

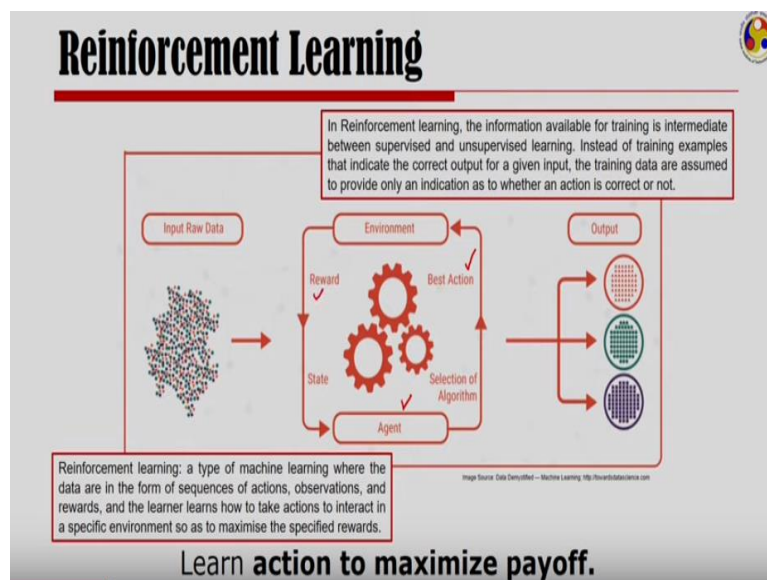
**Fundamentals of Artificial Intelligence**  
**Prof. Shyamanta M. Hazarika**  
**Department of Mechanical Engineering**  
**Indian Institute of Technology-Guwahati**

**Lecture - 32**  
**Reinforcement Learning**

Welcome to fundamentals of artificial intelligence. We are in the last week of our course. Today, we look at reinforcement learning. Having looked at supervised learning, which covers learning under labeled data and unsupervised learning, which is about finding structures underlying the data without any supervision, we look at the third category of machine learning which is learning based on rewards that the agent derives from the environment based on his actions.

This lecture today on reinforcement learning is to be seen as a continuation of our lectures on decision making particularly the sequential decision problem and the lecture on making complex decisions. First, we would introduce the very idea of reinforcement and then we would only look at one particular idea of approximating the value function using what is called Q-learning.

**(Refer Slide Time: 02:33)**



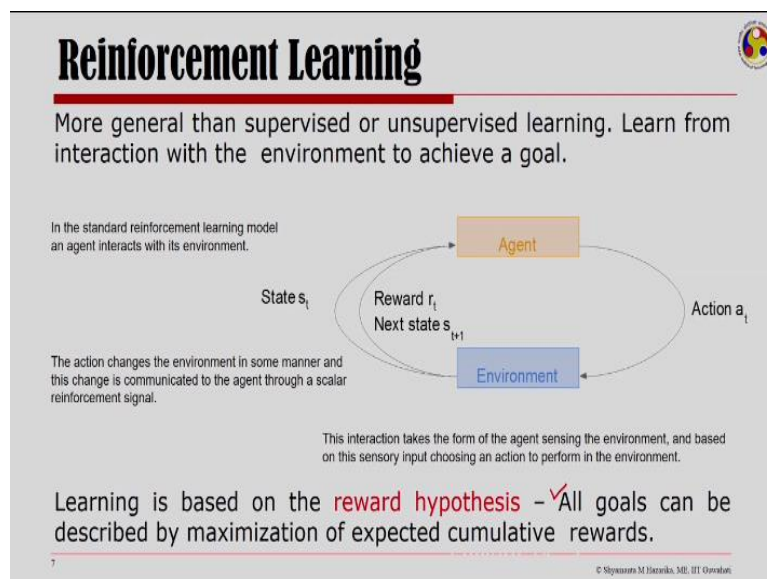
Now, if you recall from our very first discussion on machine learning, we had seen that reinforcement learning involves learning action to maximize payoff and reinforcement learning is where the data are in the form of sequence of actions,

observations and rewards and the learner learns how to take actions to interact in a specific environment so as to maximize the specified rewards.

In reinforcement learning the information available for training is actually intermediate. It is in between supervised and unsupervised learning. So instead of training examples that indicate the correct output for a given input, the training data are assumed to provide an indication as to whether an action is correct or not. And in reinforcement learning, we have an agent which picks up an action and derives a reward from the environment.

And based on the payoff, his concept of which actions to prefer is formulated or reinforced.

**(Refer Slide Time: 04:10)**



Now reinforcement learning is more general than supervised or unsupervised learning. You learn from interaction with the environment to achieve a goal. In the standard reinforcement learning model, an agent needs to interact with the environment as we have seen in the previous slide. Thereafter this interaction takes the form of the agent sensing the environment and based on the sensory input choosing an action to perform in the environment.

This action that he performs in the environment changes the environment in some manner and this change is communicated to the agent again through a scalar reinforcement signal which is called the reward. Thereafter, we enter the next step

which is a  $t + 1$ . Learning is based on the reward hypothesis. That is all goals can be described by maximization of expected cumulative rewards.

Now we have introduced this idea of what is a Markov decision process or what is a partially observable Markov decision problem while we were discussing sequential decision making and therefore, here we will only quickly introduce the components of a RL agent. A reinforcement agent may include one or more of these components. It contains something called a policy which is the agent's behavior function.

**(Refer Slide Time: 06:26)**

## Components of a RL Agent

An RL agent may include one or more of these components:

- Policy: agent's behaviour function
- Value function: how good is each state and/or action
- Model: agent's representation of the environment

- A policy is the agent's behaviour
- It is a map from state to action, e.g.
- Deterministic policy:  $a = \pi(s)$
- Stochastic policy:  $\pi(a|s) = P[A_t = a | S_t = s]$

- Value function is a prediction of future reward
- Used to evaluate the goodness/badness of states
- And therefore to select between actions, e.g.

$$v_\pi(s) = \mathbb{E}_\pi [R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s]$$

- A model predicts what the environment will do next
- $\mathcal{P}$  predicts the next state
- $\mathcal{R}$  predicts the next (immediate) reward, e.g.

$$P_{ss'}^a = P[S_{t+1} = s' | S_t = s, A_t = a]$$

$$R_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$$

12 © Shyamantan M Hazareika, M.E., IIT Guwahati

And then we have a value function which is about how good is each state and/or action and then we have a model which is the agent's representation of the environment. A policy is the agent's behavior. It is a map from state to action. So you could have certain things which are deterministic leading to a deterministic policy. Or you could have a stochastic policy.

This we have covered in our lecture on sequential decision problems and while we were discussing making complex decisions. The value function if you recall, is a prediction of future reward. So it is used to evaluate the goodness or badness of states and therefore, to select between actions. And a model is one which predicts what the environment will do next. So there would be predictions of the next state or there could be predictions of the next reward.

**(Refer Slide Time: 07:43)**

## Elements of RL Problem



### ✓. The Environment

- Every RL system learns a mapping from situations to actions by trial-and-error interactions with a dynamic environment.
- This environment must at least be partially observable; the observations may come in the form of sensor readings, symbolic descriptions, or possibly "mental" situations.
- If the RL system can observe perfectly all the information in the environment that might influence the choice of action to perform, then the RL system chooses actions based on true "states" of the environment.
- This ideal case is the best possible basis for reinforcement learning and, in fact, is a necessary condition for much of the associated theory.

13

© Shyamam M Hazare, M.E., IIT Guwahati

We will more closely look at rather the elements of our reinforcement learning problem. So by this time you may have come to the realization that reinforcement learning involves an agent, environment, an action and some feedback from the environment. So the elements of a reinforcement learning problem include the environment. Every reinforcement learning system actually learns a mapping from situation to actions by trial and error interaction with a dynamic environment.

This environment must be at least partially observable. The observations may come in the form of sensor reading, symbolic descriptions, or possibly mental situations. If the reinforcement learning system can observe perfectly all the information in the environment that might influence the choice of action to perform, then the learning system can choose actions based on the true states of the environment.

This ideal case is actually the best possible basis for reinforcement learning and in fact is a necessary condition for most of the associated theory.

**(Refer Slide Time: 09:20)**

# Elements of RL Problem



## 2. The Reinforcement Function

RL systems learn a mapping from situations to actions by trial-and-error interactions with a dynamic environment.

- The "goal" of the RL system is defined using the concept of a reinforcement function, which is the exact function of future reinforcements the agent seeks to maximize.
- There exists a mapping from state/action pairs to reinforcements; after performing an action in a given state the RL agent will receive some reinforcement (reward) in the form of a scalar value.
- The RL agent learns to perform actions that will maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state.

14

© Shyamanta M Hazra, IIT Guwahati

The second element of a reinforcement learning problem is the reinforcement function. Now the reinforcement system learn a mapping from situations to action by trial and error interaction with a dynamic environment. And the goal of the reinforcement learning system is defined using the concept of a reinforcement function. A reinforcement function is the exact function of future reinforcements, that the agent seeks to maximize.

There exist a mapping from state action pairs to reinforcements. After performing an action in a given state, the reinforcement agent will receive some reinforcement or what we call rewards in the form of a scalar value. The reinforcement learning agent learns to perform actions that will maximize the sum of the reinforcements received when starting from some initial state and proceeding to a terminal state.

**(Refer Slide Time: 10:36)**

## Elements of RL Problem



### 3. The Value Function

Having the environment and the reinforcement function defined; the question now is how the agent learns to choose "good" actions.

- A *policy* determines which action should be performed in each state; a policy is a mapping from states to actions.
- The *value* of a state is defined as the sum of the reinforcements received when starting in that state and following some fixed policy to a terminal state.
- The *optimal policy* would be the mapping from states to actions that maximizes the sum of the reinforcements. The value of a state is dependent upon the policy.
- The *value function* is a mapping from states to state values and can be approximated using any type of function approximator (e.g., multilayered perceptron, memory based system, radial basis functions, look-up table, etc.).

15

© Shyamanta M Hazra, MIT, IIT Guwahati

The third element of a reinforcement learning problem is the value function. Now having the environment defined either as a Markov decision problem or a partially observable Markov decision problem and the reinforcement function defined as well. The question now is how the agent learns to choose good actions? A policy determines which action should be performed in a state.

A policy if you recall is a mapping from states to actions. And the value of a state is defined as the sum of reinforcements received when starting in that state and following some fixed policy to a terminal state. The optimal policy would be the mapping from states to action that maximizes the sum of the reinforcements and the value of a state is dependent upon the policy.

Now the value function is a mapping from the state to state values and can be approximated using any type of function approximator. So people have used multilayered perceptron, memory based systems, radial basis functions or look-up tables for the value function approximation.

**(Refer Slide Time: 12:19)**

# The Reinforcement Function



- There are **at least three noteworthy classes** often used to construct reinforcement functions that properly define the desired goals.

## 1. Pure Delayed Reward and Avoidance Problems

- In Pure Delayed Reward class of functions the reinforcements are all zero except at the terminal state.
- The sign of the scalar reinforcement at the terminal state indicates whether the terminal state is a goal state (a reward) or a state that should be avoided (a penalty).

Let us take a moment and look at more closely the reinforcement function. For it is the reinforcement function that tells us how the agent proceeds further in an environment that he does not know anything about. There are at least three noteworthy classes often used to construct reinforcement function that properly define the desired goals. The first of them is the pure delayed reward and avoidance problem.

In pure delayed reward class of functions, the reinforcements are all zero except at the terminal state. That is, you get the reward only at the terminal state. So the sign of the scalar reinforcement at the terminal state indicates whether the terminal state is a goal state that is you get a positive reward or a state that should be avoided and you get a penalty.

**(Refer Slide Time: 13:30)**

# The Reinforcement Function



- There are **at least three noteworthy classes** often used to construct reinforcement functions that properly define the desired goals.

## 1. Pure Delayed Reward and Avoidance Problems



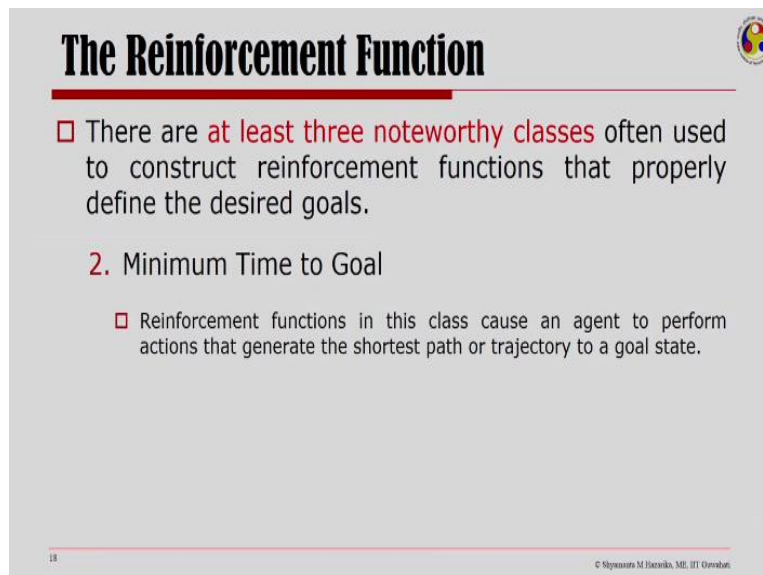
Example of a pure delayed reward reinforcement function: Standard cart-pole or inverted pendulum problem. A cart supporting a hinged, inverted pendulum is placed on a finite track. The goal of the RL agent is to learn to balance the pendulum in an upright position without hitting the end of the track. The situation (state) is the dynamic state of the cart pole system. Two actions are available to the agent in each state: move the cart left, or move the cart right. The reinforcement function is zero everywhere except for the states in which the pole falls or the cart hits the end of the track, in which case the agent receives a -1 reinforcement.

Now here is an example of a pure delayed reward in reinforcement function. A standard cart-pole or inverted pendulum problem is often one where we would use a pure delayed reward and avoidance. Here a cart supporting a hinge inverted pendulum is placed on a finite track. And the goal of the reinforcement learning agent is to learn to balance the pendulum in an upright position without hitting the end of the track.

So the state is the dynamic state of the cart-pole system. Two actions are available to the agent in each state. Either it can move the cart left or move the cart right. Now the reinforcement function is zero everywhere except for the states in which the ball falls or the card hits the ends of the track in which case we need to penalize the agent and therefore, the agent receives a -1 reinforcement.

So the standard pole or inverted pendulum problem is an example where you could use pure delayed reward reinforcement function.

**(Refer Slide Time: 15:02)**



**The Reinforcement Function**

- There are **at least three noteworthy classes** often used to construct reinforcement functions that properly define the desired goals.

**2. Minimum Time to Goal**

- Reinforcement functions in this class cause an agent to perform actions that generate the shortest path or trajectory to a goal state.

18 © Shyamanta M Hazra, M.E., IIT Guwahati

The second type of reinforcement function is the minimum time to goal function. Reinforcement functions in this class actually cause an agent to perform actions that generate the shortest path or trajectory to a goal state.

**(Refer Slide Time: 15:22)**



# The Reinforcement Function



- There are **at least three noteworthy classes** often used to construct reinforcement functions that properly define the desired goals.

## 2. Minimum Time to Goal



An example is one commonly known as the "Car on the hill" problem. The problem is defined as that of a stationary car being positioned between two steep inclines. The goal of the RL agent is to successfully drive up the incline on the right to reach a goal state at the top of the hill. The state of the environment is the car's position and velocity. Three actions are available: forward thrust, backward thrust, or no thrust at all. Agent learn to use momentum to gain velocity to successfully climb the hill. The reinforcement function is  $-1$  for ALL state transitions except the transition to the goal state, in which case a zero reinforcement is returned.

19

© Shantanu M. Hareishi, M.E., IIT Guwahati

An example is one commonly known as the car on the hill problem. Now here the problem is defined as that of a stationary car being positioned between two steep inclines. And the goal of the reinforcement learning agent is to successfully drive up the incline on the right to reach a goal state at the top of the hill. The state of the environment is the car's position as well as the velocity.

Now three actions are available to the agent. He could give a forward thrust, a backward thrust or no thrust at all. And somehow the agent need to learn to use momentum to gain velocity to successfully climb the hill. Now the reinforcement function that returns a reward here is  $-1$  for all state transitions, except the transition to the goal state in which case actually a zero reinforcement is returned. So this is the minimum time to goal reinforcement function.

**(Refer Slide Time: 16:48)**

# The Reinforcement Function



- There are **at least three noteworthy classes** often used to construct reinforcement functions that properly define the desired goals.

## 3. Games

- An alternative reinforcement function would be used in the context of a game environment - two or more players with opposing goals.
- RL system can learn to generate optimal behavior for the players; finding the minimax, or saddlepoint of the reinforcement function.
- Agent would evaluate the state for each player and would choose an action independent of the other players action.
- Actions are chosen independently and executed simultaneously, the RL agent learns to choose actions for each player that would generate the best outcome for the given player in a "worst case" scenario.

20

© Shantanu M. Harsola, M.E., IIT Guwahati

The third reinforcement function is what we encounter in games. And here an alternative reinforcement function is used when we have two or more players with opposing goals. Now the reinforcement learning system can learn to generate optimal behavior for the players finding the minimax. Recall when we were discussing game trees, we saw the minimax to be the value that you could get to the top of the game tree based on a strategy.

So here the reinforcement learning agent can learn to generate optimal behavior for the players finding the minimax or what we call the saddlepoint of the reinforcement function. Now the agent would evaluate state for each player and would choose an action which is independent of the other players action. Actions are chosen independently and they are executed simultaneously under such a reinforcement function.

The reinforcement learning agent learns to choose actions for each player that would generate the best outcome for the given player in a worst case scenario.

**(Refer Slide Time: 18:18)**

## Reinforcement Learning



- In our discussion of **Markov Decision Problems**, we assumed that we knew the agent's reward function,  $R$ , and a **model of how the world works**, expressed as the transition probability distribution.
- In reinforcement learning, we would like an agent to **learn to behave well in an MDP world**, but **without** knowing anything about reward function or the **transition probability distribution** when it starts out.

Parameter Estimation - you can estimate the next-state distribution  $P(s'|s, a)$  by counting the number of times the agent has taken action  $a$  in state  $s$  and looking at the proportion of the time that  $s'$  has been the next state. Similarly, you can estimate  $R(s)$  just by averaging all the rewards you've received when you were in state  $s$ .

Now having discussed the reinforcement function, let us quickly look at how is reinforcement learning related to what I have been emphasizing that this is some form of continuation of what we covered in sequential decision making or making complex decisions. So in our discussion of Markov decision problems, we assumed that we knew the agents reward function  $R$  and the model of how the world works expressed as the transition probability distribution.

On the other hand, in reinforcement learning, we would like an agent to behave well in a Markov decision problem. But, here without knowing anything, so we would not know anything about the reward function or the transition probability distribution when it starts out and this is what makes it interesting.

So we have a Markov decision problem, but we neither know anything about the reward function nor we know anything about the transition probabilities when we start out in reinforcement learning. So what can best be done is what is referred to as parameter estimation. One of the options could be to estimate the next states distribution and this could be done by counting the number of times the agent has taken an action in a particular state and looking at the proportion of time that  $s'$  has been the next step.

So similarly, we could estimate the reward at  $s$  by averaging all the rewards you have received when you were in state  $s$ .

**(Refer Slide Time: 20:22)**

## Approximating the Value Function



- Reinforcement learning is a difficult problem because the learning system may perform an action and not be told whether that action was good or bad.
- Initially, the approximation of the optimal value function is poor. In other words, the mapping from states to state values is not valid.
- The **primary objective of learning is to find the correct mapping**. Once this is completed, the optimal policy can easily be extracted.
  - Q-learning - finds a mapping from state/action pairs to values; one of the most successful approaches to RL.

So approximating the value function is what is the most difficult of the problems in reinforcement learning and this is difficult because the learning system may perform an action and it would not be told whether that action was good or bad. Initially, the approximation of the optimal value function is poor. In other words, the mapping from states to state values would not be valid at all.

The primary objective of learning is to find the correct mapping and once this is completed, the optimal policy can easily be executed. So here we would today look at Q-learning which is an algorithm of finding a mapping from state action pairs to values and this is one of the most successful approaches to reinforcement learning.

Here one thing to take note of before I proceed to discuss Q-learning is the very fact that the very idea of introducing this final three lectures in machine learning on reinforcement learning thereafter, learning in neural networks and finally, some introduction to deep learning is with the view to give an exposure to these topics rather than mathematical regard.

For the interested ones we would suggest that they would look up the books that I have referred to in my first lecture on machine learning. So here we will give you the very fundamental idea of Q-learning and see how reinforcement learning happens.

**(Refer Slide Time: 22:38)**

## Q-Learning



- Deterministic Markov decision process - the state transitions are deterministic
  - ✓ An action performed in state  $x_t$  always transitions to the same successor state  $x_{t+1}$ .
  - In a nondeterministic Markov decision process, a probability distribution function defines a set of potential successor states for a given action in a given state.
- If the MDP is non-deterministic, then value iteration requires that we find the action that returns the maximum expected value
  - ✓ The sum of the reinforcement and the integral over all possible successor states for the given action.

Q-learning is about getting to the mapping, but then what is so problematic about value iteration? Let us try to understand that before we move to Q-learning. So deterministic Markov decision process, the state transitions are deterministic. So if I have an action performed in state  $x_t$  and it always transitions to the same successor state  $x_{t+1}$ .

Now, in a non deterministic Markov decision process, a probability distribution function defines a set of potential successor states for a given action in a given state. If the Markov decision process is non-deterministic, then value iteration requires that we find the action that returns the maximum expected value. Now this could involve the sum of the reinforcement and the integral over all possible successor states for the given action.

**(Refer Slide Time: 23:46)**

## Q-Learning



- Theoretically, value iteration is possible in the context of non-deterministic MDPs.
  - Computationally impossible to calculate the necessary integrals without added knowledge or some degree of modification.
- Q-learning solves the problem of having to take the max over a set of integrals.
- Rather than finding a mapping from states to state values (as in value iteration), **Q-learning finds a mapping from state/action pairs to values** (called Q-values).
  - Q-value is the sum of the reinforcements received when performing the associated action and then following the given policy thereafter.

24

© Shyamanta M Bhattacharya, M.E., IIT Guwahati

Theoretically, such a value iteration is possible in the context of non-deterministic Markov decision problems. However, it is computationally impossible to calculate the necessary integrals without aided knowledge or some degree of modifications. Q-learning solves the problem of having to take the maximum over a set of integrals. Rather than finding a mapping from state to state values as done in value iteration, Q-learning finds a mapping from state action pairs to Q-values.

So these values that I get from state action pairs, these are called the Q-values. So Q-value is in fact the sum of the reinforcements received when performing the associated action and then following the given policy thereafter.

**(Refer Slide Time: 24:52)**

## Q-Function



- $Q^*(s,a)$  is the **expected** discounted future reward for starting in state  $s$ , taking action  $a$ , and continuing optimally thereafter.

Assuming we have some way of choosing actions, now we're going to focus on finding a way to estimate the value function directly.

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

The Q value of being in state  $s$  and taking action  $a$  is the immediate reward,  $R(s)$ , plus the discounted expected value of the future. We get the expected value of the future by taking an **expectation** over all possible next states,  $s'$ . In each state  $s'$ , we need to know the value of behaving optimally. We can get that by choosing, in each  $s'$ , the action  $a'$  that maximizes  $Q^*(s', a')$ .

25


© Shyamanta M Bhattacharya, M.E., IIT Guwahati

The Q function is actually the expected discounted future reward for starting in state  $s$  taking action  $a$  and continuing optimally thereafter. That is, we have some way of choosing actions. So we are now going to focus on finding a way to estimate the value function directly and that is done by the Q function. So the Q value of being in state  $s$  and taking action  $a$  is the immediate reward  $R_s$  plus the discounted expected value of the future.

So we get the expected value of the future by taking an expectation over all possible next states. In each state, we need to know the value of behaving optimally. We can get that by choosing in each state the action that maximizes this value of Q.

**(Refer Slide Time: 26:06)**

## Q-Function



- $Q^*(s,a)$  is the expected discounted future reward for starting in state  $s$ , taking action  $a$ , and continuing optimally thereafter.
 

Assuming we have some way of choosing actions, now we're going to focus on finding a way to estimate the value function directly.

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

- If you know  $Q^*$ , then it's really easy to compute the optimal action in a state.
  - Take the action that gives the largest Q value in that state.
- When using  $V^*$ , it required knowing the transition probabilities to compute the optimal action, so this is considerably simpler.
- And it will be effective when the model is not explicitly known.

26
© Shyamant M. Harekha, MIT, IIT Guwahati

So what actually we have done here is replaced the very essence of getting to the value function by getting to a value which is referred to as the Q value. So if you know Q star, the optimal Q, then it is easy to compute the optimal action in a state. Take the action that gives the largest Q value in that state and recall when using optimal value and the value function, it required the knowledge of the transition probabilities to compute the optimal action.

So computing optimal action in Q learning is considerably simpler. And it will be effective when the model is not explicitly known.

**(Refer Slide Time: 26:59)**

# Q-Learning



- Q learning, which estimates the  $Q^*$  function directly, without estimating the transition probabilities.
  - Once we have  $Q^*$ , finding the best way to behave is easy.
- The learning algorithm deals with individual “pieces” of experience with the world.
  - One piece of experience is a set of the current state  $s$ , the chosen action  $a$ , the reward  $r$ , and the next state  $s'$ .
  - Each piece of experience will be folded into the  $Q$  values, and then thrown away!

27

© Shyamant M Hazarika, M.E., IIT Guwahati

So Q learning which estimates the optimal  $Q$  function directly without estimating the transition probabilities, let us find the best way to behave in a given environment. And the learning algorithm deals with individual pieces of experience with the world. So one piece of experience is a set of the current state  $s$ , the chosen action  $a$ , the reward  $r$  and the next state  $s$  prime. So each piece of experience will be folded into the  $Q$  values and then they would not be used again.

(Refer Slide Time: 27:45)

# Q-Learning



- A piece of experience in the world is  $(s,a,r,s')$

As in value iteration, by initializing the  $Q$  function arbitrarily. Zero is usually a reasonable starting point.

- Initialize  $Q(s,a)$  arbitrarily

- After each experience, update  $Q$ :
  - The basic form of the update looks like this. The parameter  $\alpha$  is a learning rate; usually it's something like 0.1. So, we're updating our estimate of  $Q(s,a)$  to be mostly like our old value of  $Q(s,a)$ , but adding in a new term that depends on  $r$  and the new state.

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha q(r,s')$$

$$q(r,s') = r + \gamma \max_a Q(s',a)$$

$q(r,s)$  is an example of the value of taking action  $a$  in state  $s$ . The actual reward,  $r$ , is a sample of the expected reward  $R(s)$ . And the actual next state,  $s'$ , is a sample from the next state distribution. And the value of that state  $s'$  is the value of the best action we can take in it, which is the max over  $a$  of  $Q(s',a)$ .

- Guaranteed to converge to optimal  $Q$  if the world is really an MDP

32

© Shyamant M Hazarika, M.E., IIT Guwahati

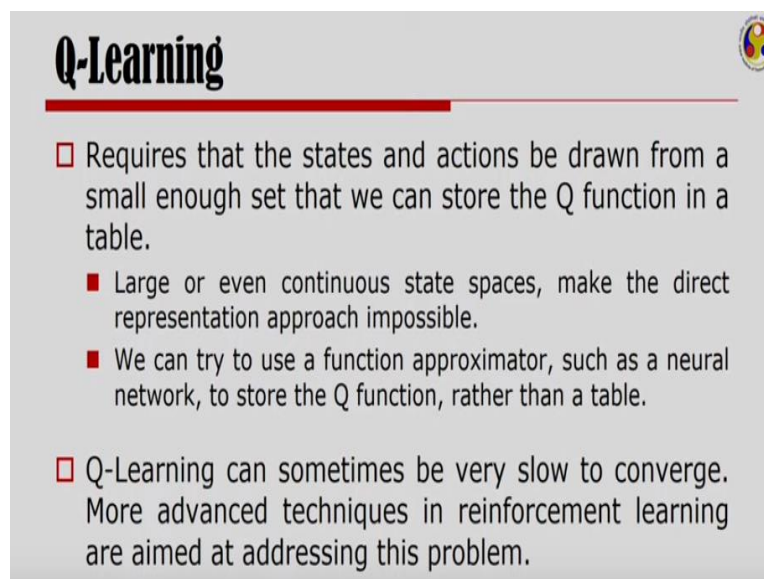
So here is the Q learning algorithm. I have a piece of experience in the world which is state  $s$ , action  $a$ , reward  $r$ , and the next state  $s$  prime. And I initialize arbitrarily, so as in value iteration, we initialize the  $Q$  function arbitrarily. So zero is usually a reasonable starting point. And thereafter, each experience you update  $Q$ . So here, when you update  $Q$ , the basic form of the update looks like this.



The parameter alpha that we have here is called the learning rate. And it is usually something like 0.1. So we update our estimate of Q to be most like our old value of Q, but adding in a new term that depends on r and the new state, which is a s prime. Now this small q(r, s) is an example of the value of taking action a in state s. The actual reward r is a sample of the expected reward function.

And the actual next state s prime is a sample from the next state distribution and the value that state s is the value of the best action we can take in it which is the maximum over a prime of all Q's, which are in s prime and a prime. Now such a learning algorithm is guaranteed to converge to optimal Q, if the world is really a Markov decision process.

**(Refer Slide Time: 29:49)**



**Q-Learning**

- Requires that the states and actions be drawn from a small enough set that we can store the Q function in a table.
  - Large or even continuous state spaces, make the direct representation approach impossible.
  - We can try to use a function approximator, such as a neural network, to store the Q function, rather than a table.
- Q-Learning can sometimes be very slow to converge. More advanced techniques in reinforcement learning are aimed at addressing this problem.

The Q learning algorithm requires that the states and action be drawn from a very small set that we can store the Q function in a table. Large or even continuous state spaces make the direct representation approach impossible. We can try to use a function approximator such as a neural network to store the Q function rather than a table. Q-learning can sometimes be very slow to converge. And therefore, there are more advanced techniques in reinforcement learning aimed at addressing this problem. However, as just a basic introduction to reinforcement learning, we do not delve into those.

**(Refer Slide Time: 30:43)**

# Reinforcement Learning



- Reinforcement learning is appealing because of its generality.
  - Any problem domain that can be cast as a Markov decision process can potentially benefit from this technique
- Reinforcement learning is a promising technology; but possible refinements that will have to be made before it has truly widespread application.
  - Reinforcement learning is an extension of classical dynamic programming in that it greatly enlarges the set of problems that can practically be solved.
  - Combining dynamic programming with neural networks, many are optimistic of solving a large class of problems.

34

© Shyamanta M. Hazra, M.E., IIT Guwahati

So to conclude our lecture today, we have looked at reinforcement learning, which is appealing because of its generality. Any problem domain that can be cast as a Markov decision process can potentially benefit from this technique. Reinforcement learning is a promising technology, but possible refinements that will have to be made before it has truly widespread application.

Two points to take note of before we sign off is that reinforcement learning is an extension of classical dynamic programming, in that it greatly enlarges the set of problems that can practically be solved. Combining dynamic programming with neural networks, many are optimistic of solving a large class of problems. Thank you very much.