Source Seeking using Reinforcement Learning

Rahul Sajnani
International Institute of Information Technology, Hyderabad, India
rahul.sajnani@research.iiit.ac.in

Abstract

We[1] tackle the problem of *Source Seeking* in a forest setting. This report covers the basics of *Reinforcement Learning* and a few ideas to solve the source seeking problem. We also cover the building blocks leading to Deep Q learning and Deep Deterministic Policy Gradients.

1 Introduction

Source seeking refers to the problem of searching and traversing to the source of a particular signal. The signal can be a heat signature of a person lost in a forest or someone requiring medical assistance. The goal of this project is to reach the source of the signal from a drone using Reinforcement Learning. This problem is difficult as the environment space and action space are continuous.

Reinforcement learning (RL) seeks to maximize the *cumulative reward* of a bot (drone in our case) by exploring the environment. This exploration enables the bot to obtain the most appropriate action to be performed at a given state. For this, we assume that the states follow a *Markov Process (MP)* covered in section 2. The final outcome of this Reinforcement learning is to obtain a policy π^* that maps from a state to action.

Section 2 covers the mathematical basics of Reinforcement learning, problem setting and its assumptions used in majority of the RL problems. Section 3 deals with a learning paradigm to obtain the optimum policy for an RL agent. Section 4.1 delves into continuous space Reinforcement learning by diving deeper into Deep Q Networks where we understand concepts, such as fixed weight target network and experience replay. Section 4.2 tries to understand Deep Deterministic Policy Gradients, that utilizes action-critic networks. Finally, Section 5 gives the timeline of the work I have done in the first month of the Independent study and a few ideas to incorporate my learning into drone-based source seeking.

2 Reinforcement learning basics and assumptions

Let S_t be the state of the agent at time t which comprises of all the information the agent uses to determine the next action. R_s^a represents the reward received by the agent when the agent takes an action a from state s. Policy π determines the action a the agent takes at state s.

Reinforcement learning states assume a *Markov Process*. A Markov Process stores the information of the past (or future) in a single state.

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t] \tag{1}$$

[1] I have referred to myself as we throughout this report.

Here, the above equality is possible only if S_t has all the information from S_1,....S_t.

2.1 Value function

Value function $V_\pi(S_t)$ stores the expected future cumulative reward that we can obtain from that state on wards by following policy π. Where $V_\pi(S_t)$ is as follows:

$$V_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{2}$$

G_t determines the future cumulative returns from time t. Where G_t is defined as follows:

$$\begin{aligned} G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3}... \\ \gamma \in (0, 1) \end{aligned} \tag{3}$$

In the above equation, R_{t+1} is the immediate reward gained by the agent by transitioning from state S_t to S_{t+1}. γ is the decay factor which provides numerical stability and avoids maximizing rewards by getting stuck inside a closed loop.

2.2 Action Value function

The Action value function $q_(s, a)$ determines the expected cumulative future return starting from state s and taking an action a and following the policy π.

It is defined as follows:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{4}$$

Having the appropriate action value function for the optimum policy π^* provides the agent with the optimum action to take at each state to proceed towards the goal.

2.3 Model

The model of the RL agent predicts the behavior of the environment. Where $\mathcal{P}^a_{ss'}$ predicts the next state and $\mathcal{R}^a_s$ predicts the next immediate reward.

$$\mathcal{P}^a_{ss'} = \mathbb{E}[S_{t+1} = s'|S_t = s, A_t = a] \tag{5}$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \tag{6}$$

3 Q-Learning

Q-learning is a model-free algorithm to determine the action value function $q(s, a)$. Since this is a model-free algorithm we do not have any information about what is the next immediate reward $\mathcal{R}^a_s$ or the state $\mathcal{P}^a_{ss'}$ without exploring the environment. Since $\mathcal{P}^a_{ss'}$ is not known, knowing the state value function does not give us the appropriate action to take at each state. Hence, we try to obtain the action value function instead to get the appropriate action.

Our objective is to find the optimum action-value function to fill the Q-table. A Q-table stores the value of the action-value $q(s, a)$ function for a discrete environment.

We wish to obtain the Q-value for each state and each action such that the following equation should be the same ∀t:

$$q(s_t, a_t) = R_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}) \tag{7}$$

Here, note that we take the maximum q value over the action space of the next state. Here, we take the maximum over all actions as the future optimum cumulative reward. When the LHS of the above equation is not equal to RHS we can use this as a loss function to update the q value at time t with the Q-value at time t+1.

Therefore, we obtain the following loss function:

$$\mathcal{L}(s_t, a_t) = (R_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}) - q(s_t, a_t)) \tag{8}$$

We can then update the Q-value at time t with learning rate α using the following equation:

$$q^{new}(s_t, a_t) = q(s_t, a_t) + \alpha(R_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}) - q(s_t, a_t)) \\ \alpha \in (0, 1), \gamma \in (0, 1) \tag{9}$$

By Q-learning we seek to maximize the future cumulative reward by minimizing this loss $\mathcal{L}(s_t, a_t)$. Note that we use the q table to obtain $\max_{a_{t+1}} q(s_{t+1}, a_{t+1})$. Once the Q-table is learnt we can obtain the optimal policy by greedily taking the action with the maximum Q-value at each state.

3.1 Exploration vs. Exploitation

In Q-learning, since we do not have a policy π to obtain the values of Q-table, the agent has to explore the environment and also not make the same mistakes as it made in the previous episode. To do this, we use another hyper-parameter $\epsilon \in (0, 1)$. The agent takes a greedy action with ϵ probability and a random action with $1 - \epsilon$ probability. This strategy is usually referred to as ϵ**-greedy** strategy and allows the agent to explore as well as not repeat its previous mistake as the agent does not have a pre-given policy.

$$\pi(s) = \max_a q(s, a) \qquad \mathbb{P} = \epsilon \qquad \textbf{(exploitation)} \tag{10}$$

$$= random(a) \qquad \mathbb{P} = 1 - \epsilon \qquad \textbf{(exploration)} \tag{11}$$

A high ϵ value results in the agent taking greedy choices which might miss out an optimum path whereas, a low ϵ value can take longer number of episodes to learn the Q-table as the agent may barely reach the goal (in very complex environments). The ϵ value determines the trade-off wherein an agent might explore more by having a low ϵ value or might exploit (act greedily) and find a local optimal path with a high ϵ value.

4 Deep Q Learning

So far, we have considered discrete space problems. In real-life scenarios, there can be infinitely many state spaces and storing the action value for each state and its corresponding actions becomes infeasible. So what is the way out? If you haven't guessed it yet, then let me break it to you, it is deep learning.

4.1 Deep Q Networks

To perform Reinforcement learning in a Deep learning setting, the first seminal paper developed the idea of Deep Q Networks (Mnih et al., 2013). Deep Q Networks estimate the action value function $q(s, a)$ given the state s as input. The state can be multiple images, direction vectors, etc. This is similar to Q learning, but here, the network replaces the Q-table to provide the Q-value for all actions. This network is applicable for a *finite set* of actions as the number of prediction heads are same as the number of actions (Figure 1).

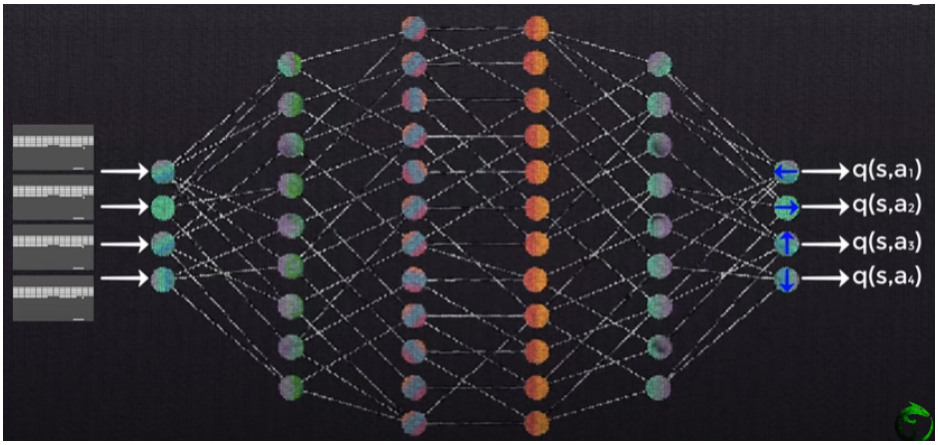

Figure 1: Deep Q Network: Given a state s as input, the network predicts the Q-values for all the actions taken at state s. Here, the Q-values are for the four actions up, down, left, and right. Image courtesy (Deeplizard, 2018)

Deep Q network minimizes over the same loss function as Q-learning, but here we use a network to do the same. The loss function is as given below:

$$\mathcal{L}_i(\theta_i) = \mathbb{E}[(R_{t+1} + \gamma \max_{a_{t+1}} q(s_{t+1}, a_{t+1}, \theta_{i-1}) - q(s_t, a_t, \theta_i))^2] \tag{12}$$

Here, θ_i represents the network weights at iteration i.

4.1.1 Fixed target Deep Q network

If we notice carefully, equation (12) uses the previous network weights θ_i for computing the next state's Q-values. This is the fixed target network whose weights are fixed, which is used as a supervisory signal while performing the i^{th} weight update. This fixed network is regularly updated after finite set of iterations.

Without the fixed target network, calculating the Q-values for state s_t and s_{t+1} might update the network in such a way, that as $q(s_t, a_t)$ moves closer to $q^*(s_t, a_t)$, the target $q^*(s_t, a_t)$ itself moves away from $q(s_t, a_t)$ (Deeplizard, 2018). This causes the outputs to chase its own tail and to avoid this a fixed target Q-network is used.

4.1.2 Experience Replay

During the journey of the agent in the environment, the observed states by the agent are highly correlated. This high correlation does not allow the deep learning model to update its weights appropriately, biasing it towards a particular action. To avoid this biased update, (Mnih et al., 2013) use an experience replay buffer that stores the observations and the outputs of the Deep Q

Network over multiple episodes. The authors store a tuple $e_t = (s_t, a_t, R_{t+1}, s_{t+1})$ into a finite data set $\mathcal{D}$, from which a batch of tuples are sampled randomly during training.

4.2 Deep Deterministic Policy Gradients

Deep Q Networks have a finite discrete action space that may not scale up to solve all the problems. Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al., 2019) allows Deep networks to regress to the most optimum action in a continuous action space. It uses an actor and a critic network. The actor network μ determines the optimum policy. Given the state s as input, the actor network obtains the optimal action a to be performed. The critic network takes in the action a and the state s as input and obtains the Q-value for performing action a in state s.

$$a_t = \mu(s_t|\theta^\mu) + \mathcal{N}_t \qquad \textbf{(Actor)} \tag{13}$$

$$y_t = q_(s_t, a_t|\theta^q) \qquad \textbf{(Critic)} \tag{14}$$

Here, θ^μ and θ^q are the network weights for the actor and critic networks respectively. The predicted action and Q-value are then supervised by cloned actor-critic fixed networks, as follows:

$$\hat{y}_t = R_t + \gamma q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})) \tag{15}$$

$$\mathcal{L}(\theta^\mu, \theta^q) = \mathbb{E}[(y_t(\theta^\mu, \theta^q) - \hat{y}_t)^2] \tag{16}$$

Here, $\theta^{\mu'}$ and $\theta^{q'}$ are the network weights for the **fixed** actor and **fixed** critic networks respectively. $\hat{y}_t$ is the supervisory signal and the gradients of the Q-value also flow through the actor network to update its weights as shown in Figure 2.

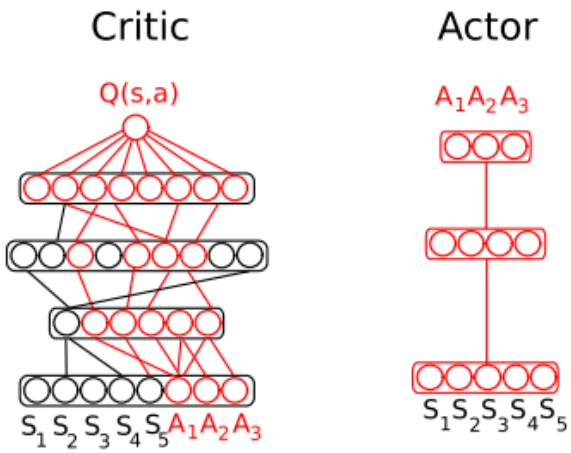

Figure 2: Deep Deterministic Policy Gradients: The actor network predicts the optimal action to take and the critic network determines the action value for taking action a from state s. Image courtesy (Olivier Sigaud, 2018)

5 Learning milestones and Source Seeking

5.1 Milestones up to now

Week 1 (December 24-31, 2020)

Implemented Q learning using blogs. Code available here.

Weeks 2-3 (January 1-14, 2021)

Covered basics of Reinforcement Learning from Deepmind lectures 1-5 (David Silver, 2015).

Week 4 (January 15-21, 2021)

Break

Week 5 (January 22-29, 2021)

Setting up AirSim on Blue simulator. I did set up the simulator on my laptop but then realized that I will require GPUs to train the network. To do this, I was trying to set up Air simulator on Blue RRC server. Airsim requires sudo privileges which are not given to users.
Problems faced: I am facing an issue setting up Airsim in the singularity container. I also found a bug and made a pull request on Microsoft Airsim's GitHub repository (which is merged! :). Link to pull request: `https://github.com/microsoft/AirSim/pull/3370`).

Week 6 (February 3-10, 2021)

Understanding Deep Q Networks and Deep Deterministic Policy Gradients. Month 1 report submission.

5.2 Source Seeking

I plan to start training the first RL drone agent soon after the Airsim issues are solved. I was able to capture the images of the environment from the drone. To train the network, I plan to use 4 images of the environment, the depth map, and the direction vector to the source of the signal as the state variable. The RGB images will provide the network information about how the drone is moving in the world. The depth map gives the agent an idea about where the obstacles are in the environment. The direction vector $D = \vec{P}_{source} - \vec{P}_{drone}$ provides the direction of the source of the signal to the network. Here, $\vec{P}_{source}$ and $\vec{P}_{drone}$ are the positions of the source of the signal and drone respectively. I am unsure about what should be the intermediate reward to provide at each time step. A collision can be a high negative reward. A very naive rewarding system here would be to give positive reward as the distance between the source and drone reduces.

6 Acknowledgement

Thank you Dr. Harikumar Kandath sir for guiding me through the learning process by providing me a learning path to cover. Thank you David Silver and DeepMind for making Reinforcement Learning lectures available to the globe.

References

Website link David Silver. 2015. Reinforcement learning lectures.

https://www.youtube.com/watch?v=xVkPh9E9GfE Deeplizard. 2018. Training a deep q-network with fixed q-targets - reinforcement learning.

Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2019. Continuous control with deep reinforcement learning.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning.

http://pages.isir.upmc.fr/~sigaud/teach/ddpg.pdf Olivier Sigaud. 2018. Deep deterministic policy gradient lecture resources.