair is a sum of the immediate reward and the discounted Q value of the next state-action pair :

$$Q(s,a) = R(s,a,s') + \gamma Q(s',a')$$

In the preceding equation, the following applies:

- $R(s,a,s')$ implies the immediate reward obtained while performing an action a in state s and moving to the next state s'
- γ is the discount factor
- $Q(s',a')$ is the Q value of the next state-action pair

- Ex: Given a trajectory τ using some policy π, find the Q value of (s_2, a_2).

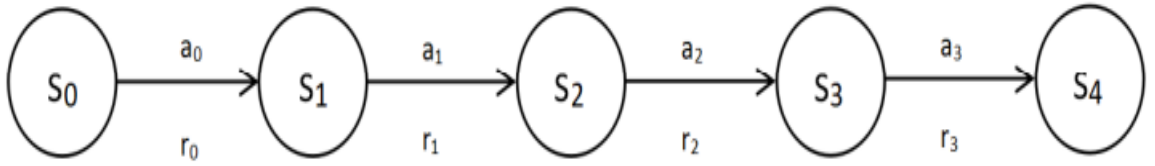


Figure 3.4: Trajectory

- *Ans:*

$$Q(s_2, a_2) = R(s_2, a_2, s_3) + \gamma Q(s_3, a_3)$$

$$Q(s_2, a_2) = r_2 + \gamma Q(s_3, a_3)$$

Thus, the Bellman equation for the Q function can be expressed as:

$$Q^{\pi}(s, a) = R(s, a, s') + \gamma Q^{\pi}(s', a')$$

Where the superscript π implies that we are using the policy π and the right-hand side term $R(s, a, s') + \gamma Q^{\pi}(s', a')$ is the **Bellman backup**.

Bellman Eqn of the Q function…

- **For stochastic environment**: An agent in state ‘s’, performs an action ‘a’, then the next state is not always the same.
- Bellman eqn of the Q function for a stochastic envt, uses the expectation (weighted average), that is, a sum of the Bellman backup multiplied by their corresponding transition probability of the next state.
- The Bellman equation of the Q function is:

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma Q^{\pi}(s',a')\right]$$

Bellman Eqn of the Q function…

- With a stochastic policy, we cannot guarantee a' in s'

$$Q^{\pi}(s,a) = \sum_{s'} P(s'|s,a)\left[R(s,a,s') + \gamma \sum_{a'} \pi(a'|s')Q^{\pi}(s',a')\right]$$

- Final Bellman Equation for Q(s,a) using expectation is :

$$Q^{\pi}(s,a) = \mathbb{E}_{s' \sim P}[R(s,a,s') + \gamma \mathbb{E}_{a' \sim \pi} Q^{\pi}(s',a')]$$

Bellman Optimality Theorem

- Gives the optimal Bellman value and Q function
- V(s) depends on the policy .
- The optimal Bellman V(s) for a particular state s, is denoted by $V^*(s)$
- It is the one that gives the maximum value among all the values of that state.
- To find $V^*(s)$, find V(s) using all possible actions, without using any policy. Choose the max among all V(s).

$$V^*(s) = \max_a \mathbb{E}_{s' \sim P}[R(s, a, s') + \gamma V^*(s')]$$

- If there are two possible actions, namely 0 and 1 in state s, then

$$V^*(s) = \max \begin{pmatrix} \mathbb{E}_{s' \sim P}[R(s,0,s') + \gamma V^*(s')] \\ \mathbb{E}_{s' \sim P}[R(s,1,s') + \gamma V^*(s')] \end{pmatrix}$$

Optimal Bellman Q function

- Without using any policy, find the Q(s', a') for all possible a', find the max of them.

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

- If there are two actions 0 and 1 in s' then,

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}[R(s,a,s') + \gamma \max \begin{pmatrix} Q^*(s',0) \\ Q^*(s',1) \end{pmatrix}]$$

To summarize….

- The Bellman optimality equations of the value and the Q functions are :

$$V^*(s) = \max_a \mathbb{E}_{s' \sim P}[R(s,a,s') + \gamma V^*(s')]$$

$$Q^*(s,a) = \mathbb{E}_{s' \sim P}\left[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')\right]$$

- They can be written by removing the expectation as

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$$

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma \max_{a'} Q^*(s',a')]$$

- Relationship between $V^*(s)$and $Q^*(s,a)$is :
- The optimal value function is the maximum expected return when we start from a state s $V^*(s) = \max_\pi V^\pi(s)$
- The optimal Q function is the maximum expected return when we start from a state s and perform an action a $Q^\pi(s,a) = \max_\pi Q^\pi(s,a)$
- So, we can say that the optimal value function is the maximum of optimal Q value over all possible actions

$$V^*(s) = \max_a Q^*(s,a)$$

- We can derive V from Q : $V^*(s) = \max_a Q^*(s,a)$

- We can derive Q from V : $Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

- Substituting we get : $V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')]$

- These Bellman equations are used to find an optimal policy.

Dynamic Programming

- Dynamic programming (DP) is a technique for solving complex problems.
- In DP, instead of solving a complex problem as a whole, we break the problem into simple sub-problems,
- Solve each sub-problem and store the solution.
- If the same sub-problem occurs, we don't recompute; instead, we use the already computed solution.
- Thus, DP helps in drastically minimizing the computation time. It has its applications in a wide variety of fields including computer science, mathematics, bioinformatics, etc.
- Two methods that use DP to find an optimal policy:
  - **Value iteration and policy iteration**

Slides Prepared by Dr J Alamelu Mangai

- DP is a model-based method – it needs the model dynamics to find an optimal policy
- Model dynamics – state transition probabilities and reward function
- **An optimal policy** – tells the agent to perform the correct action in each state.

- To find this optimal policy, first we find the optimal value function of each state, $V^*(s)$.

- Later, use this optimal value function $V^*(s)$, to find the optimal policy [by finding the Q function]

- The Bellman's optimal value function is :

$$V^*(s) = \max_a \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \qquad (9)$$

- In the relationship between the value and Q functions, we know that given the value function we can derive the Q function :

$$Q^*(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V^*(s')] \qquad (10)$$

- Substituting (10) in (9), we get

$$V^*(s) = \max_a Q^*(s,a)$$

- If we know the Q values of all (s, a) pairs of a particular state s, using this we can find the optimal value function of that state.
- Ex: if the Q-values of all (s, a) pairs are :

State	Action	Value
s_0	0	2.7
s_0	1	3
s_1	0	4
s_1	1	2

- Using this we can find the optimal state values of each state as :

State	Value
s_0	3
s_1	4

- This is the outline of the value-iteration algorithm

The value iteration algorithm

- LO1: Name the two methods to find the optimal policy using dynamic programming
- LO2: Describe the value iteration algorithm to find an optimal policy
- LO3: Given the model dynamics of a particular state in a RL environment, apply the value iteration algorithm, to find an optimal policy

Slides Prepared by Dr J Alamelu Mangai

The value iteration algorithm

- **Step 1**: Compute the optimal value function of each state iteratively, by taking the max of the Q functions (of all possible actions in a state), i,e

$$V^*(s) = \max_a Q^*(s, a)$$

- **Step 2:** Extract the optimal policy from the computed optimal value function

Step 1 : Value iteration Algorithm..

1. Initialize the value table of all states to zero.
2. For each state s, do :

   2.1 for each possible action a in state s do:

      2.1.1 find the Q-value of this (s,a) pair as :

$$Q(s,a) = \sum_{s'} P(s'|s,a)[R(s,a,s') + \gamma V(s')] \qquad \text{[OR]} \qquad Q(s,a) = \sum_{s'} P^{a}_{ss'}\left[R^{a}_{ss'} + \gamma V(s')\right]$$

   2.2 find the max among the Q values and update that as the value of that state

$$V^*(s) = \max_a Q^*(s,a)$$

Slides Prepared by Dr J Alamelu Mangai

Value iteration Algorithm..

3. If the value table of 2 consecutive iterations doesn't change then, go to next step to find the optimal policy; else repeat step 2, using the updated value table of this iteration.

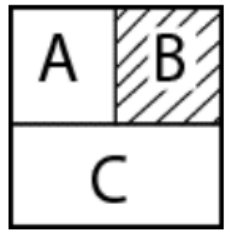
4. Find the Q value of each (s,a) pair using the optimal value table obtained in step 3 using :

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

5. Extract the optimal policy from the Q-value table got in step 4.

Value iteration Algorithm Example…

- Given a smaller grid world environment :



- Actions : 0 – left/right and 1 – up/down
- **Goal : from state A reach state C, without visiting the shaded state B**
- **What is the optimal policy?**
  - **Ans : perform action 1 in state A.**

- Given the model dynamics of state A:

Table 3.3 shows the model dynamics of state **A**:

State (s)	Action (a)	Next State (s')	Transition Probability $P(s'\|s,a)$ or $P^a_{ss'}$	Reward Function $R(s,a,s')$ or $R^a_{ss'}$
A	0	A	0.1	0
A	0	B	0.8	-1
A	0	C	0.1	1
A	1	A	0.1	0
A	1	B	0.0	-1
A	1	C	0.9	0

Table 3.3: Model dynamics of state A

Change the reward of the last row to 1

Solution:

- **Step 1: Compute the optimal value function**

1. Initialize the value table of all states to zero.
2. For each state s and all possible actions a find Q(s,a) as

$$Q(s,a) = \sum_{s'} P^{a}_{ss'}[R^{a}_{ss'} + \gamma V(s')]$$

3. Using $V^*(s) = \max_a Q^*(s,a)$ find $\max_a Q^*(s,a)$ and update this as the value table of state s
4. Repeat steps 2 and 3 until the value table converges.

- Example for Step 1:

Iteration 1:

Initialize value table to zero

State	Value
A	0
B	0
C	0

Stn1: On state A: Possible actions are (0, 1)

∴ find $Q(A,0) =$ ~~$P^{0}_{AA}[R^{0}_{AA} + \gamma V(A)] +$~~

$Q(A,1) =$

To visit $Q(s,a) = \sum P^{a}_{ss'}[R^{a}_{ss'} + \gamma V(s')]$

ngai

$$\therefore Q[A,0) = P_{AA}^{0}\left[R_{AA}^{0} + \gamma V(A)\right] + P_{AB}^{0}\left[R_{AB}^{0} + \gamma V(B)\right]$$

$$+ P_{AC}^{0}\left[R_{AC}^{0} + \gamma V(C)\right]$$

$$= 0.1\left[0 + 0\right] + 0.8\left[-1 + 0\right] + 0.1\left[1 + 0\right]$$

$$= 0 + 0.8 + 0.1 = -0.7$$

Slides Prepared by Dr J Alamelu Mangai

$$Q(A,1) = P'_{AA}\left[R'_{AA} + \gamma V(A)\right] + P'_{AB}\left[R'_{AB} + \gamma V(B)\right]$$

$$+ P'_{AC}\left[R'_{AC} + \gamma V(C)\right]$$

$$= 0.1\left[0 + 0.9 \times 0\right] + 0\left[-1 + 0\right] + 0.9\left[1 + 0\right]$$

$$= 0 + 0 + 0.9 = 0.9$$

∴ updated value table:

State	value
A	0.9
B	0
C	0

Q(B,0) = P

(B,1) =

Similarly, if we know the model dynamics of state B & C we can find Q(B,0) Q(B,1) Q(C,0) and Q(C,1). Finally updated state value at end of iter1 (assume) is:-

State	Value
A	0.9
B	-0.2
C	0.5

Slides Prepared by Dr J Alamelu Mangai

Step 2: Using the value table of itr 1, find the values of all states with all possible actions in each state:

Ex: On state A:-

$$R(A,0) = P^0_{AA}\left[R^0_{AA} + \gamma V(A)\right] + P^0_{AB}\left[R^0_{AB} + \gamma V(B)\right] + P^0_{AC}\left[R^0_{AC} + \gamma V(C)\right]$$

$$= 0.1\left[0 + 0.9\right) + 0.8\left(-1 - 0.2\right) + 0.1\left[1 + 0.5\right)$$

$$= -0.72$$

$$R(A,1) = P^1_{AA}\left[R^1_{AA} + \gamma V(A)\right] + P^1_{AB}\left[R^1_{AB} + \gamma V(B)\right] + P^1_{AC}\left[R^1_{AC} + \gamma V(C)\right]$$

$$= 0.1\left(0 + 0.9\right) + 0.0\left[-1 - 0.2\right) + 0.9\left(1 + 0.5\right)$$

$$= 1.44$$

Pg (20)

Similarly we find the Q-value of all states & update the state table as [Assumption]

State	Value
A	1.44
B	-0.50
C	1.0

Value table from iter2:

Itr 3: Repeat the same steps while updating the values, use the updated value table got from the previous itr.

Assume value table from itr 2

State	Value
A	1.94
B	-0.70
C	1.3

Convergence?

Keep repeating, until the value table bet 2 consecutive itrs does' change or changes by a very small (fraction based on a threshold)

Assume value table from itr 3 is

state	value
A	1.95
B	-0.72
C	1.3

~~itr~~ ∵ the diff is small, we take this as the optimal value table.

ngai

Next, step 2: to extract the optimal policy from the optimal value table.

Step2. Extract the optimal policy from the optimal value table (function) from step1.

Assume Optimal value table is

State	Value
A	1.95
B	-0.72
C	1.3

Use the Q fn to compute the policy for a state s.

2.1) find the Q-value of all (s, a) pairs as.

$$\hat{Q}(s,a) = \sum_{s'} P^{a}_{ss'} \left[R^{a}_{ss'} + \gamma V(s') \right]$$

[Use the optimal value fn of step1 to find V(s')]

22) 'Extract the optimal policy, by selecting the action that has the max Q value is a state.

$$\pi^* = \underset{a}{\operatorname{argmax}}\, Q(s,a)$$

for state A :

for State A :-

$$Q(A,0) = P_{AA}^{0}\left[R_{AA}^{0} + \gamma V(A)\right] + P_{AB}^{0}\left[R_{AB}^{0} + \gamma V(B)\right] + P_{AC}^{0}\left[R_{AC}^{0} + \gamma V(C)\right]$$

$$= 0.1\left[0 + 1.95\right] + 0.8(-1 - 0.72) + 0.1\left[1 + 1.3\right]$$

$$= -0.951$$

$$Q(A,1) = 2.26.$$

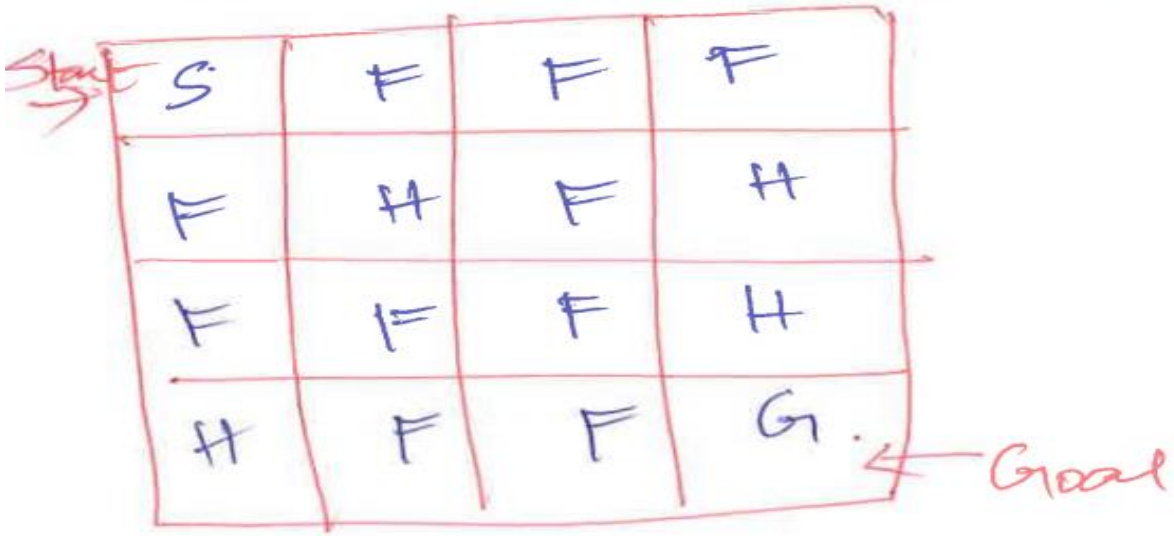
∴ Q table is

State	Action	Value
A	0	−0.95
A	1 ✓	2.26 ✓
B	0	−0.5
B	1 ✓	0.5 ✓
C	0	−1.1
C	1 ✓	1.4 ✓

from this Q table, pick the action in each state that has the max. value as an optimal policy.

∴ In state A, the optimal policy is action 1, i.e, moving down..

Solving the frozen lake problem with value iteration



S .. starting state

F .. frozen state

H .. hole state

G .. goal state.

States are encoded from 0 to 15.

Actions:

left	0
down	1
right	2
up	3

Step1: Compute the optimal value fn.

```
def value_itr(env):
    num_itrs = 1000
    threshold = 1e-20
    gamma = 1.0
    value_table = np.zeros(env.observation_space.n)
    for i in range(num_itrs):
        updated_value_table = np.copy(value_table)
        for s in range(env.observation_space.n):
```

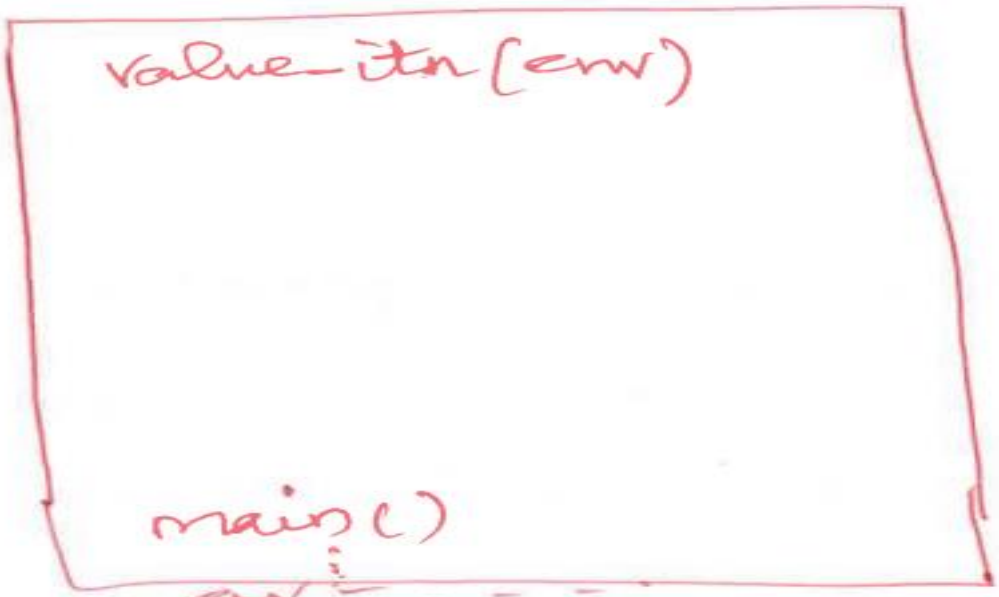
Slides Prepared by Dr J Alamelu Mangai

```
q_values = [
        sum([prob * (r + gamma * updated_value_table[s_
            for prob, s_, r, _ in env.P[s][a])
                for a in range(env.action_space.n)]
value_table[s] = max(q_values)

if (np.sum(np.fabs(updated_value_table -
                   value_table)) <= threshold):
    break

return value_table.
```

Step 2: finding optimal policy: -

```
def extract_policy ( value_table):
    gamma = 1.0
    policy = np.zeros( env.observation_space.n)
    for s in range ( env.observation_space.n):
        Q_values = [ sum([prob * (r + gamma * value_table[s_
                    for prob, s_, r, _ in env.P[s][a]])
                    for a in range ( env. action_space.n)]
    policy[s] = np.argmax( np.array [Q_values])
return policy
```

#Flow diagram:-




```
def value_iteration(env):
    :
    return value_table
def extract_policy(value_table):
    return policy
main()
// initialize the env
optimal_value_fn = value_iteration(env)
optimal_policy = extract_policy(optimal_value_fn)
Print(optimal_policy)
```

Policy iteration algorithm

- LO1: Describe the policy iteration algorithm to find an optimal policy
- LO2: Given the model dynamics of a particular state in a RL environment, apply the policy iteration algorithm, to find an optimal policy
- LO3: Differentiate the RL environments where DP can/cannot be applied.

Policy iteration Method:-

~~1. Compute the optimal value fn, using~~

On the value iteration method:

1. Compute the optimal value fn iteratively, by taking the max over the Q fn (Q values)

2. Extract the optimal policy from the optimal value fn, got in step 1.

On the policy iteration method:

1. Compute the optimal value fn iteratively, by using the policy.

2. Extract the optimal policy from the optimal value fn, got in step 1. [this is the same policy that generated the optimal value fn]

How to find the value fn of a state for a given policy π ?

If π is stochastic:

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

If π is deterministic:

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

(or)

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

Assume the policy is deterministic:

1. Start with a random policy π_0.
2. Initialize the value fn (table) of all states to zeros.
3. Use π_0, to find an optimal value table iteratively V^{π_0}. [This will not be optimal value table, since it is generated from a random policy]
4. Use V^{π_0} to generate a policy π_1..

5. compute V^{π_1} using π_1. check if V^{π_1} is optimal. If so, stop. Else generate π_2 from V^{π_1}.

6. Use π_2 to generate V^{π_2}. If V^{π_2} is optimal stop. Else generate a new policy π_3 .. and so on.

$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \cdots \pi^* \rightarrow V^{\pi^*}$

↑ random policy

↑ optimal policy

How to decide the value function is optimal?

How to decide that the value fn is optimal?
If it doesn't change over iterations?

Since the value fn is generated from a policy, over a series of iterns,
If the policy doesn't change bet^n 2 consecutive iterations, then it is an optimal policy π^*.

The value fn generated from π^* is the optimal value fn V^{π^*}.

Slides Prepared by Dr J Alamelu Mangai

Pseudo-code:-

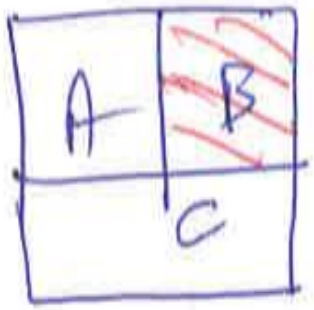
```
Policy = random-policy
for i in range(num_itrs):
    value_fn = compute_value_fn(policy)
    new_policy = extract_policy(value_fn)
    if policy == new_policy:
        break
    else
        Policy = new_policy.
```

Algorithm for Policy Iteration..

1. Initialize a random policy
2. Compute a value function iteratively, from this policy
3. Extract a new policy using the value function got from step 2.
4. If the extracted policy is same as the policy used in step 2, then stop; else send the extracted new policy to step 2 and repeat steps 2 to 4.

An example for Policy iteration

Ex: [Pg: 118]



States: A .. 0
B .. 1
C .. 2

Actions: 0 - left/right; 1 - up/down.

Goal: from A, reach C, without visiting B.

Given model dynamics of State A.

State s	Action a	Next state s'	$P^a_{ss'}$	$R^a_{ss'}$
A	0	A	0.1	0
A	0	B	0.8	-1
A	0	C	0.1	1
A	1	A	0.1	0
A	1	B	0.0	-1
A	1	C	0.9	1

Step 1:- Initialize a random policy:-

A → 1 ; B → 0 ; C → 1.

Step 2: Compute the value fn using this random policy

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

Step 3: ~~Initialize the value table of all states to~~ zero.

State	Value
A	0
B	0
C	0

Initial value table

$\gamma = 1$

~~step~~ Goal : find the V(s) for a state, only for those actions mentioned in the policy, π.

$\therefore$ V(A) . . only for action 1.

$$\therefore V(A) = P^1_{AA}\left[R^1_{AA} + \gamma V(A)\right] + P^1_{AB}\left[R^1_{AB} + \gamma V(B)\right] + P^1_{AC}\left[R^1_{AC} + \gamma V(C)\right]$$

$$= 0.1[0+0] + 0[-1+0] + 0.9[1+0]$$

$$= 0.9$$

Slides Prepared by Dr J Alamelu Mangai

Clearly from the model dynamics of states B and C, find V(B) for action 0 and V(C) for action 1.

Assume the value table from itr1 is

State	value
A	0.9
B	−0.2
C	0.1

This will not be optimal.

Itr2: find V(S) using the value table from Itr1.

$$V(A) = P'_{AA}\left[R'_{AA} + \gamma V(A)\right] + P'_{AB}\left[R'_{AB} + \gamma V(B)\right] + P'_{AC}\left[R'_{AC} + \gamma V(C)\right]$$

$$= 0.1\left[0 + 0.9\right] + 0\left[-1 - 0.2\right] + 0.9\left[1 + 0.1\right.$$

$$= 1.08$$

111 Lastly keep & find V(B) and V(C). Assume

value table from itr2 is

State	value
A	1.08
B	−0.5
C	0.5

value table from itr3 is

A	1.45
B	−0.9
C	0.6

value table from itr4 is

A	1.45
B	−0.9
C	0.6

∴ No change the ~~optimal~~ final value table ← is ~~the~~

Step3: Extract a new policy using this value fn from step2:

∴ policy ~~is greedy~~ tells us which is the correct action in each state. This is given by

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

for all (s, a) pairs of a state s.

~~for state A:-~~

∴ find $Q(A, 0)$: $P^{0}_{AA} \left[R^{0}_{AA} + \gamma V(A) \right] + P^{0}_{AB} \left[R^{0}_{AB} + \gamma V(B) \right)$

$$+ P^{0}_{AC} \left[R^{0}_{AC} + \gamma V(C) \right]$$

$$= 0.1 \left[0 + 1 \cdot A b \right] + 0.8 \left[-1 - 0.9 \right) +$$

~~$Q(A, A)$~~ : $0.1 \left[1 + 0.61 \right]$

$$= -1.21.$$

$$Q(A,1) = P_{AA}^{1}\left[R_{AA}^{1} + \gamma V(A)\right] + P_{AB}^{1}\left[R_{AB}^{1} + \gamma V(B)\right]$$
$$+ P_{AC}^{1}\left[R_{AC}^{2} + \gamma V(C)\right]$$

$$= 0.1\left[0 + 1 \cdot A6\right] + 0.0\left[-1 - 0.9\right] + 0.9\left[1 + 0.6\right]$$

$$= 1.59$$

Similarly do for states B and C for all actions.

∴ Q table is

State	Action	Value
A	0	-1.21
A	1 ✓	1.59 ✓
B	0 ✓	0.1 ✓
B	1	0.0
C	0 ✓	0.5 ✓
C	1	0.0

from this Q table, pick the action in each state, that gives the max. Value. This gives a new policy.

$A \rightarrow 1$; $B \rightarrow 0$; $C \rightarrow 0$.

Step 4: check the new policy

If The extracted new policy from step (3) is same as the policy used in step (2), then stop: else goto step 2 with this new policy & repeat steps 2 to 4.

Solving the Frozen Lake Problem with policy iteration

main()

```
import gym
import numpy as np
env = gym.make('FrozenLake-v0')
optimal_policy = policy_itr(env)
print(optimal_policy)
```

```
def policy_itr(env):
    num_itrs = 1000
    // initialize policy to zero in all states
    policy = np.zeros(env.observation_space.n)

    for i in range(num_itrs):
        val_fn = compute_val_fn(policy)
        new_policy = extract_policy(val_fn)
        if (np.all(policy == new_policy)):
            break
        policy = new_policy
    return(policy)
```

```
def compute_val_fn(policy):
    num_itns = 1000
    threshold = 1e-20
    gamma = 1.0
    // init. val. table of all states to zero
    val_tab = np.zeros(env.observation_space.n)

    for i in range(num_itns):
        updated_val_tab = np.copy(val_tab)

        for s in range(env.observation_space.n):
            // find value of a state only for action 'a' that is in the policy
            a = policy[s]
```


Slides Prepared by Dr J Alamelu Mangai

// $V^{\pi}(s) = \sum_{s'} P^{a}_{ss'} [R^{a}_{ss'} + \gamma V^{\pi}(s')]$

```
val_tab[s] = sum(
        [prob * (r + gamma * updated_val_tab[s_]
            for prob, s_, r, _ in env.P[s][a]])

if (np.sum(np.fabs(updated_val_tab - val_tab)) <=
                                    threshold:
    break
return(val_tab)
```


ngai

```
def extract_policy(val_table):
    // policy is generated using Q(s,a), for all
    // actions 'a' in a state 's'; not with resp to any π.
    // Q(s,a) = Σ_{s'} P^a_{ss'} [ R^a_{ss'} + γ V(s') ]

    gamma = 1.0
    // init. policy of all states to zero.
    policy = np.zeros(env.observation_space.n)
    for s in range(env.observation_space.n):
        Q_values = [sum([prob * [r + gamma *
                                 val_tab[s_]]
                         for prob, s_, r, _ in env.P[s][a]])
                    for a in range(env.action_space.n)]

        policy[s] = np.argmax([np.array(Q_values))
        // this returns the index of max Q-value
    return policy.
```

Limitations of DP :-

DP is a model-based method to find the optimal policy. In both value & policy itn methods, we need to know the model dynamics [transn probabilities] and reward functions of E

Summary of Value-iteration algorithm

In value iteration:-

$$V^*(s) = \max_a Q^*(s,a)$$

1. find the optimal val fn of a state 's', by finding the max over all Q functions (s,a) of all actions in state 's'.

$$Q(s,a) = \sum_{s'} P^a_{ss'} \cdot [R^a_{ss'} + \gamma V(s')]$$

2. Extract the optimal policy from this optimal val fn

Slides Prepared by Dr J Alamelu Mangai

Summary of Policy-iteration algorithm

On Policy iteration:-

1. find the optimal val fn of a state 's', by ~~finding the~~ using the policy iteratively

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

Start with a random policy and find the value fn.

Pg:: (37)

2. ~~find~~ The policy ~~that~~ generated this optimal value fn, is the optimal policy.

∴ in both methods, we should know $P^{a}_{ss'}$.

Mangai