Reward function:

The reward function influences the effect of reinforcement learning. The original reward function is not very powerful. So we redefine a reward function like this:

if the car is near the flag:

cancel the -1 penalty

We thought that this could encourage the car to recognize this area as the reward zone. As a result, it will prefer to come back to this area and may go further to reach the goal. This does work. However, it has one drawback that it may take over one hundred episodes to get the flag once. What’s more, due to the reward function, reaching the flag is not that attractive. The total score may only a small number different from those not reaching the goal. Therefore, it may take several other episodes to get to the top of the mountain again. In total, this process will take several hundred episodes to reach the flag stably. This is a long time training process.

Then we change the reward function to speed up this process like this:

if the car is away from its start position:

the farther, give the more reward

The start position of the car is a uniform random value in [-0.6, -0.4]. We design this function for two reasons. The first one is the goal position, 0.5, which is far away from the start position. The other reason is that this small car cannot go all the way right and get the flag. It needs to go a little leftwards and then go back to reach the top. The more left it goes, it shall have more gravitational potential energy. Therefore, we are also glad to see the car goes left. In this way, this reward function clearly reflects our design. It speeds up the learning a lot. Usually, the car can reach the flag within one hundred episodes. The car is more willing to go to the higher region. But the stability problem still exists. We decide to give an attractive bonus to the goal to enhance the memory. So we add one more line after the if statement:

if the car reaches the destination:

give a big bonus

This statement has two effects. One is it generally solves the stability problem I mentioned before. The other one is the score is outstanding enough to be used as the signal to end the training early. If the last ten scores are all very large, we consider this model as a stable and successful one.

Epsilon:

Function run() decides the next action. When in the start state, the actions are selected randomly. After that, we should still ensure a possibility that the car can move randomly. At first, we write the code like this:

if random[0, 1] > epsilon -\*episode:

random select action

else:

use the prediction of the current state

The epsilon equals 0.1. But since the possibility is too little, there is very little freedom to explore. Due to this kind of lack of exploration at the beginning, the learning effect is not that good. Later we change to another function. We define an initial epsilon as 1. Then we define the epsilon\_dcey and min\_epsilon.

if random[0, 1] > epsilon:

random select action

if epsilon > min\_ epsilon:

epsilon \*= epsilon\_dcey

else:

use the prediction of the current state

This function provides more freedom when the agent just starts. As time goes by, the possibility reduces but has a min value, which ensures we will always have random actions. This can help the car recognizes the rewards of climbing high and speed up the learning process.

Here are our training results. This figure is the score list while training. The right one is the saved model. Our model ensures the car can reach the flag within one or two episodes.

That’s the end of our pre3. Thank you. Do you have any questions?