**Lecture 1: Introduction to Reinforcement Learning**

* Science of decision making
* What makes reinforcement learning different from other types machine learning? – There is no supervisor, only a reward signal (trial and error)
* Feedback is delayed, not instantaneous – Only in retrospect do you realize that the decisions made are good or bad
* Time matters – data keeps changing with time and actions of the agent
* Some real-world applications of RL: Humanoid walking, learning to play Atari, Backgammon, helicopter stunt manoeuvres, pong etc.

**Rewards:**

* A reward is a scalar feedback signal
* Indicates how well an agent is doing
* Goal is to maximize the total future reward

**Sequential Decision Making:**

* Goal: select actions to maximise the total reward
* Actions may have long term consequences
* Reward may be delayed
* It may be better to sacrifice immediate reward to gain more long-term reward
* Examples: Financial investment takes a long time to mature

**Agent and Environment:**

* Observation 🡪 AGENT 🡪 action (influenced by rewards)

**History and state:**

* Sequence of observations, actions, rewards is called the history
* All observable variables up to time t
* What happens next depends on the history
* State is a concise summary of the history of an agent
* St = f(Ht)🡪 state is a function of history
* The environment state: It is the environment’s private representation used within it, used to determine what will happen next in the environment’s perspective. The environment state is usually not visible to the agent, and even if it is visible it will contain irrelevant information (e.g., air velocity is not important to a robot, only its own velocity is relevant for making decisions).
* Agent state: The agent state is the agent’s internal representation. It is the information used by the agent to pick the next action, based on its history.
* Information State: An information state (a.k.a. Markov state) contains all useful information from the history.

A state is called a Markov state if and only if: P(St+1 |St ) = P(St+1 |S1 , S2, ……St) 🡪 Future is independent of the past given the present

* The environment state is Markov
* The entire history Ht is Markov

Fully Observable Environments:

* Full observability is a state where the agent directly observes the entire observation state ( Ot = Sta = Ste )
* Agent state = Environment state = Information State 🡪 Formally, this is a Markov Decision Process (MDP)

Partially Observable Environment:

* Ex: A robot with a camera isn’t told its absolute position
* A trading agent only observes the current prices
* This is called a partially observable Markov decision process (POMDP)
* Now we have to build the agent’s state (Sta). Sta = Ht 🡪 Remember everything
* Beliefs of environment state: Probability distribution which calculates the probability of being present in a given state
* Recurrent neural networks also can be used

Major components of an RL agent:

* Policy: Agent’s behaviour (map from state to action). It can be either deterministic policy: a=(s) or stochastic (especially useful for random exploration)
* Value function: determines how good is it to be in a particular state or to carry out a particular action. Essentially evaluates the goodness/badness of states and thus select between different possible actions that the agent can take
* Model: Agent’s representation of the environment (predicts what the environment will do next)

Models:

* Two kinds: Transition model (P): Predicts the next state (i.e., dynamics; Reward Model: Predicts the next reward

Categorizing RL agents:

* Value based: Based on the value function and hence no policy is needed (it’s implicit)
* Policy based: Based on the policy (without explicitly storing the value)
* Actor critic: Uses both policy and value function
* Model Free: Policy/Value is used without modelling the environment
* Model Based: Policy/Value function is combined with the agent’s model of the environment

Problems in RL:

1. RL problem: environment is initially unknown; agent interacts with the environment and it improves its policy
2. Planning: model of the environment is known; the agent performs computations with its model (without any external interaction)
3. Exploration is the process of knowing more about the environment by giving up short-term known rewards in the pursuit of a possible larger reward
4. Exploitation is using the information that you already have to maximize the reward

**Lecture 2: Markov Decision Process (MDP):**

* Markov decision processes formally describe an environment for reinforcement learning
* The environment is *fully observable*; i.e., the current state completely characterizes the process
* Almost all RL problems can be formalised as MDPs
* Optimal control deals with continuous MDPs
* Partially observable problems can be converted to MDP
* Bandits are MDPs with only one state (clicking on an Advertisement will give you a reward, and you won’t if you don’t; so there’s only one state and this is called a Bandit)
* **DEFINITION: A state is called a Markov state if and only if: P(St+1 |St ) = P(St+1 |S1 , S2, ……St) 🡪 Future is independent of the past given the present**

**State Transition Matrix:**

* For a Markov state s and successor state s’, the state transition probability is defined by:

Pss’ = **P**[St+1 = s’ | St = s] {probability of transitioning into future state s’ given that the current state is s}

**Markov Process:**

* It is a memoryless random process, i.e., a sequence of random states S1, S2,…Sn with the Markov property.
* A Markov process (aka Markov chain) is a tuple (S,P), where S is a finite set of states, and P is the set of transition probabilities in the transition probability matrix.

**Markov Reward Process (MRP):**

* A Markov Reward process is a tuple (S,P,R,ϒ), where

S 🡪 Finite set of states

P 🡪 State transition matrix

Pss’ = P[St+1 = s’ | St= s]

Rs = E[Rt+1 | St =s]

ϒ = Discount factor (0<=ϒ<=1) is the present value of future rewards

**Return (Gt )**

The return Gt is the total discounted reward from time-step t.

Gt = Σ (k=0 to Infinity) ϒk Rt+k+1

Due to the presence of the reward function ϒ which gets exponentially smaller, intuitively it means that there is lesser value in the present for rewards of the future.

**Why discount?**

1. This is used because there is more uncertainty into the future. This is to show that we don’t have a perfect model of the future
2. Mathematically convenient to discount rewards
3. Avoids infinite returns in cyclic Markov processes
4. Animals and humans tend to seek instant rewards

It is sometimes possible to use undiscounted factors (ϒ = 1)only if you that all the states terminate.

**Value Function:**

* The state value function v(s) of an MRPis the expected return starting from state s (mean of all the possible returns from a current state S)
* Myopic value function: ϒ🡪0. We only care about the immediately adjacent states and not about any other states of the process into the future.
* Far-sighted value function: ϒ 🡪 1

**Bellman Equation for MRPs:**

* v(s) = E [ R t+1 + ϒGt+1 | St =s] (E 🡪 Expectation or mean of all possible states possible from current state)

= E [ R t+1 + ϒv(St+1 )| St =s]

* v = R + ϒPv

A close up of text on a white background

Description automatically generated

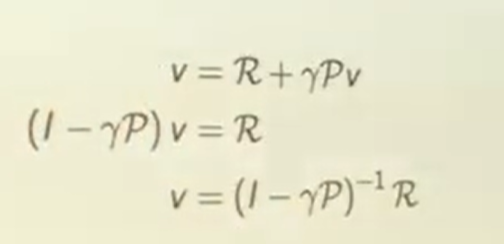
* Bellman’s equation in Matrix form:

A picture containing device, meter

Description automatically generated

Here, P is the state transition matrix described above.

* Solving Bellman’s equation:



* Computational complexity when inverting this is n3 (n= no. of states) and thus isn’t a practical way of solving MRPs with large complexities.

**Markov Decision Process:**

* A Markov decision process (MDP) is an MRP with decisions. It is an environment where all the states are Markov.
* A Markov decision process is a tuple (S, *A*, P(a), R(a), ϒ), all notations and definitions being consistent with the MRP. State probability matrix depends on what action you take. So we have an additional set of probabilities of taking various actions available at our disposal. In addition, the rewards also depend directly on our action. This dependence is denoted by the subscript (a).
* A policy π(a|s) = P[At = a | St = s] is a distribution over actions given states and it fully defines the behaviour of an agent.
* MDP policies depends only on the current state and not on the history (Markov property)
* In addition, the action depends only on the current state that we’re in and not the time-step at which the state occurs. So, the action is independent of time.

Types of Value Functions in an MDP:

1. State-value function: vπ (s) = Eπ [Gt | St=s, At =a] – In this there is an equal probability of picking all the available actions from a given state.
2. Action-value function: The expected return starting from state s, taking action a, and following policy π(a|s).

Bellman equation for the state-value function for an MDP:

A close up of a logo

Description automatically generated

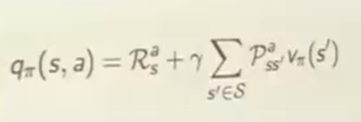
A picture containing object

Description automatically generated

Similarly, the Bellman equation for action-value function can be decomposed as:

A screenshot of a cell phone

Description automatically generated



Since any episode will have a combination of both state-value functions and action-value functions, we can put them together to achieve the following value function:

A close up of text on a white background

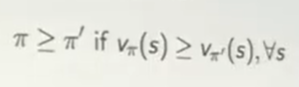
Description automatically generated

**Optimal Value Function:**

* It states the best possible performance in an MDP
* An MDP is “solved” when we know its optimal value function.
* The optimal state value function v\*(s) is the maximum value function over all policies.
* The optimal action value function a\*(s) is the maximum action value over all policies.

**Optimal Policy:**

* A policy is said to be optimal if it is better than all other policies in that it’s value function is better than all other policies in every possible state ‘s’.



* There can be more than one optimal policy and each optimal policy will need to achieve the same reward.

**Bellman Optimality Equation for v\***

* The optimal value functions are recursively related by the Bellman Optimality equations:

A picture containing object

Description automatically generated

**Bellman Optimality Equation for Q\***

A close up of text on a white background

Description automatically generated

**Bellman Optimality Equation for V\***

A close up of text on a white background

Description automatically generated

**Solving Bellman Optimality Equations:**

* Bellman Optimality Equation is non-linear
* No closed form solution (in general)
* Many iterative solution methods (Value iteration, Policy iteration, Q learning, Sarsa)