ARTIFICIAL INTELLIGENCE

21AM503

Anirudhan Adukkathayar C
Assistant professor,
AIML Dept.
NMAMIT

Syllabus..

Unit III

Contact Hours : 10

- **Reinforcement Learning:** Introduction, Passive reinforcement learning, Generalization in reinforcement learning, Applications of reinforcement learning,
- **Q-Learning Intuition:** Plan of attack, Bellman Equation, The Plan, Markov Decision Process, Policy vs Plan, Adding Living penalty, Temporal Difference

REINFORCEMENT LEARNING

In which we examine how an agent can learn from success and failure, from reward and punishment.

- what to do in the absence of labeled examples of what to do
- **Rewards** - served to **define optimal policies** in Markov decision processes (MDPs).
- **optimal policy** is a policy that **maximizes** the **expected total reward**
- The **task of reinforcement learning** is to use **observed rewards to learn** an **optimal** (or nearly optimal) **policy** for the environment.
- In many complex domains, reinforcement learning is the only feasible way to train a program to perform at high levels.

REINFORCEMENT LEARNING

Example

- in game playing, it is very hard for a human to provide accurate and consistent evaluations of large numbers of positions, which would be needed to train an evaluation function directly from examples.
- Instead, the program can be told when it has won or lost, and it can use this information to learn an evaluation function that gives reasonably accurate estimates of the probability of winning from any given position.
- Similarly, it is extremely difficult to program an agent to fly a helicopter; yet given appropriate **negative rewards** for **crashing**, wobbling, or deviating from a set course, an agent can learn to fly by itself.

REINFORCEMENT LEARNING

- agent does not know how the environment works or what its actions do, and we will allow for probabilistic action outcomes
- the **agent faces an unknown Markov decision process.**
- **agent designs**
- A **utility-based agent** learns a **utility function on states** and uses it to select actions that maximize the expected outcome utility.
- A **Q-learning agent** learns an **action-utility function**, or Q-function, giving the expected utility of taking a given action in a given state.
- A **reflex agent learns** a policy that maps directly from states to actions.

PASSIVE REINFORCEMENT LEARNING

- **passive learning,** where the agent's **policy is fixed** and the task is to **learn the utilities of states** (or state–action pairs); this could also involve learning a model of the environment.

- **active learning**, where the agent must also learn what to do.

- The principal issue is **exploration**: an agent must experience as much as possible of its environment in order to learn how to behave in it. how an agent can use inductive learning to learn much faster from its experiences.

PASSIVE REINFORCEMENT LEARNING

- a **passive learning agent** using a **state-based representation** in a **fully observable environment.**
- In passive learning, the agent's **policy π is fixed**: in state s, it always executes the action π(s).
- Its **goal** is simply to **learn how good the policy is**—that is, to **learn the utility function $U^{\pi}(s)$.**
- Figure 21.1 shows a policy for that world and the corresponding utilities.
- The passive learning agent **does not know** the transition model **P (s′ | s, a),** which specifies the probability of reaching state s′ from state s after doing action a;
- nor **does it know** the **reward function R(s)**, which specifies the reward for each state.

PASSIVE REINFORCEMENT LEARNING

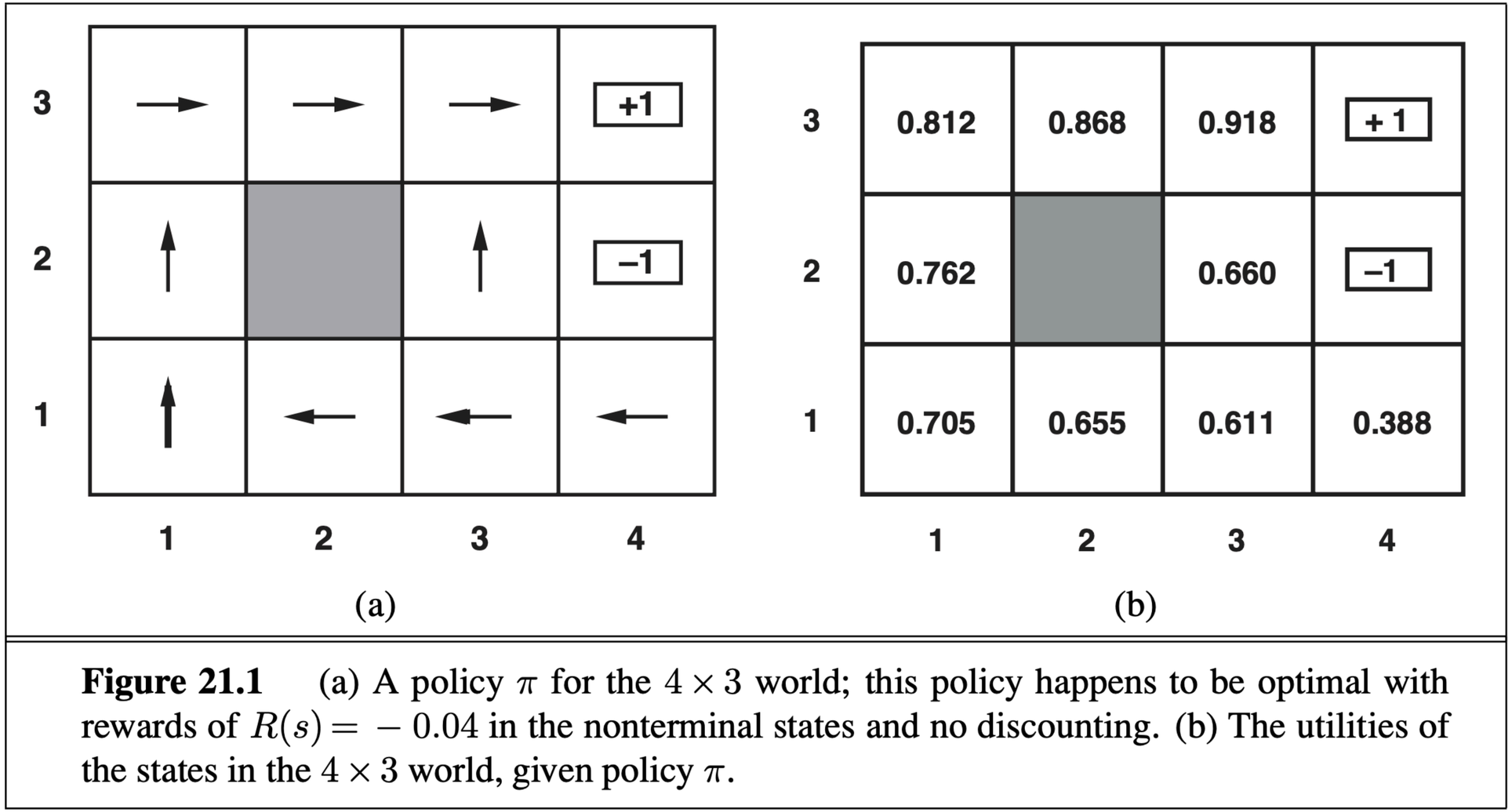

Figure 21.1 (a) A policy π for the 4×3 world; this policy happens to be optimal with rewards of $R(s) = -0.04$ in the nonterminal states and no discounting. (b) The utilities of the states in the 4×3 world, given policy π.

PASSIVE REINFORCEMENT LEARNING

- The agent **executes** a set of **trials** in the environment using its policy π.
- In each trial, the agent **starts in state (1,1)** and experiences a sequence of state transitions until it **reaches one of the terminal states**, (4,2) or (4,3).
- Its **percepts** supply both the **current state** and the **reward** received in that state.
- $$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
  $$(1,1)_{\textbf{-.04}} \rightsquigarrow (1,2)_{\textbf{-.04}} \rightsquigarrow (1,3)_{\textbf{-.04}} \rightsquigarrow (2,3)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (3,3)_{\textbf{-.04}} \rightsquigarrow (4,3)_{\textbf{+1}}$$
  $$(1,1)_{\textbf{-.04}} \rightsquigarrow (2,1)_{\textbf{-.04}} \rightsquigarrow (3,1)_{\textbf{-.04}} \rightsquigarrow (3,2)_{\textbf{-.04}} \rightsquigarrow (4,2)_{\textbf{-1}} \ .$$

PASSIVE REINFORCEMENT LEARNING

- Note that each state percept is subscripted with the reward received.
- The **object** is to **use the information about rewards** to l**earn the expected utility** $U^{\pi}(s)$ associated with each nonterminal state s.
- **The utility is defined to be the expected sum of (discounted) rewards obtained if policy π is followed**.
- $$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right] \qquad (21.1)$$

  where $R(s)$ is the reward for a state, S_t (a random variable) is the state reached at time t when executing policy π, and $S_0 = s$. We will include a **discount factor** γ in all of our equations, but for the 4×3 world we will set $\gamma = 1$.

PASSIVE REINFORCEMENT LEARNING

Direct utility estimation

- A simple method for **direct utility estimation** was invented in the late 1950s in the area of **adaptive control theory** by Widrow and Hoff (1960).
- the **utility** of a state is the **expected total reward from that state onward** (called the **expected reward-to-go**), and each trial provides a sample of this quantity for each state visited.
- **at the end of each sequence,** the algorithm calculates the **observed reward-to-go for each state** and **updates** the **estimated utility** for that state accordingly, just by keeping a running average for each state in a table.
- In the **limit of infinitely many trials,** the sample average will **converge** to the true expectation

PASSIVE REINFORCEMENT LEARNING

Direct utility estimation

- It is clear that direct utility estimation is just an **instance of supervised learning** where each example has the **state as input** and the observed **reward-to-go as output.**

- This means that we have **reduced reinforcement learning** to a standard **inductive learning problem**

- it **misses** a very important source of information, namely, the fact that the **utilities of states are not independent!**

PASSIVE REINFORCEMENT LEARNING

Direct utility estimation

- **By ignoring the connections between states, direct utility estimation misses opportunities for learning**
- the second of the three trials given earlier reaches the state (3,2), which has not previously been visited.
- The next transition reaches (3,3), which is known from the first trial to have a high utility.
- The Bellman equation suggests immediately that (3,2) is also likely to have a high utility, because it leads to (3,3), but direct utility estimation learns nothing until the end of the trial.
- More broadly, we can view direct utility estimation as searching for U in a hypothesis space that is much larger than it needs to be, in that it includes many functions that violate the Bellman equations

PASSIVE REINFORCEMENT LEARNING

Direct utility estimation - Theory Slide simplified

- the agent executes a sequence of trials or runs (sequences of states-actions transitions that continue until the agent reaches the terminal state).
- Each trial gives a sample value and the agent estimates the utility based on the samples values.
- Can be calculated as running averages of sample values.
- The main drawback is that this method makes a wrong assumption that state utilities are independent while in reality they are Markovian.
- Also, it is slow to converge.

PASSIVE REINFORCEMENT LEARNING

Direct utility estimation - Theory Slide simplified

- Suppose we have a 4x3 grid as the environment in which the agent can move either Left, Right, Up or Down(set of available actions).
- An example of a run
- $(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$
- Total reward starting at (1,1) = 0.72

PASSIVE REINFORCEMENT LEARNING

Adaptive dynamic programming

- An adaptive dynamic programming (or **ADP) agent** takes advantage of the **constraints among the utilities of states by learning the transition model** that connects them and **solving** the corresponding Markov decision process**(MDP)** using a **dynamic programming method**
- For a passive learning agent, this means **plugging** the learned transition model **P (s′ | s, π(s))** and the **observed rewards R(s)** into the Bellman equations to calculate the utilities of the states.
- we can adopt the approach of **modified policy iteration** using a simplified value iteration process to update the utility estimates after each change to the learned model.
- Because the model usually changes only slightly with each observation, the value iteration process can use the previous utility estimates as initial values and should converge quite quickly.

PASSIVE REINFORCEMENT LEARNING

Adaptive dynamic programming

- The process of learning the model itself is easy, because the environment is fully observable.
- This means that we have a supervised learning task where the input is a state–action pair and the output is the resulting state.
- In the **simplest case**, we can represent the t**ransition model as a table of probabilities**.
- We keep track of **how often each action outcome occurs** and **estimate the transition probability P (s′ | s, a)** from the frequency with which s′ is reached when executing a in s.
- For example, in the three trials given on page 832, Right is executed three times in (1,3) and two out of three times the resulting state is (2,3), so P ((2, 3) | (1, 3), Right) is estimated to be 2/3.

PASSIVE REINFORCEMENT LEARNING

Adaptive dynamic programming - Theory simplified

- ADP is a smarter method than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.

- It can be solved using value-iteration algorithm.

- The algorithm converges fast but can become quite costly to compute for large state spaces.

- ADP is a model based approach and requires the transition model of the environment.

PASSIVE REINFORCEMENT LEARNING

Adaptive dynamic programming - Theory simplified

- The utility of each state equals **its own reward** plus the **expected utility of its successor states. That is, the utility values obey the Bellman equations for a fixed policy**

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi(s)) U^{\pi}(s')$$

- Where R(s) = reward for being in state s,
- P(s'|s, π(s)) = transition model,
- γ = discount factor and Uπ(s) = utility of being in state s’

b) Adaptive Dynamic Programming (ADP) : ADP is a smarter method than Direct Utility Estimation as it runs trials to learn the model of the environment by estimating the utility of a state as a sum of reward for being in that state and the expected discounted reward of being in the next state.

PASSIVE REINFORCEMENT LEARNING

Temporal-difference learning

- Another way of bringing the Bellman equations to bear on the learning problem
- **use the observed transitions to adjust the utilities of the observed states so that they agree with the constraint equations.**
- when a **transition** occurs from state **s to state s′**, we apply the following update to $U^{\pi}(s)$:
- $U^{\pi}(s) \leftarrow U^{\pi}(s) + \alpha(R(s) + \gamma\, U^{\pi}(s') - U^{\pi}(s))$.

PASSIVE REINFORCEMENT LEARNING

Temporal-difference learning

- Here, **α** is the **learning rate parameter**. Because this update rule uses the **difference in utilities between successive states**, it is often called the **temporal-difference, or TD, equation**
- All temporal-difference methods **work by adjusting the utility estimates towards the ideal equilibrium** that holds locally when the utility estimates are correct
- While ADP adjusts the utility of s with all its successor states, TD learning adjusts it with that of a single successor state s'.
- TD is **slower in convergence** but much **simpler** in terms of **computation**.

PASSIVE REINFORCEMENT LEARNING

Temporal-difference learning

- **c) Temporal Difference Learning (TD):** TD learning does not require the agent to learn the transition model. The update occurs between successive states and agent only updates states that are directly affected.

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

Where α = learning rate which determines the convergence to true utilities.

While ADP adjusts the utility of s with all its successor states, TD learning adjusts it with that of a single successor state s'. TD is slower in convergence but much simpler in terms of computation.

GENERALIZATION IN REINFORCEMENT LEARNING

WHY FUNCTION APPROXIMATION IS REQUIRED

- A **state** is the combination of observable **features** or **variables**. Which means every time a feature or a variable has a new value, it results in a a new state.
- let's take a concrete example. Suppose an agent is in a **4x4 grid**, so the location of the of the agent on the grid is a feature. This gives 16 different locations meaning 16 different states.
- But that's not all, suppose the **orientation** (north, south, east, west) is also a feature. This gives 4 possibilities for each location, which makes the number of states to 16*4 = 64. Furthermore if the agent has the possibility of using **5 different tools** (including "no tool" case), this will grow the number of states to 64 * 5 = 320.
- Hence this may results in infinitely more number of states with changes in the value of feature

GENERALIZATION IN REINFORCEMENT LEARNING

WHY FUNCTION APPROXIMATION IS REQUIRED

- One way to **represent those states** is by creating a **multidimensional array** such as V[row, column, direction, tool]. Then we either query or compute a state.
- For example V[1, 2, north, torch] represents the state where the agent is at row 1, column 2, looking north and holding a torch. The value inside this array cell tells how valuable this state is.
- So once we have the set of states we can assign a value-state function for each state.
- Needless to say that the amount of memory needed to accommodate the **number of state** is **huge** and the amount to **time** needed to compute the value of each state is also prohibitive.

GENERALIZATION IN REINFORCEMENT LEARNING

WHY FUNCTION APPROXIMATION IS REQUIRED

- **Solutions**
- It is always useful to keep in mind what we are trying to do, because with all the details we might lose sight.
- The idea is that we want to find the value of each state/action, in an environment, so that the agent follows the optimum path that collects the maximum rewards.
- In order to address this shortcoming, we can adopt a **new approach based** on the **features of each state**.
- The aim is to **use** these **set of features** to **generalize the estimation of the value** at **states** that have **similar features.**
- We used the word **estimation** to indicate that this approach will never find the true value of a state, but an **approximation** of it.
- Despite this seemingly inconvenient result, however this will achieve **faster computation** and much more generalisations.
- The **methods** that compute these approximations are called **Function Approximators.**

GENERALIZATION IN REINFORCEMENT LEARNING

- **utility functions** learned by the agents are **represented** in **tabular form** with **one output** value for each input tuple
- Such an approach works reasonably well for small state spaces
- **time to convergence** and (for ADP) **the time per iteration** increase rapidly as the **space gets larger**.
- This suffices for two-dimensional maze-like environments
- Chess - 10^{40} States

GENERALIZATION IN REINFORCEMENT LEARNING

- One way to handle such problems is to use function approximation, which simply means using any sort of representation for the Q-function other than a lookup table
- The representation is viewed as approximate because it might not be the case that the true utility function or Q-function can be represented in the chosen form
- evaluation function for **chess** that is represented as a **weighted linear function of a set of features** (or basis functions) f1, . . . , fn:
- $\hat{U}_\theta(s) = \theta_1 f_1(s) + \theta_2 f_2(s) + \cdots + \theta_n f_n(s)$.
- A reinforcement learning algorithm can learn values for the **parameters $\theta = \theta_1, \ldots, \theta_n$** such that the evaluation function $\hat{U}_\theta$ approximates the **true utility function**. Instead of, say, 10^{40} values in a table, this function approximator is characterized by, say, n=20 parameters— an enormous compression.

GENERALIZATION IN REINFORCEMENT LEARNING

- Although no one knows the true utility function for chess, no one believes that it can be represented exactly in 20 numbers.
- If the approximation is good enough, however, the agent might still play excellent chess
- **Function approximation** makes it practical to represent **utility functions for very large state spaces**, but that is not its principal benefit.
- The compression achieved by a function approximator allows the learning agent to **generalize from states it has visited to states it has not visited.**
- That is, the most important aspect of function approximation is not that it requires less space, but that it **allows** for **inductive generalization over input states.**

GENERALIZATION IN REINFORCEMENT LEARNING

- On the flip side, of course, there is the problem that there could fail to be any function in the chosen hypothesis space that approximates the true utility function sufficiently well.
- As in all inductive learning**, there is a tradeoff** between the **size of the hypothesis space** and the **time it takes to learn the function.**
- A **larger hypothesis** space increases the likelihood that a **good approximation can be found**, but also means that **convergence** is likely to be **delayed**
- **Function approximation** can also be very helpful for **learning a model of the environment.**

APPLICATIONS OF REINFORCEMENT LEARNING

- consider applications in game playing, where the transition model is known and the goal is to learn the utility function, and in robotics, where the model is usually unknown.
-

APPLICATIONS OF REINFORCEMENT LEARNING

APPLICATIONS OF REINFORCEMENT LEARNING

- **Robotics:**
- RL is used in Robot navigation, Robo-soccer, walking, juggling, etc.
- **Control:**
- RL can be used for adaptive control such as Factory processes, admission control in telecommunication, and Helicopter pilot is an example of reinforcement learning.
- **Game Playing:**
- RL can be used in Game playing such as tic-tac-toe, chess, etc.
- **Chemistry:**
- RL can be used for optimizing the chemical reactions.
- **Business:**
- RL is now used for business strategy planning.
- **Manufacturing:**
- In various automobile manufacturing companies, the robots use deep reinforcement learning to pick goods and put them in some containers.
- **Finance Sector:**
- The RL is currently used in the finance sector for evaluating trading strategies.