Deep Reinforcement Learning

Shiv Kumar

02/01/2019

1 Introduction

We learn using trial and error based learning, by gaining understanding of the world through interaction. We understand the cause and effects of the environment using this interaction, i.e. we understand the response to our actions and accordingly remember which actions yielded better results and which yielded bad results. After establishing an understanding of the world, we try to accomplish specific goals.

The scientific understanding on how this learning happens, i.e. the computational approach is RL. We study simpler environments at start with well defined rules and dynamics and these simpler algorithms will obviously have certain limitations as well. But their analysis gives us a good starting point and also a motivation to study more complex algorithms.

2 Applications of DRL

1. Self Driving Cars
2. Games :
   - TD Gammon : RL agent to play backgammon. There are 10^{20} states which need to be decided on. This is pretty complex. This agent helps us gain more knowledge of the game which was constrained due to human knowledge till now and helps us devise better strategies that had never been thought of before.
   - Alphago : More states or configs in this game than total atoms in the world. The agent beat one of the best players of go 4-1.
   - Atari games : Learning to play games just from pixels.
3. Robotics : A robot learning to walk (analogous to how humans learn to walk).
4. Finance
5. Biology
6. Inventory Management

3 Basic Intuition

In RL, there is an agent which is primarily the learner or the decision maker in certain given situation. For example, a puppy which has just entered the world. It has a very complex body structure which allows it to execute different possible tasks(like sitting, walking, jumping, etc.). Now when being trained, given commands by its owner, it gets to choose between a lot of possible actions. If the action performed is in sync with the owner's commands, then the pup is rewarded with some biscuits or some other kind of reward(pat on the back, or anything else), whereas if it isn't in sync, then it isn't rewarded. The

pup has a lot of possible actions but does not really have a sense of the cause and effect of these actions initially i.e. it does not know doing which action for which command is right and hence will have to try it out.

Given a certain command, it chooses an action by random (having full understanding that it has no idea what it is doing). For example, the owner asks the pup to sit down, and choosing a random action, it decides to sit down, and then waits for a response to its action and receives a single treat which is encouraging. Now it is obvious behaviour pattern that an agent would want to yield maximum possible reward in a situation.

It is again given an instruction to maybe walk and it decides to perform some other action and waits for response but does not get any return this time, which can be said to relatively discouraging and hence tries to understand whether the action is bad in general or just a wrong mapping. As time goes on, it will be able to perform a lot more interaction with its owner and gain a better understanding of what command maps to what action and accordingly gain better ability to maximize reward i.e. more interaction, more feedback, better ability to map commands and actions and hence better chance to maximize reward.

This process of systematically proposing and testing hypothesis (*proposition made on the basis of reasoning, without any assumption of its truth*) is the basic concept behind RL.

Note: The situation is not simple as this in general, there are some other problems as well which need to be kept in mind while understanding RL, i.e.

- Exploration-Exploitation Dilemma :
  - Exploration : Exploring potential hypothesis for now to choose actions. This is important to increase knowledge.
  - Exploitation : Exploiting the already available knowledge to make the best possible guess. This is important to increase reward signal.

  The problem comes in when we need to decide how to balance these two. Whether to be satisfied by what we already know or to experiment and see if there is a better strategy to this given that it may result in a worse result.
- Delayed rewards: Certain strategies might have less pay-offs in the short term but higher in the long term and vice-versa. Making the decision on which action to take then is also a dilemma.

The main takeaway is that learning from experience is the main approach to learning here.

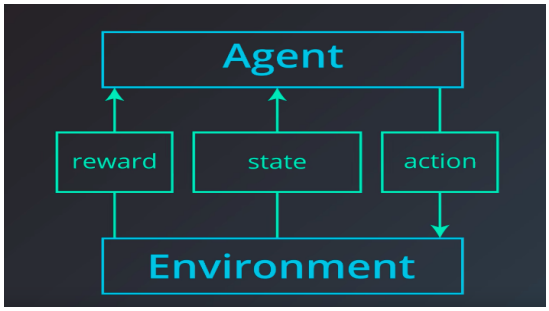
4 The RL framework

RL is basically an agent learning to interact with its environment to maximize some kind of reward signal or just reward. Here agent is the one who/which learns from trial and error(experience based learning) how to behave in an environment to maximize reward. Here we work with the assumption that time evolves in discrete time steps and at initial time-step, agent observes the environment. Also, the agent is able to fully observe what ever the state the environment is in at present.

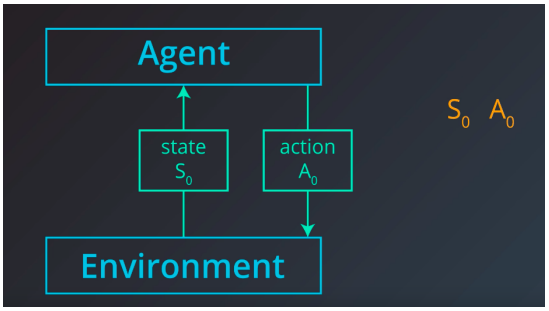
The agent receives an environment state at T_0, where it takes some action, based on which it receives an updated state and a corresponding reward as well, and this process goes on. This interaction of the agent is manifest as a sequence of states, actions and rewards.

$$\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4.., S_n, A_n, Rn + 1\}$$

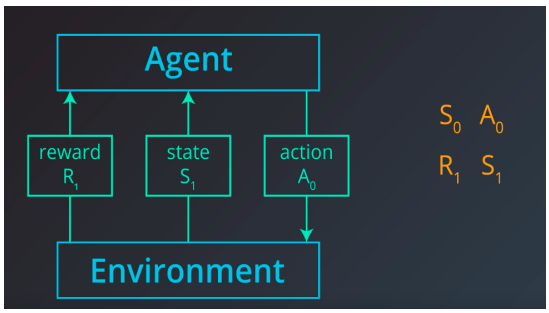
Now, the goal of any agent is to maximize the cumulative expected reward. Hence it should accordingly choose a strategy which results in such a maximized reward. This is done by interacting with the environment, i.e. the agent learns to play by the rules of the environment to complete a specific task or accomplish some goal.



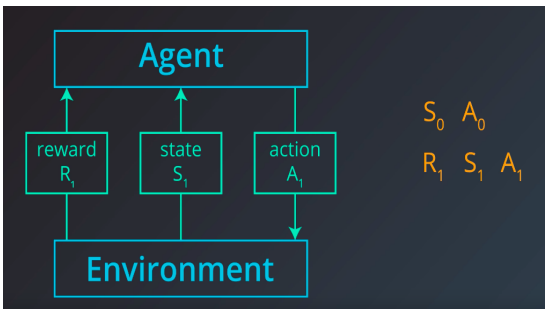
(a) The basic interaction



(b) Interaction at T_0



(c) Interaction post action at T_0



(d) Action for next time step

Figure 1: The basic RL framework

So we can define a mathematical model for a real world problem using the following:

- States
- Actions
- Rewards
- Rules of the environment

In RL there are majorly two types of tasks*(used to classify RL problems)*(instance of a RL problem, like if playing a game is a problem, a turn playing the game can be considered a task). They are episodic tasks and continuous tasks.

- Episodic tasks : Tasks(of problems) that have a defined terminal state or ending temp step such that games, car reaching its destination of crashing, walking robot agent falling. Here, when the episode ends, the agent looks at the total reward received to analyze performance. It then starts again with the gained knowledge, and over many episodes, agent starts making better decisions for itself, in order to choose a strategy to increase cumulative reward, which can be the game playing score in a game. The sequence of states, actions and rewards can be seen as follows :

$$\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4.., R_T, S_T\}$$

- Continuous tasks: These problems do not have an ending time step. Interaction with the environment continues without a pre-specified limit. For example, an algorithm that buys and sells stocks in response to the financial market. The sequence of states, actions and rewards can be seen as follows :

$$\{S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, S_3, A_3, R_4.., R_T, S_T, ...\}$$

Considering the game of chess, where the opponent is a part of the environment, we are presented with a state of the environment(configuration of the state), and based on that, we need to select/or take action,

after which we get an updated configuration of the board, on the basis of which we are supposed to take another action and so on, till one doesn't win. This is a classic example of an episodic task. The reward here is received here only at the end of the game(delayed reward) i.e. +1, -1(sparse rewards). Since the reward is provided at the end, it would be unclear whether all moves were bad or whether the game was going well till one stage and one bad move was the mistake that ruined the whole game and same for positive reward as well.

4.1 Reward Hypothesis

In RL, all agents formalize their goals based on maximizing the expected cumulative reward, i.e. they specify a reward signal or function or formula, maximization of which results in the completion of some specified goal. This is the reward hypothesis. As in the case of the puppy, the reward was pretty easy to visualize, but how do we judge a robot which is learning to walk? It would be very subjective based on the viewer and trainer and hence we say that if the robot is able to maximize the reward function then it will achieve the goal we set for it that in this case, learning to walk with stability.

We use the the problem of teaching a robot to walk more deeply by trying to analyze the work done by Deepmind on the same. First of all, to specify or frame this problem we need states, actions and rewards.

- Actions : Decisions to be made in order for the robot to walk, as the humanoid has several joints, and we can specify actions as the amount of force to be applied to the joints in order to move.
- States : Context provided to the agent(robot) for choosing intelligent actions which include
    1. Current position and velocities of the joints
    2. Measurements of the surface i.e. information about the surface : Flat? Inclined? Large step?
    3. Contact sensor data : One use of which can be to see if the robot is standing or falling.
- Reward designed as a feedback mechanism that tells the agent, if it has chosen apt movements or not. This is done by specifying a reward function as follows:

$$r = min(v_x, v_{max}) - 0.005(v_y^2 + v_z^2) - 0.05y^2 - 0.02||u||^2 + 0.02$$