Lecture Notes

Classical Reinforcement Learning

Reinforcement learning is learning to take an action in a situation in order to maximize numerical reward signal. The agent (the learner) is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Reinforcement learning is similar to 'human learning'.

Agent-Environment Interaction

You learnt that agent observes the environment, then takes an action which he thinks is good. After taking the action, the environment tells him how good that action was, in the form of numerical **rewards**; and presents a new **observation** to the agent. The agent does so in order to complete the **task.**

There are two types of tasks:

- Continuous - tasks that do not have a definite end - e.g. learning to walk, controlling a chemical plant, driving a car
- Episodic tasks - tasks that have a definite end (these tasks have a terminal state)- e.g. most games (videos games, Chess, Ludo) etc. are episodic since at the end of the game the agent either wins or loses.

Markov State

You learnt that a **state** is a representation of the environment at any point in time. The environment will give all the signals, but how relevant those signals are for the agent to take an action, is what you've to decide.
A **state vector** is a list of features that help the agent to take an action. For each RL problem, state vector would be different.

You learnt that **Markov state** is some function of the knowledge base: that captures all relevant information from the knowledge base. The Markov assumption states that the current state contains all the necessary information about the past (states, actions and rewards) to take the next action. All these processes that work in accordance to Markov property are called **Markov Decision Processes** (popularly called MDPs).

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

Elements of Reinforcement Learning Problem

Policy is a set of rules which helps the agent decide the action that it should take in a given state such that the agent can maximise its rewards in the long run. There are two types of policies:

- deterministic policy: $\pi(s) \rightarrow a$
- probabilistic policy: $\pi(a|s)$

Exploration-Exploitation Trade-off: Exploiting an action is fine if you have exhaustively explored all the actions from a given state. But this is generally not the case in real-life situations. In most scenarios, you would have explored only a small fraction of all possible actions. Therefore, you keep on exploring, so that you don't lose out on the benefits of the unexplored actions.

Value Function of state: the total amount of reward an agent can expect to accumulate over the future, starting from that state. State-value function tells how good it is for the agent to be in a particular state. The agent would want to be in a state such that its **total rewards** (immediate + expected future rewards) are maximized. It is represented as v(s)

Action Value Function: the total reward an agent can expect if it performs an action 'a' from state 's'. It is represented by q(s,a)

You learn that **objective of RL Agent** is to find an *optimal policy* **(**that can help the agent to accumulate maximum rewards in any task). For both episodic and continuous tasks, the objective is to find optimal policy.

Optimal Policy

A policy π^* is called optimal only if $\forall\, \pi$: $\pi^* \geq \pi$

Policy π is better than π' if, for all states, the value function is higher if you follow policy π than if you follow policy π'. Mathematically,

$$\pi \geq \pi^* \; \forall s: v_{\pi}(s) \geq v_{\pi'}(s)$$

The **objective of RL agent** is to find an optimal policy that can help the agent to accumulate maximum rewards in any episode.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

RL vs Supervised Learning

Next, you learnt the difference between Supervised Learning and Reinforcement Learning.

- In Reinforcement Learning, you deal with the processes where the agent actively interacts with the environment, whereas in supervised learning, there is no interaction with the environment and given a dataset, you are required to predict the target.
- RL is an active learning, where the agent learns only by interacting. While supervised learning is passive learning, where the agent learns only by extracting features from a given dataset.
- In supervised learning, there is a teacher (ground-truth) which tells you whether the result for a given observation is correct or not. And, then the model can be improved by minimising the error term. On the other hand, in reinforcement learning, there is no teacher. The environment acts only as a critic, where it tells you how good or bad the action is by giving rewards. It doesn't tell whether the action taken is the ultimate best or not.

Bellman Expectation Equations

You learnt to define state and action-value function for a given policy as follows:

- $v_\pi(s) = \sum_a \pi(a|s) q_\pi(s,a)$

This is the first basic equation of RL. The intuition of this equation is that the value function of a state 's' is the weighted sum over all action-value functions for different actions that can be taken from state 's'.

- $q_\pi(s,a) = \sum_{s'} \sum_r p(s',r|s,a)(r + \gamma v_\pi(s'))$

This is the second fundamental equation in RL. The intuition of this equation is that the action-value function for an action 'a' taken from state 's' is the weighted sum of total reward (immediate reward + discounted future reward) over model probabilities.

Where **γ** is a discount factor.

This discount factor determines the importance of future rewards, i.e. how much you value the future rewards.

- A factor of 0 would mean that you don't care about future rewards at all and only the immediate reward matters.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

- A factor of 1 would mean that you value the immediate and future rewards equally.

Another important feature of having a discount factor is for continuous tasks. When total rewards are calculated for continuous tasks, the total rewards may not converge. So, the discount factor will keep the total rewards bounded.

Bellman Optimality Equations

Then, you learnt about state and action value functions for optimal policy. Value of a state under an optimal policy must be equal to the expected total return for the best action from that state. At each state, **greedily pick the best action** that has the maximum q-function.

State-value and action-value functions for the optimal policy can be defined as:

- $v^*(s) = \sum_a \pi^*(a|s)q^*(s,a)$
- $q^*(s,a) = \sum_{s'} \sum_r p(s',r|s,a)[r + \gamma v^*(s)]$

These equations relate the optimal policy to the optimal state and optimal action-value equations. These equations are popularly known as **the Bellman Equation of Optimality.**

Model of the Environment

The model is represented as: p(s’,r|s,a). It is the probabilistic distribution of finding the agent in state s and reaping the reward r, given a particular action ‘a’ is taken in a particular state ‘s’. This is known as **the model of the environment.**

In most real-world scenarios, you wouldn’t know what exactly the model of the environment is. You implicitly infer about the model from the observations. Such problems are called **model-free**. Whereas the problems where you know the model explicitly beforehand are called **model-based.**

So, the objective of an RL agent is to find the optimal policy either using the explicit model (model-based) or by implicitly inferring the model from the actions taken from various states (model-free).

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

Model-Based Methods Dynamic Programming

You'll learn that there are two basic steps for solving any RL problem.

- **Policy evaluation** refers to the iterative computation of the value functions for a given policy.
- **Policy improvement** refers to the finding an improved policy given the value function for an existing policy.

Then, you learnt that why dynamic programming can be used to solve Markov Decision Process

- Bellman equations are recursive in nature. You can break down Bellman equation to two parts – (i) optimal behaviour at next step (ii) optimal behaviour after one step
- You can cache the state value and action value functions for further decisions or calculations.

The basic assumption for model-based method is that: model of the environment p(s', r | s, a) is available.

There are two ways to arrive at the optimal policy and the optimal value functions:

- Policy Iteration
- Value Iteration

Policy Iteration

The pseudo-code for Policy Iteration is as below:

1. Initialize a policy and state-value functions (for all states) randomly
2. Policy Evaluation (also known as the prediction problem):

   Measure how good that policy is by calculating the state-values corresponding to all the states until the state-values are converged

$$v_\pi(s) = \sum_a \pi(a|s)[\sum_{s'} \sum_r p(s', r|s, a)[r + \gamma v_\pi(s')]]$$

3. Policy Improvement (also known as the control problem):
   You learnt to do this by following a greedy approach, i.e, by taking the best action at every state**.** The action with the **maximum state-action value q_π(s,a)** is the best action. The improved policy π' is:

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

$$\pi'(a|s) = \begin{cases} 1 & a = arg\max_a[\sum_{s'}\sum_r p(s',r|s,a)[r+\gamma v_\pi(s')]] \\ 0 & otherwise \end{cases}$$

Value Iteration

In policy evaluation, you perform iterative calculations through all states until convergence. This becomes a problem if you have a large state-space. Performing state-value calculations for a huge state-space until these values stabilise is **computationally expensive**.

Instead of policy iteration, **Value Iteration** doesn't require an entire iterative calculation at the policy evaluation step and yet guarantees the convergence to optimality.

The pseudo-code of Value Iteration is as below:

1. Initialize state-value functions (for all states) randomly
2. Policy Evaluation & Improvement:
   Exactly one update is performed for calculating the state-values. These state-values are corresponding to the action that yields the **maximum state-action value**.

   $$v(s) = \max_a[\sum_{s'}\sum_r p(s',r|s,a)[r+\gamma v(s')]]$$

The policy improvement is an implicit step in state-value function update. The update step combines the **policy improvement** and (truncated) **policy evaluation steps.**

Generalised Policy Iteration

Policy Iteration and Value Iteration are two sides of a coin. There are some techniques that cover the spectrum between policy iteration and value iteration. These are called Generalised Policy Iteration (GPI) methods.

Next, you learnt to apply both Policy Iteration and Value Iteration on Ad Placement Problem.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

Model-Free Methods – Monte Carlo Methods

In most real-world scenarios, the model is not available. You have to implicitly infer the model from the observations. Methods designed to solve these scenarios are called model-free methods.

Monte-Carlo method is based on the concept of **the law of large numbers**. The Q-function can be written as **the expected value of total rewards over model distribution:**

$$Q_\pi(s,a) = \sum_{s',r} m_{s',r}[total\ reward_{s'}] = E_{model}[total\ reward] = E_{model}[r + \gamma v_\pi(s')]$$

You run the episodes **infinite number of times** and calculate the rewards $[r + \gamma v_\pi(s')]$ earned every time a (state, action) pair is visited. Then take the average of these values. The average value will give an estimate of the actual expected state-action value for that (state, action) pair.

The Monte-Carlo methods require only knowledge base (history/past experiences)—sample sequences of (states, actions and rewards) from the interaction with the environment, and no actual model of the environment. Model-free methods also use **prediction** and **control** steps to solve any Reinforcement learning problem.

Monte-Carlo Prediction:

Monte-Carlo prediction problem is to **estimate** q_π(s,a), i.e. the **expected total reward** the agent can get after taking an action 'a' from state 's'.

- For this, you need to run multiple episodes
- Track the total reward that you get in every episode corresponding to this (s,a) pair
- The **estimated** action-value is then given by $\mathbf{q_\pi(s,a)} \approx \sum \frac{\boldsymbol{r_i}}{\boldsymbol{n}}$

The most important thing to ensure is that, while following the policy π, each state-action pair should be visited enough number of times to get a true estimate of q_π(s,a). For this, you need to keep a condition known as the **exploring starts**. It states that every state-action pair should have a non-zero probability of being the starting pair.

You also learnt about first-visit and multiple visit update variations. Most of times we prefer first-visit update.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

Monte-Carlo Control:

Policy improvement is done by constructing an improved policy π′ as the ϵ-greedy maximisation with respect to $q_{\pi}(s,a)$. So, π′(a|s) is:

$$\pi'(a|s) = \left\{ \begin{array}{cc} 1-\epsilon+\frac{\epsilon}{|A(s)|} & a = arg\max_{a'} Q_{\pi}(s,a') \\ \frac{\epsilon}{|A(s)|} & otherwise \end{array} \right\}$$

Where |A(s)| is the action space, i.e. total possible actions in any given state and ϵ is a hyperparameter that controls the tradeoff between **exploration and exploitation**.

- If ϵ is too small, then actions are biased to be more greedy
- If ϵ is too large, then actions explore more.

Off-Policy

A way to handle the dilemma of exploring and exploitation is by using two policies:

- **Target Policy:** the policy that is learnt by the agent and that becomes the optimal policy
- **Behaviour Policy:** the policy that is used to generate episodes and is more exploratory in nature

This is called **off-policy** because the agent is learning from data 'off' the target policy.

In another case, if the (optimal) policy learnt and the policy that is used to generate episodes is the same, it is called **on-policy** learning.

Importance sampling: is a technique that is used to ensure that the state-actions pairs produced by the target policy are also explored by behaviour policy.

The sample episodes are produced by behaviour policy but you want to learn a target policy.

Thus, it is important to measure how important/similar the samples generated are to samples that the target policy may have made. Importance sampling is one way of sampling from a **weighted distribution** which favours these “important” samples.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

Model-Free Methods – Temporal Difference Learning

You learnt that in Monte Carlo methods, you need to wait until the end of the episode before updating the q-value. Therefore, you can use Temporal Difference (TD) methods where you can update the value after every time step. And this update can be very advantageous when some of the states are extremely disastrous.

The most popular algorithm of TD method is **Q-Learning**.

In Q-learning, you do **max-update**. After taking action ‘a’ from state ‘s’, you get an immediate reward of r and land to state s’. From s’, you take the most greedy action, i.e., action with highest q-value.

So, q-value function update becomes:

$$q(s,a) := q(s,a) + \alpha((r + \gamma \max_a q(s',a)) - q(s,a))$$

Where **α is the learning rate** that decides how much importance you give to your current value.

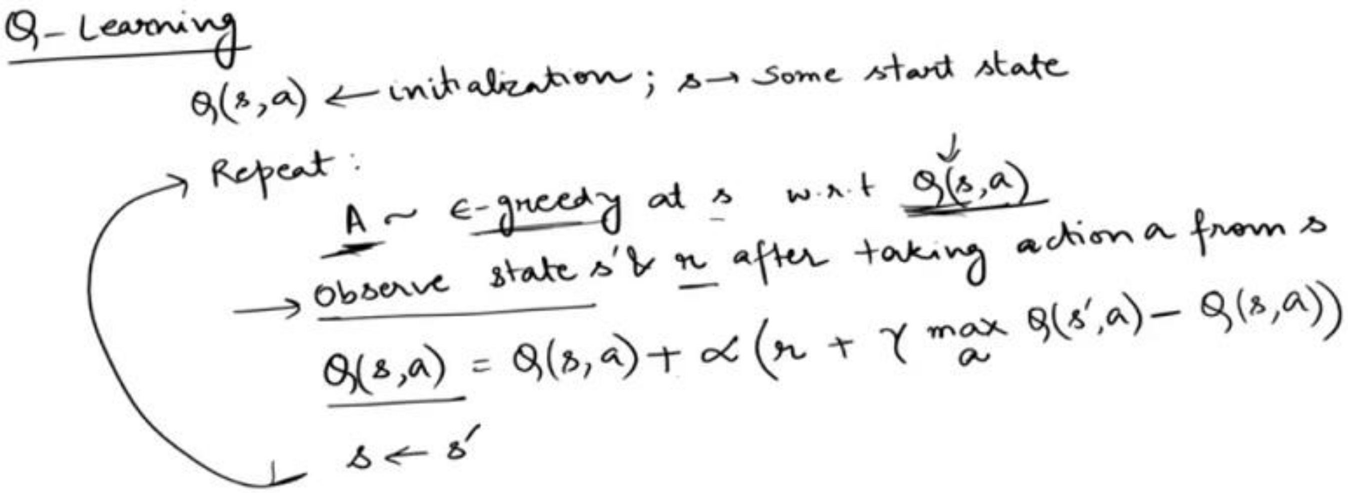
- If α is too low, that means, you value the incremental learning less
- And If α is too high, that means, you value the incremental learning much more than your current estimates.

Q-learning directly learns the optimal policy, because the estimate of q-value is updated on the basis of 'the estimate from the **maximum estimate of possible next actions**', regardless of which action you took.

If there is a risk of a large negative reward close to the optimal path, Q-learning will tend to trigger that reward while exploring. And in practice, if the mistakes are costly, you don't want Q-learning to explore more of these negative rewards. You will want something more conservative. SARSA and Double Q-learning are more conservative approaches that avoid high risk.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

The pseudo-code of Q-learning is as below:



```
Q-Learning
    $Q(s,a) \leftarrow$ initialization ; $s \rightarrow$ Some start state
    Repeat :
        $A \sim \epsilon$-greedy at $s$ w.r.t $Q(s,a)$
        Observe state $s'$ & $r$ after taking action $a$ from $s$
        $Q(s,a) = Q(s,a) + \alpha\left(r + \gamma \max_{a} Q(s',a) - Q(s,a)\right)$
        $s \leftarrow s'$
```

OpenAI Gym Environments

Founded by Elon Musk and Sam Altman, **OpenAI** is a non-profit research company that is focussed on building out AI algorithms. **OpenAI Gym** is a toolkit for developing and comparing reinforcement learning algorithms. Gym provides different environments to implement any reinforcement learning algorithms. You learnt to use these environments and apply RL algorithms on it

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved

Disclaimer: All content and material on the UpGrad website is copyrighted material, either belonging to UpGrad or its bonafide contributors and is purely for the dissemination of education. You are permitted to access print and download extracts from this site purely for your own education only and on the following basis:

- You can download this document from the website for self-use only.
- Any copies of this document, in part or full, saved to disc or to any other storage medium may only be used for subsequent, self-viewing purposes or to print an individual extract or copy for non-commercial personal use only.
- Any further dissemination, distribution, reproduction, copying of the content of the document herein or the uploading thereof on other websites or use of content for any other commercial/unauthorized purposes in any way which could infringe the intellectual property rights of UpGrad or its contributors, is strictly prohibited.
- No graphics, images or photographs from any accompanying text in this document will be used separately for unauthorised purposes.
- No material in this document will be modified, adapted or altered in any way.
- No part of this document or UpGrad content may be reproduced or stored in any other web site or included in any public or private electronic retrieval system or service without UpGrad's prior written permission.
- Any rights not expressly granted in these terms are reserved.

© Copyright 2018. UpGrad Education Pvt. Ltd. All rights reserved