CS5446 AI Planning and Decision Making

HW3 Task 2: Solving Large Stochastic Environment

NUS
National University
of Singapore

By Tey Shiwei A0112101M and Chua Chin Siang A0112089

Problem Definition:

Compare to Task 1, the provided training examples are relatively huge which requires the agent to take around 40~50 steps to goal. The other constrain is that the agent must have to solve the large stochastic environment within 600 seconds.

The second criteria imply that we cannot create a learner that learns the environment with infinite time. Therefore, we are creating a function approximation Convolutional Network model to be train with the current environment we have and let it perform on the test cases provided.

Method

The first thing we do is to try out all available methods online, such as ActorCritic method, DQN (from Task 1) and DDQN. It works well on the environment provided in Task1 but failed for Task 2.

Temporary results of the online methods that was mentioned

Task 1 (Small env): It doesn't work well in the early episodes but improves drastically once the model learnt the pattern. It succeeded most of the time after many iterations.

Task 2 (Large env): It doesn't work well in the entire learning phase. Succeeded once of twice but I guess it was due to luck.

Improvements

Hyperparameters

Hyperparameter might be a key to solve our issue. We tweaked the parameters and increase the number of training episodes to a very large number and the model should work well given an infinite training time.

Results: However, the improvements are very minimal. It doesn't grows linearly with the number of episodes. Same for learning rates, discounting factor, number of hidden layers in the model etc.

Define: channel, h, w = env.observation_space.shape
col and row represents the vertical and horizontal steps for agent to reach the goal
Normalizer = Z = 0.2/(total sum of rewards)

Reaching the highest layer safely

The second idea came up by giving small rewards when the agent is able to avoid car crashing. Hence, we started off by giving out rewards if the agent is going "UP" without crashing. Once the agent reached the top layer safely, it will definitely reach the goal safely by FORWARD3.

Scoring method: For agent moving to a higher level, it gains higher rewards. Staying at the same level getting the same reward. Moving down will gain 0 reward.

Results: As this is a large environment, not all situation favours going UP.

Distance to Goal

Focusing on the agent to move as far as possible. Giving it reward if it moves.

Scoring method: $Reward_{distance} = [\ (w - row + 1) + \ (h - col + 1)\ ] * Z$

Normalized Distance to Goal

Rewarding the agent by the number of grids to the goal state, normalized by width and height respectively. Max reward set it to be 0.2 to prevent agent prioritizing moving further than reaching the goal.

Scoring method: $Reward_{distance} = \left[\frac{w-row+1}{w} + \frac{h-col+1}{h}\right] * Z$

Results: It improves the model drastically and achieved 6.6 points

More weights to vertical movements

```
bonus = ( (h-(col+1))*factor + w -(row+1) )*normalizer, normalizer = 0.2; factor = 1.3
fw3: 0.042
fw2: 0.028
up: 0.032
fw1: 0.014
down: 0.004
[[0.2   0.186 0.172 0.158 0.144 0.13  0.115 0.101 0.087 0.073]
 [0.182 0.168 0.154 0.139 0.125 0.111 0.097 0.083 0.069 0.055]
 [0.163 0.149 0.135 0.121 0.107 0.093 0.079 0.065 0.051 0.037]
 [0.145 0.131 0.117 0.103 0.089 0.075 0.061 0.046 0.032 0.018]
 [0.127 0.113 0.099 0.085 0.07  0.056 0.042 0.028 0.014 0.   ]]
```

Moving fw3 giving out the most score, followed by UP, fw2, fw1 and down. As long as the agent moves and do not crash, it gains reward.

Results: Not so good as the agent is moving everywhere and not towards the goal.

Cliff effect

```
bonus = ( (h-(col+1))*factor + w -(row+1) )*normalizer, normalizer = 0.2; factor = 1
fw3: 0.046
up: 0.031
fw2: 0.031
fw1: 0.015
down: 0.000
[[ 0.2    0.185  0.169  0.154  0.138  0.123  0.108  0.092  0.077  0.062]
 [-1.     0.169  0.154  0.138  0.123  0.108  0.092  0.077  0.062  0.046]
 [-1.    -1.     0.138  0.123  0.108  0.092  0.077  0.062  0.046  0.031]
 [-1.    -1.    -1.     0.108  0.092  0.077  0.062  0.046  0.031  0.015]
 [-1.    -1.    -1.    -1.     0.077  0.062  0.046  0.031  0.015  0.   ]]
```

Giving out penalty if the agent reached the area ("-1"). Because if the agent reaches the positions stated "-1", it will not reach the goal regardless of the remaining actions.

Results: During training, we realized that the agent keep reaching the "cliff" and avoiding reaching the goal.

Focus on reaching the safe area (Highlighted in yellow)

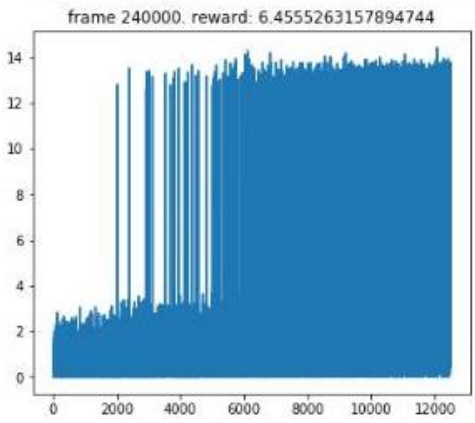
```
bonus = ( (h-(col+1))*factor + w -(row+1) )*normalizer, normalizer = 0.2; factor = 3
fw3: 0.029
up: 0.038
fw2: 0.019
fw1: 0.010
down: 0.019
[[ 0.2    0.19   0.181  0.171  0.162  0.152  0.143  0.133  0.124  0.114]
 [-1.     0.162  0.152  0.143  0.133  0.124  0.114  0.105  0.095  0.086]
 [-1.    -1.     0.124  0.114  0.105  0.095  0.086  0.076  0.067  0.057]
 [-1.    -1.    -1.     0.086  0.076  0.067  0.057  0.048  0.038  0.029]
 [-1.    -1.    -1.    -1.     0.048  0.038  0.029  0.019  0.01   0.   ]]
```

Encouraging the agent to move towards the safe area by giving out higher rewards at those positions.

Results: Still improving, doesn't get a very good improvement yet.

Final results

- Focuses on tuning factors, normalizers, penalties and hyperparameters of the model setup
- Rewards shaping is very important, in a long run, it could improve the overall score of the model as shown



- The first training results can get 60% of correct test results
- Re-training the best model helps to further improve the test accuracy with some of the following trick
    - Low starting range of epsilon during retraining
    - Reduces epsilon when the average rewards get higher

The best score we could achieve is 8.784:

- Tmax_50: 8.7
- Tmax_40: 8.867