Deep RL Algorithm Performance for Resource Collection Maximization

Bofu Dong, Yanzheng Wu, Luxi Liu

1 Introduction

The current reinforcement Learning(RL) can be classified into two distinct fields: Model-Based RL and Model-Free RL. The model-based RL relies on the interaction with the environment and it utilizes the immediate outcomes from such interactions to choose the next action to maximize the potential rewards. The model-free RL on the other hand does not heavily depend on the environment. It learns from the given policy and updates the state-action value function until the agent reached a terminated state.

Granted, each method has its pros and cons, for instance, model-free methods are usually less efficient than model-based methods but sometimes the environment model might not be given to us hence limiting the use of model-based methods. However, as the environment model or agent behavior variables becomes more complicated, neither of the two fields could perform perfectly standalone. Hence, it is not ideal to only adopt one single learning method to solve some real-world problems.

Now, imagine if we put a puppy into an unknown environment, the puppy might first gather information about the surrounding through vision, hearing, sniffing, or other senses. After the puppy has a rough sense of its surrounding, it might then feel more confident exploring around to have a better understanding of the new environment. Hopefully, the dog could find some treats or toys from exploration for the sake of survival but the point is that no prior knowledge about the environment is given to the subject or any specified goal state(in the dog example the only ultimate goal is to survive). Can an RL agent mimic the dog's process of exploration in an open-world environment and learned by itself to maximize collecting resources?

We define the open-world environment as non-spontaneous, i.e., any resource object within the environment only yields the intrinsic reward when interacting with the agent and the reward is non-deterministic relating to the agent's actual needs. For instance, if the agent is thirsty then finding a pond to drink could give more rewards than other actions or states.

From solving and exploring the resource collection maximization problem in an open-world environment, we would explore the potential of RL incorporating other types of deep learning methods to develop better means to solve more complex real-world problems that previously would be too overwhelming to solve using only RL methods.

2 Background Related Work

The combination of RL with Deep Neural Network was proved to be successful in many fields, including but not limited to robotics, video games, natural language processing, computer vision, education, transportation, finance and healthcare.[1]

In many practical decision making problems, the state space of Markov Decision Process (MDP) are high dimensional. This means, that with deep learning, Reinforcement Learning is able to solve more complicated tasks with lower prior knowledge because of its ability to learn different levels of abstractions from data.

In a complex problem, deep reinforcement learning replaces tabular methods of estimating state values with function approximation. Function approximation not only eliminates the need to store all state-value pairs, and also enables agent to generalize the state-value pairs that are unseen.

2.1 Experience replay

Experience replay[2] is a technique that allows agent to 'mentally experience' the effects of its actions without actually executing them, and is shown to be effective in reducing the number of action executions required.

Experience replay is used not only to improve the efficiency of network, but is also needed for the independent and identically distributed assumption of neural networks, because a naive Q-learning algorithm is often strongly affected by the correlation between the sequence of experiences during learning.

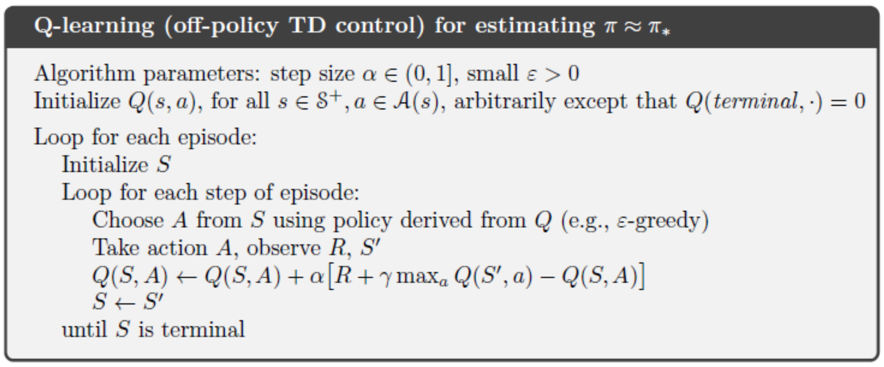
2.2 target network

The fact that Q-Learning updates a guess with a guess could potentially cause wrong correlations, we want to make the training of our neural network more stable by using target network.

This idea comes from the fact that one can interleave acting and learning process in experience replay. Target network is a technique that we keep a copy of our neural network and use it for $Q(S', A')$ value - the predicted Qvalues of this second Q-network is called target network, and are used to backpropagate to the main Q-network.[3]

3 Technical Approach / Methodology / Theoretical Framework

3.1 Q-Learning



Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

```
Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
    Initialize $S$
    Loop for each step of episode:
        Choose $A$ from $S$ using policy derived from $Q$ (e.g., $\varepsilon$-greedy)
        Take action $A$, observe $R$, $S'$
        $Q(S, A) \leftarrow Q(S, A) + \alpha\big[R + \gamma \max_a Q(S', a) - Q(S, A)\big]$
        $S \leftarrow S'$
    until $S$ is terminal
```

At each step, the agent tries to move on the map, selecting from the simple action set {left, right, up, down}. Q-Learning will try to learn the state-action values for positions on the map. Each position is either empty (-1 reward), or has an item (a positive reward). We choose to not discount rewards. At each state S, we take epsilon greedy action A (a move), collect reward, goes to S', and update the Q value for state S action A. We will try different learning rate to optimize our Q-Learning method. After the learning, we generate the optimal path to see how many steps it took the agent to collect all items on the map.

3.2 Deep Q-Network

A deep learning method shares same action set, and information on the environment as above. The sole difference is that for the deep Q-network, it learns a function that allows agent to generalized on Q-value. The overall implementation algorithm is as follows:[5]

```
Initialize network $Q$
Initialize target network $\hat{Q}$
Initialize experience replay memory $D$
Initialize the Agent to interact with the Environment
while not converged do
    /* Sample phase
    $\epsilon \leftarrow$ setting new epsilon with $\epsilon$-decay
    Choose an action $a$ from state $s$ using policy $\epsilon$-greedy($Q$)
    Agent takes action $a$, observe reward $r$, and next state $s'$
    Store transition $(s, a, r, s', done)$ in the experience replay memory $D$

    if enough experiences in $D$ then
        /* Learn phase
        Sample a random minibatch of $N$ transitions from $D$
        for every transition $(s_i, a_i, r_i, s'_i, done_i)$ in minibatch do
            if $done_i$ then
                $y_i = r_i$
            else
                $y_i = r_i + \gamma \max_{a' \in \mathcal{A}} \hat{Q}(s'_i, a')$
            end
        end
        Calculate the loss $\mathcal{L}$ =1/N $\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2$
        Update $Q$ using the SGD algorithm by minimizing the loss $\mathcal{L}$
        Every $C$ steps, copy weights from $Q$ to $\hat{Q}$
    end
end
```

3.2.1 Experience replay

A large buffer of the past experience and sample training data from the experience is used instead of the latest experience to prevent oscillating or diverging action values. This technique is called experience buffer, and it contains a collection of experience tuples: (S, A, R, S'). The tuples are gradually added to the environment as the agent interact with the environment. The implementation of this is a buffer of fixed size, with the oldest experience replaced by the new input.[3]

3.2.2 Target network

The initialization of target network is simply copying the same initial network, and in implementation, the target network's parameters are not trained, but they are periodically synchronized with the parameters of the main Q-network. [5]

4 Evaluation

We seek to evaluate the performance of Tabular Methods vs. Function Approximation.

We will compare different agents participating in the game, which could be summarized as:

1. Special agent: using Neural Network to determine best action.

2. Conventional Q-learning agent.

Assuming our task is to collect items on the map, at each step we have an undiscounted reward of -1, until we collect all the items.

We will compare:

1. Time, including training speed and action speed. Action speed would be how many steps does it take to retrieve everything on the map. We will seek to improve the runtime for deep learning by, for instance, simplify the image information.

2. Memory usage.

We will also compare the performance on small grid and big grid.

We expect deep learning to perform better on bigger, more complex map. We expect that if there are too many states to track/we have a large variety of different screens, some problems might not be solvable with a Q-table.[4] The actual effect will be decided by our experiments.

5 Timeline and Individual Responsibilities

5.1 Responsibilities:

- Yanzheng Wu: Introduction and background research, fetching for related datasets;
- Bofu Dong: Background research, and look into papers on deep RL algorithms, structure and coding of deep NN;
- Luxi Liu: set up the n-step sarsa agent/other TD(n) learning agents. Compare metrics for performance across agents.

5.2 Timeline:

	Yanzheng Wu	Bofu Dong	Luxi Liu
Week 1	Introduction research	Background and related work checking	Methodology Research
Week 2	CNN model selection	Implementing experience replay	Selecting RL Algorithm
Week 3	NN implementation	Implement the network	Improve RL models by adjusting parameters
Week 4	Hyper-tuning NN	Implement the network, and train and test on data	make table or graph for comparisons
Week 5	report revision	finish the report	finish writing report
Week 6	Prepare presentation	Prepare presentation	Prepare presentation

References

[1] Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Found. Trends Mach. Learn.*, 11:219–354, 2018.

[2] Longxin Lin. Reinforcement learning for robots using neural networks. 1992.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013.

[4] Jordi Torres. Deep q-network (dqn)-i. 2020.

[5] Jordi Torres. Deep q-network (dqn)-ii, experience replay and target networks. 2020.