Reinforcement Learning Primer

By Ian Ho

What is Reinforcement Learning?

The area of machine learning is divided into 3 sub-areas: supervised learning, unsupervised learning, and reinforcement learning. In supervised learning, the AI learns to assign numerical or categorical labels from labeled examples. In unsupervised learning, the AI discovers hidden structure in unlabeled examples. In reinforcement learning, the AI learns what actions to take when interacting with an environment.

Reinforcement Learning is learning what actions to take in certain situations in order to maximize a numerical reward signal. The agent is not told what actions to take but instead must try actions to discover which actions yield the most reward. The agent does not try to maximize the immediate reward, it tries to maximize the total future reward.

A Very Simple Example of Reinforcement Learning
Imagine you are a dog and your owner is trying to teach you a new trick. They are holding up a treat right in front of you and they keep repeating some word to you. Since you are a dog, you have no idea what they are saying but you try doing random actions hoping that the owner will give you the treat.

First, you try sitting down - the owner does not give you the treat. Then, you roll over - still no treat. Next, you high five the owner - the owner gives you the treat! Since you like treats, everytime the owner says that word from now on you high five them because you want to maximize the amount of treats you get.

Terminology

Before we continue let’s talk about some terminology. Almost all of the definitions relate to other terms so you may have to jump back and forth between the definitions.

Agent: The thing that interacts with its environment and is trying to learn to maximize its reward based on the actions it makes.

Environment: What the agent is interacting with. The environment determines what rewards the agent gets and the new state the agent arrives at. The environment can be seen as a mapping from a state and an action to a new state and a reward.

State: How the environment is at a particular time. The state encodes all the information available to the agent.

Action: A decision that the agent can choose to make. The action is how the agent interacts with the environment. At each state, there is a set of available actions that the agent can make.

Reward Signal: Defines the goal of the reinforcement learning problem. After each action is taken, the environment returns a number to the agent. The agent's only goal is to maximize the total reward it receives in the long run. The reward signal may be stochastic and return rewards at set probabilities.

Value Function: The total amount of reward that the agent can expect to accumulate in the future when being in a particular state. This function is a mapping from state to a real number. Another type of value function is the State-Action Value Function which is the total amount of reward that the agent can expect to accumulate in the future when taking a particular action when in a particular state. The state-action value function is a mapping from a state-action pair to a real number. The state-action value function is also known as the Q function.

Policy: Defines the agent's decision making at a given time. The policy can be seen as a mapping of the state of the environment to actions to be taken in those states. Policies can be stochastic, meaning that in a state there is a probability that each action should be taken. The optimal policy is the policy that results in the maximum total reward.

Model: Mimics the behaviour of the environment and predicts how the environment will behave. Models are used for planning which does not do any trial-and-error learning. RL algorithms are often divided into *model-based* methods and *model-free* methods.

Applying the Terminology to an AI Playing Chess

Let's say you wanted to train an AI to play the game of Chess (if you don't know the rules of Chess, do some quick research before continuing). The AI itself would be the agent. The board, rules of the game and opponent would together make up the environment.

Given the locations of the Chess pieces during the AI's turn (the state), the AI would decide which of its pieces to move and where (the action). Then it would be the opponent's turn, who would move one of their pieces. After the opponent is done their turn, the new locations of the chess pieces would represent a new state. Note: We do not care about the locations of the pieces during the opponent's turn because the AI does not have any decisions to make when it's the opponent's turn.

The reward system is designed by whoever is designing the algorithm. A good reward system would be as follows: If the opponent is in check-mate in the new state, the AI would receive a +1 reward. Otherwise, the AI would receive a +0 reward. Therefore, if the AI wins the game, the total reward will be 1 but if it loses, the total reward will be 0. By learning to maximize the total reward of this reward system, the agent would be learning to win the game. If instead, rewards were assigned for removing the opponent's pieces from the board, then the AI would likely still perform decently well. However, the goal of Chess is to put your opponent in check-mate, not take all their pieces off the board. Thus, the first reward system mentioned would have better results.

The value function is the expected future reward after being in a particular state. So since our total reward will be 1 if the AI wins and 0 if it loses, then the value function will represent the probability of winning when in a particular state. So if the AI had played 4 games already and only won the last one, then the value function of the starting positions in Chess would be 0.25 ((0 + 0 + 0 + 1) / 4).

Each time the AI needs to make a decision, it will consult it's policy. The policy would be a function that takes in the positions of all the pieces (the state) as input and would output which piece to move and where. So if the AI had just started a game, in order to decide what piece to move, it would look at the starting positions of all the pieces. The policy would take this information as input and then output an action. An example of an action in Chess might be "1.e4" which is a Chess move where one of the pawns in the middle moves up 2 spaces.

As the AI plays more and more games, it will update the value functions and then update its policy. How the algorithm adjusts its policy depends on the algorithm.

If someone were to predict what decisions the opponent would make, they could use those predictions together with the rules of the game to create a model of the environment. This simulated environment to play against would be the model.

Exploration vs Exploitation Trade-Off

Exploration: Trying new actions that you haven't tried before to explore what rewards you would receive.
Exploitation: Choosing actions that you have tried before because they returned high rewards in the past.

The exploration vs exploitation trade-off is a unique problem to reinforcement learning. If you *exploit* too much, then you are likely to receive decently high rewards, but you may be missing out on receiving the maximum reward possible. If you *explore* too much, then you may discover what actions would lead to the maximum rewards, but you aren't taking advantage of those best actions and may be receiving low rewards.

The problem arises if you have to maximize the reward through a limited number of actions, or if you want to receive large rewards as quickly as possible, or if you want your algorithm to converge to the optimal policy as quickly as possible.

Slot Machine Example

Consider a casino with 2 slot machines that each accept 1 token every time you use it. On the first slot machine you will win $0, 50% of the time and $10, 50% of the time. On the second slot machine you will win $1, 100% of the time. Alex, Ben and Carly each have 10 tokens and decide to go to the casino that day to try the slot machines. However, none of them are told this information about what distribution each slot machine follows but they have all heard rumours that one slot machine is better than the other. Alex, Ben and Carly do not know each other and do not share information about what they learn about the slot machines. This problem is also known as the multi-armed bandit problem.

Alex goes first. He starts by trying the first slot machine and wins 0$. He then tries the second slot machine and wins $1. He figures that since he won more on the second machine, he will spend his 8 remaining tokens on the second machine. He ends the day with $9.

Ben goes next. He starts by trying the first slot machine and wins $0. He then tries the second slot machine and wins $1. He goes back to the first slot machine and wins $10 then back to the second machine and wins $1. He's very curious and keeps alternating between the two because he wants to find out which one is better. He ends up with $25.

Carly goes third. She starts by trying the first slot machine and wins $0. She then tries the second slot machine and wins $1. She goes back to the first slot machine and wins $10 then back to the second machine and wins $1. She decides she will use her last 6 tokens on the first machine. She ends the day with $42.

Round	**Alex**		**Ben**		**Carly**	
	Slot Machine	**Winnings**	**Slot Machine**	**Winnings**	**Slot Machine**	**Winnings**
1	1	$0	1	$0	1	$0
2	2	$1	2	$1	2	$1
3	2	$1	1	$10	1	$10
4	2	$1	2	$1	2	$1
5	2	$1	1	$0	1	$0
6	2	$1	2	$1	1	$10
7	2	$1	1	$10	1	$0
8	2	$1	2	$1	1	$10

9	2	$1	1	$0	1	$0
10	2	$1	2	$1	1	$10
Total		**$9**		**$25**		**$42**

Alex, Ben and Carly all had very different strategies in how they choose to *exploit* vs *explore*. Alex immediately started trying to exploit after trying both machines once and ended up choosing the machine with lower expected returns. Ben kept trying to explore and didn't take advantage of the first machine which returned higher rewards. Carly had a more balanced approach and explored for the first 4 rounds but then exploited the first machine once she realized it had higher expected returns.

Solutions to Reinforcement Learning Problems

The goal of reinforcement learning algorithms is to learn the optimal policy that will maximize the expected future reward. There are many different algorithms however in this text we will explain the most popular algorithm: Q-Learning.

Q-Learning

Q-Table
The Q-table is an m x n matrix where m is the total number of states and n is the total number of actions. The value in the ith row and jth column represents the state-action value function (Q-value) for taking the jth action in the ith state. The Q-table initially contains all zeros.

Selecting Actions
In each state, the agent decides if it wants to explore or exploit. One algorithm might decide to explore 25% of the time and exploit 75% of the time. This parameter is a design choice. If an agent is in the ith state and wants to exploit, the agent looks at the ith row and selects the action with the highest Q-value. If an agent is in the ith state and wants to explore, the agent selects a random action.

Updating the Q-Table
Once an action has been taken and the agent receives a reward and enters a new state, the Q-value needs to be updated. The update is shown below.

$$Q(s_t,a_t) \leftarrow \underbrace{Q(s_t,a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1},a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t,a_t)}_{\text{old value}} \right)$$

To make this easier to understand, set the learning rate to 1 and the discount factor to 1 for now. With these parameter values we get the equation:

$$Q(s_t, a_t) \; < - \, r_t \; + \; max \, Q(s_{t+1}, a)$$

We will explain the simplified equation in words to give a more intuitive explanation. The algorithm looks at the Q-values for the new state s_{t+1}. The maximum Q-value for any action in that new state $max \; Q(s_{t+1}, a)$, represents the maximum expected future reward after being in the new state. Thus by adding the reward r_t that was just received after taking action a_t, the value $r_t + max \; Q(s_{t+1}, a)$ represents the maximum expected future reward from taking action $a_t$in the state s_t. If the learning rate and discount factor both equal 1, then this value is the one we use to update $Q(s_t, a_t)$.

The Learning Rate

The learning rate determines how much the algorithm trusts new information. With a learning rate of 0, the agent will learn nothing. With a learning rate of 1, the agent will only consider new information. This parameter needs to be tuned by the programmer.

The Discount Factor

The discount factor determines the importance of future rewards. This value can be anywhere between 0 and 1. A value of 0 will cause the algorithm to only value immediate rewards and a value of 1 results in the algorithm striving for maximum future reward. It may seem obvious that we would want the discount factor to be 1 however there are many issues that can occur when this value is high. This parameter needs to be tuned by the programmer.

Function Approximation

If the state space is very large, then only a small percentage of the total possible states will have been visited. Thus, most of the Q values will not have been updated. If this is the case, it will take very long to train the agent.

Many of you have probably heard the term “deep reinforcement learning” before. Deep reinforcement learning is when a neural network is used to approximate a function such as the Q-value. If the state space is really large then only a small percentage of the Q-values will have been updated. By using these updated values, we can use this as training data to train a neural network to approximate the entire Q-table. In that case, if states have not been visited yet, if we approximate the Q-values to be close to the Q-values in a similar state then we can get a better approximation of the actual Q-value. Thus, using function approximation will speed up training significantly.

Self-Learning

- Introduction to Reinforcement Learning Course by David Silver x UCL x DeepMind (https://www.davidsilver.uk/teaching/)
    - A series of lectures on YouTube taught by the guy that lead the development of AlphaGo
- Reinforcement Learning textbook by Richard Sutton, Andrew Barto

    - This textbook is the most widely accepted RL textbook in the world
- OpenAI Gym
    - A fun tool to play around and get hands-on experience