Reinforcement Learning

Introduction

It's not hard to understand the idea of reinforcement learning. Suppose we have a robot (agent), and we hope it to learn how to play a computer game. Let's say, this robot can get access to the joystick of the game console, and can see what is happening on the screen.

Now we give the robot screenshot of the game 60 times per second, and ask it to hit the button it wants to press. If the robot does well, his score will increase (positive reinforcement), otherwise, we give it penalty (negative reinforcement). Gradually, the robot will understand how to play the game with knowing which actions will bring it reward, and which actions will give it penalty.

In other words, reinforcement learning is learning what to do and how to map situations to actions. The end result is to maximize the numerical reward. The learner is not told which action to take, but instead must discover which action will yield the maximum reward.

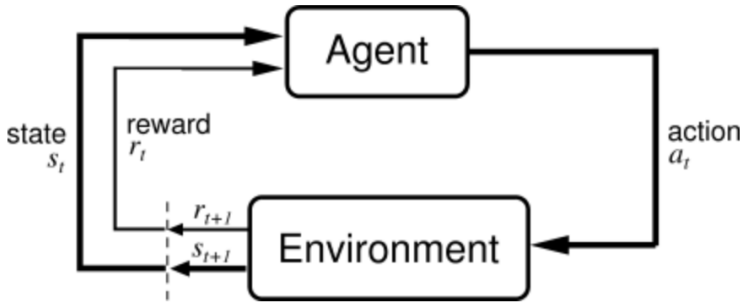

Fig. 1 a simplified description of a RL problem

Framework for RL

Before to understand how to solve a reinforcement learning problem, let's go through a classic example– Multi-Armed Bandit Problem, from which we would understand the fundamental problem of exploration vs exploitation and then go on to define the framework to solve RL problems.

Suppose you have many slot machines with random payouts. Now you want to do is get the maximum bonus from the slot machines as fast as possible. What would you do?

One naive approach might be to select only one slot machine and keep pulling the lever all day long. Sounds boring, but it may give you "some" payouts. With this approach, you might hit the jackpot (with a probability close to 0.00000….1) but most of the time you may just be sitting in front of the slot machine losing money. Formally, this can be defined as a **pure exploitation** approach.

Or we could pull a lever of each or every slot machine and pray to God that at least one of them would hit the jackpot. This is another naive approach which would keep you pulling levers all day long, but give you sub-optimal payouts. Formally this approach is a **pure exploration** approach.

Both of these approaches are not optimal, and we have to find a proper balance between them to get maximum reward. This is said to be exploration vs exploitation dilemma of reinforcement learning.

First, we formally define the framework for reinforcement learning problem and then list down the probable approaches to solve the problem.

Markov Decision Process:
The mathematical framework for defining a solution in reinforcement learning scenario is called Markov Decision Process. This can be designed as:

- Set of states, S
- Set of actions, A
- Reward function, R
- Policy, π
- Value, V

We have to take an action (A) to transit from our start state to our end state (S). In return getting rewards (R) for each action we take. Our actions can lead to a positive reward or negative reward.
The set of actions we took define our policy (π) and the rewards we get in return defines our value (V). Our task here is to maximize our rewards by choosing the correct policy. So, we have to maximize

$$E(r_t|\pi, s_t)$$

for all possible values of S for a time t.

1. Model-free vs. Model-based: feedback from the environment, imagination
2. Policy-based vs. Value-based (Combination is Actor-critic): continuous and discrete
3. Monte-Carlo update vs. Temporal-difference update
4. On-policy vs. Off-policy

Q-Learning

We will be using Deep Q-learning algorithm. Q-learning is a policy based learning algorithm with the function approximator as a neural network.

The pseudo-code of Q-Learning:

1. Initialize the value table 'Q(s, a)'
2. Observe the current state 's'
3. Choose an action 'a' for that state based on one of the action selection policies
4. Take the action, and observe the reward 'r' as well as the new state 's'
5. Update the value for the state using the observed reward and the maximum reward possible for the next state. The update can be done according to the formula and parameters described above
6. Set the state to the new state, and repeat the process until a terminal state is reached

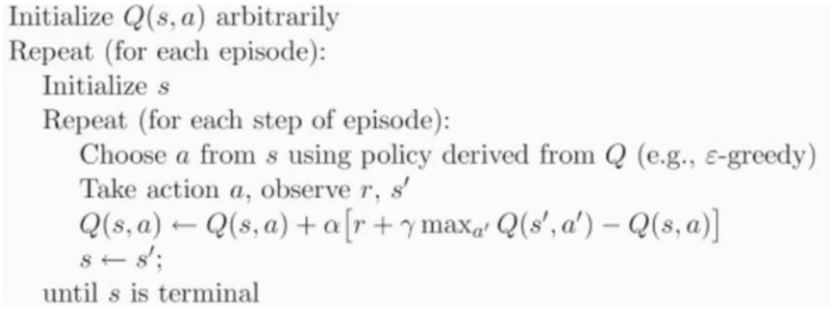

Fig. 2 pseudo code of Q-Learning

Deep Q Network is similar to Q Learning, but replace the table with a Neural Network.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D

    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Fig. 3 pseudo code of DQN

RL platforms

Deepmind Lab

Deepmind Lab is an integrated agent-environment platform for general AI research with a focus on first person perspective games.

OpenAI Gym/Universe

OpenAI Gym is a platform for creating, evaluating and benchmarking artificial agents in a game environment. While OpenAI Universe is essentially an extension to OpenAI gym, with support for literally “anything” you can do on a computer. Universe is built to emulate how a human interacts with a computer.

Peoject Malmo

Project Malmo is a research initiative by Microsoft research, with an aim to build AI agents to do complex tasks, on the top of Mincraft.

Exercise #1 Carpole problem

Implement carpole model with the help of OpenAI Gym, using Deep Q Network, and Epsilon Greedy policy.

(hints: reward is the combination of the distance between centre and the car, and the angle of the rod)