Introduction to Reinforcement Learning

Charlie Wu*

May 2020

1 Introduction

There are three main "categories" of machine learning: supervised learning, unsupervised learning, and reinforcement learning. We have mostly been dealing with supervised learning and we have touched on unsupervised learning with K-means clustering. We will now get experience with *reinforcement learning*. At its core, reinforcement learning is about learning how to solve problems by creating a strategy through experience. To create this strategy, an RL system must undergo many trials and scenarios and learn from its successes and mistakes along the way, in a very similar way us humans approach solving new tasks and problems.

2 Background

Before we delve into the concepts of RL, we need to define a few key terms: an *agent*, *episode*, *state*, *action*, and *reward*. An agent is the RL actor that is responsible for navigating through an environment to achieve its goal, which is maximizing the total reward obtained at the end of the episode. A state is any situation part of the environment that an RL agent can "be in" at any given time. An action is any "decision-making" step an agent can take at any state S_i to get to the resulting state S_{i+1}. A reward is determined by the environment itself and is a defined "prize" or "consequence" that the agent receives every time it takes an action to get to a new state. An episode consists of a sequence of states, actions, and rewards that the agent goes through as it navigates from an initial starting state to a terminal state, whatever that might be.

In this lecture, we will be dealing only with discrete actions for simplicity's sake - ie. having the choice of actions being "left and right on the x axis" instead of "a positive or negative translational speed in the x axis".

We will also be dealing with agents who know all the information about any state that it is in - ie. the agent will never be obscured from any details regarding its state that may affect its decision-making (choosing an action).

*Based off of Mihir Patel/Kevin Fu's original lecture

3 Example

Let's imagine an RL agent that is being trained to play Flappy-Bird. The environment that the agent acts in consists of the entire game, including the flying bird, the walls, etc. The agent is the bird, whose goal is to flap its wings and fly to avoid as many of the walls as possible. The *action-space* is the set of actions that the agent can take at any given time (at any given state that the agent is in). In this game, the agent has a static action-space, so the possible actions that the bird can take are the same at any point in the game. These actions are either "flying up" or "not flying and subsequently falling". At each state, the agent chooses an action that causes it to advance to a subsequent state. If the agent chooses to fly up, the bird will be at a higher y position in the resulting state, along with the walls having moved towards the bird due to the progression of time. Each time the agent successfully navigates through a wall, it is rewarded with a constant positive reward, and each time it collides into a wall (resulting in a terminal state and end of the game/episode), it receives a final (very) negative "reward".

Our goal is to train the RL agent to learn to choose the actions that result in it achieving the maximum final total reward, which in practical terms means flying past as many walls as possible without colliding. During each episode, the agent will experience successes and failures, which will correlate to various rewards based on actions it takes. The agent will gain additional experience and train itself over time to gradually learn how to pick the better and better actions.

4 Markov Decision Process (MDP)

One key concept that has to be explained before exploring RL in depth is the Markov Decision Process characteristic. For any system to be a MDP, each new state must be dependent only on the previous state, action taken from that previous state, and reward for transitioning to the current state. MDPs are commonly described as being "memoryless", such that being in any new state S_{i+1} is only influenced by S_i and the corresponding action taken and reward received.

When visualizing MDPs, we can imagine any future decisions being independent of previous decisions. With this, how we arrive at a certain state will not influence the action taken at that state. An agent could move through a sequence of 1000 states or 10 states to reach S_i, but under the Markov property, the action taken by that agent will be the same in either scenario, as the "ideal" action can be determined only by the information given by the current state. When working with MDPs in RL, this assumption must be rendered as true, providing a simplification in our decision-making process. When our RL system decides on the best action to take at a specific state, it will only consider the information given by the current state (which is all the information it needs to make its ideal decision), as it is a MDP.

5 RL Agent Policy

We can define a *policy* π for an RL agent as being a strategy function that determines the probabilities of actions given a state. An ideal policy will yield the best action given any state. This action is picked out of the state's action space, and the action space may be variable depending on the current state - ie. a Flappy-Bird can't continue flying up if it is already at its maximum height, and a maze navigator can't move in any direction that is a dead-end or restricted by walls/corners. The policy of an agent can be viewed as part of the decision-making process that picks the ideal action for the agent and is ultimately what influences how well an agent performs in an environment.

When not training an agent, or when implementing a near-ideal RL agent, we will use a *greedy policy*, which will always select the action that it believes to be the best action; this process can be referred to as *exploitation*. However, when training an RL agent from scratch, our agent will not have the experience to distinguish ideal actions, so we must force the agent to sometimes take actions it believes not to be ideal (as these seemingly non-ideal actions may prove to actually be better than what the agent thinks is best). This process is called *exploration*, and we can induce these "non-ideal" actions onto the agent by picking a random action or adding noise to the action the agent picks. A good balance between exploration/exploitation can be enforced simply using an $\epsilon - greedy$ policy, where the agent agent exploits its knowledge to pick what it thinks is best for $\epsilon\%$ of the time, but is forced to explore new actions $1 - \epsilon\%$ of the time.

6 Value Functions

How do we find the best action to take in a given state? Since we are working with a discrete action space, we want to choose the action that will yield the *maximum possible expected future reward*. Breaking this definition down: we want to judge the value of a state not only based on how much immediate reward it can yield, but based on the total sum of all the rewards the agent will receive if it follows the consistent policy π until the terminal state. In other words, we care about evaluating states not only on the benefits they will yield the agent in the present, but also all the additional benefits the agent will receive in the future after following the same trajectory (series of actions). For example, if you had to choose between a $1000 bonus and a $500 monthly raise, the raise would be the more rewarding choice in the long run, even though the large bonus yields more return in the present.

You may encounter the terms of *state-action* and *state-value* functions, but fundamentally they are similar, as state-value functions quantify how "good" each state is, and state-action functions go one step further quantify how "good" an action taken at a given state is. Either way, assuming the Markov property, we can confidently say that the only factor influencing our value rating of the state is the state itself.

7 Bellman Equation

The Bellman Equation helps us mathematically define how we evaluate the maximum possible expected future reward. One inherent property of the future is that it is not as predictable as the present. It is relatively easy to predict the action that an agent (or anyone/anything, including us humans) may take in a short while from the present, but it is harder to say for certain what states an agent will be in/what actions an agent will take a long time from now. When calculating a future reward, we can introduce a *discount factor* which helps to still take predicted future rewards into consideration while mitigating some of the unpredictability risk. The discount factor is a value between 0 and 1, and we can define the Bellman Equation recursively by $V(S_i, a_i) = R(S_i, a_i) + \gamma * V(S_{i+1}, a_{i+1})$.

We can figure out the present reward of taking an action at a state simply by seeing the reward an agent receives after taking that action. Since we want $V(S_i, a_i)$ to accurately define the future reward, we can view it as the sum of the present reward and the value function at the next state, given the action the agent takes. This one-step-forward value function is discounted by the discount rate γ, and all future rewards will be continuously discounted by $\gamma^{\cdots}$. As the agent gains more experience through training, this value function will improve to allow more accurate estimations of the future.

8 RL Agent Rewards

Rewarding an agent as it navigates through an environment can be a simple or complex task. In our Flappy-Bird example, an agent is only rewarded with positive rewards after it crosses past walls. Since passing these walls is the main final goal that the agent has to accomplish, a fair bit of exploration will be required for the agent to properly start to learn a policy that persuades it to keep the bird at the height of the gap in the wall to pass the wall. However, to simplify this process, a separate reward system could be created that "encourages" the agent to bring the y-position of the bird closer to the y-position of the gap in the wall. As an example, the agent could be given a large reward for passing the gaps (completing its end-goal tasks), but be given smaller rewards during every state based on the difference in y-height distance between the gap and the bird. This could result in a system that requires less time/episodes to train, as the agent is "nudged into the correct direction" while learning.

This may prove beneficial in some situations, but may also prove detrimental in others. For example, a linear distance reward may not work for an environment like a maze, as being on the closest path to the end of the maze does not necessarily equal the best path to solve the maze (dead ends, etc).

Implementing a rewards system such as this also requires us to know what the best simple strategy for the agent is (in this case, getting the bird near the gap). Sometimes, we may not know intuitively what the best way to solve a problem is and can only reward an agent on completing an end-goal, requiring

the agent to "pave its own path" to determine the best policy for itself.

9 Using Neural Networks

For our RL agent to train, we can take advantage of neural networks. For a complex environment, the agent will not be able to explore all states and memorize the ideal actions to take at each state (for large state/action spaces, this is extremely memory inefficient and requires too much time to realistically explore). Instead, neural networks can be used to approximate a value function. In a commonly used RL network, a Deep Q Network (DQN), the state-value function is estimated by a neural network. A DQN takes in the inputs of a state and is able to predict the value of that state, while being trained in an environment where the agent acts through states and receives rewards. Through this, the DQN allows an agent to gradually learn an optimal policy by being able to evaluate possible actions at given states with increasing accuracy, as its value-function predictions evolve through many episodes of training.

10 Conclusion

This lecture was a brief introduction to reinforcement learning, and we have only barely scratched the surface of this field in machine learning. Reinforcement learning is currently a topic of much interest in the machine learning community, as RL systems are constantly evolving to take on greater and more difficult tasks, and many RL networks have even bested humans. In the future, reinforcement learning will be used to train the next generation of artificially intelligent systems and will be utilized to discover the most optimal solutions to everything from board games to network infrastructure to self-driving cars.