DEEP REINFORCEMENT LEARNING

Ziqi XU

ICM 2A EMSE

Vocabulary

Screen of maze

A representation of the maze, either in 3D array or transformed into a 1D array. A screen may be resized from the original maze (640 * 640 * 3) since it is too complicated to handle large arrays.

Episode

A list of previous state, action, reward, state.

Experience class

A class implemented in the first approach. `Remember()` function stores episodes into memory, `get_data()` takes some samples from the memory and calculates their outputs.

Step

One step means that the agent takes a move in the maze

Epoch

One epoch means that the agent has reached the goal from the beginning or the steps have run out.

Introduction

This report is about applying deep reinforcement learning method to solve the maze problem. The maze size can vary from 10 by 10 to 100 by 100. Classical reinforcement learnings like Q-learning can solve small mazes, but they cannot deal with large ones. This is why it should apply deep reinforcement learning methods. In this report, **DQN** will be implemented.

First approach

Since the maze problem is a third-party environment of Gym, there are few available online resources that discuss our problem. For the first approach, I have referenced the following work (https://www.samyzaf.com/ML/rl/qmaze.html). In this post, the maze is represented as a NumPy array, the Experience class stores different screens of the maze (each screen of maze is transformed into a 1D array). The following paragraph represents the training process.

1. Build neuron network (Keras)

A simple linear network, three linear layers with two activation layers.

2. Get screen of maze

Using env.render() to get a 3D array of the maze, then resize it and transform into a 1D array.

3. Get state, action and reward

4. Store an episode into memory

An episode consists of previous state, action, reward, state.

5. Get inputs and targets for neuron network

Retrieve a batch of screens from memory, then calculates their targets using Bellman equation.

6. Start training neuron network

Loop until the agent reaches goal, then reset the maze and run another epoch

The result is not satisfying due to the problem of **Keras** on my computer. In fact, the CPU usage is around 30% and the fan is not even working. In addition, the algorithm cannot handle large screen of maze, a 10 by 10 screen will work but not with 100 by 100.

I have rewritten the program using **PyTorch** framework, and this time the CPU usage is nearly 100%, the dot on the screen is moving faster. However, it still takes a long time to solve a 10 by 10 maze, which means the algorithm cannot find the goal within 10000 steps in the beginning of the training. In addition, the agent seems to be stuck in certain locations and does not know how to get out, which means the network has been overfitted or the reward function is not accurate.

I have tried to modify the reward function, by adding a positive reward when the agent gets closer to the goal. The result is not satisfying either, because the agent tends to explore only the left part of the maze, without reaching the goal. We need a better version of reward function so that the agent can understand the maze better.

Second approach

By comparing the first approach with some online solutions for other RL tasks (https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html), I have noticed a few improvements.

1. We need two separate networks: **target network** and **policy network**. They are exactly the same at the beginning, but the parameters of policy network are copied to the target network every 10 epochs. By doing so, we can fix the target network for a while, which helps the convergence of the training.

2. We need a **CNN** network to handle screens of maze, since neuron networks can learn the features from images better. I have implemented the CNN from the link above, with three 2D convolution layers and ReLU functions between them.

3. The **exploration rate** should be larger at the beginning of the training, allowing the algorithm to discover more about the maze. In this second approach, the exploration rate is set to decay from 0.9 to 0.7 for the first two epochs, and exponentially decay from 0.9 to 0.1 for the rest epochs.

4. Make sure we have enough samples in the memory before training the network. The optimization function will return if the length of memory is shorter than the required size (128).

5. The implementation of **CUDA** should accelerate the training.

This time, I have tested with the 50 by 50 maze. The speed of the training is satisfying, running 50000 steps in 30 minutes (the program crashed when using Q-learning on 50 by 50 maze). Here is the result of the training. Timed out step is set to 62500.

Epoch	Steps taken
0	Timed out
1	55053
2	Timed out
3	18512
4	48368
5	29710

6	34685
7	10953
8	30456
9	27203

From the table above, we can see that the number of steps has not been reduced during the training, and when the parameters of policy network is copied to target network, the agent is again stuck at certain locations. It seems that this sophisticated is not capable of guiding the agent around the maze either.

Difficulties and Improvements

1. Since the network knows nothing about the maze, its performance is quite poor in the first few epochs, yet it takes hours to run just a few epochs even with CUDA activated.

2. The reward function should also be rewritten, so that it adapts to the maze environment.

3. There are some RL libraries such as Keras-rl and Cherry-rl that facilitates the training, but since the maze problem is a third-party environment, it takes some time to implement these libraries. In addition, there are other networks that worth a try, such as DDPG. Due to the limitation of time and computational resources, I did not implement other networks.

Conclusion

Training a DQN network is even more difficult than a Deep Learning network, since it takes a lot of time to wait for the agent reaches the goal. For this particular maze problem, it may be more suitable to implement some traditional methods, such a Breadth First Search or A* algorithm (of course, the maze should be modified).

References

[1] Deep Reinforcement Learning for Maze Solving,
https://www.samyzaf.com/ML/rl/qmaze.html

[2] Reinforcement Learning (DQN) Tutorial, Adam Paszke,
https://pytorch.org/tutorials/intermediate/reinforcement_q_learning.html