Modular Reinforcement Learning on a Toy Example

Abrar Anwar
aa76875
University of Texas at Austin
abraranwar123@gmail.com

Abstract

Q-learning in a large state space with multiple types of goal can be split into several modules that accomplish different goals. These modules can be combined together so that a cumulative decision can be made that brings an agent closer to all the goals at once. A toy environment is designed to simulate picking up litter and avoiding obstacles. This environment is used to show off the capabilities of a modular reinforcement learning approach using Q-learning. The example shows a modular approach as a capable method to solving reinforcement learning problems.

1. Introduction

1.1. Q-Learning

A Markov Decision Process (MDP) is a form of Markov model where we are given a series of states, actions, transition probabilities, and a reward. Q-learning is a model-free approach to reinforcement learning that does not require a model of the environment and can deal with stochastic environments. The Q-learning update formula is as follows:

$$Q(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$$

The result for a given Q(s,a) can be seen as the expected value of total reward received. The discount factor γ represents how much to value rewards. As iterations are increased, there is an exponentiation factor to γ, causing future rewards to be weighted lesser than closer ones. The latter half of the equation can be seen as the temporal difference, where we are trying to predict future values given the current state. This helps train our Q function to be better. A fundamental issue with Q-learning in certain cases can be incredibly large state spaces, which can cause this approach to be potentially bad.

Regardless of the size of the state space, the exploitation-exploration problem must be solved. This can be briefly summarized as given a model that might be able to choose the best known action given the current state, can we explore the environment so that we are able to update our model to be better. In order to do this, epsilon-greedy aims to choose an action from the current model with a probability of $1 - \epsilon$, otherwise with a probability of ϵ, choose a random action to force exploration of the environment. A starting value for ϵ could be 0.1, and have the value decay over each episode until it reaches some maximum like .9. This means each training iteration towards the end, a random action would be selected randomly at a 10% probability, allowing for some learning in more complex environments.

1.2. Modular Reinforcement Learning

In cases where there are multiple goals for a reinforcement learning agent to learn, the number of potential states could grow increasingly large. To combat this, modular reinforcement learning aims to split each goal off into separate modules, each of which are able to come up with the best action for their own goals. Each of these modules have Q-functions. Normalized Q-functions by state for each module can be used to calculate a weighted Q matrix that can be used to select the true best action that is able to accomplish all the goals.

2. Method

2.1. Environment

For our toy example, the environment is a 6x25 grid world, as we see in Figure 1. There exists 3 sidewalks where the agent ideally stays in. Furthermore, there exists multiple obstacles that are generated on the map at a 1/5 probability at any given position. With the same probability, litter is scattered around the map as well.

2.2. Model and Rewards Description

The state space of the model is the 4 directions of the agent and whether what we are looking for is there or not. This causes the state space to be really small, as a modular output only deals with an item being there or not in each

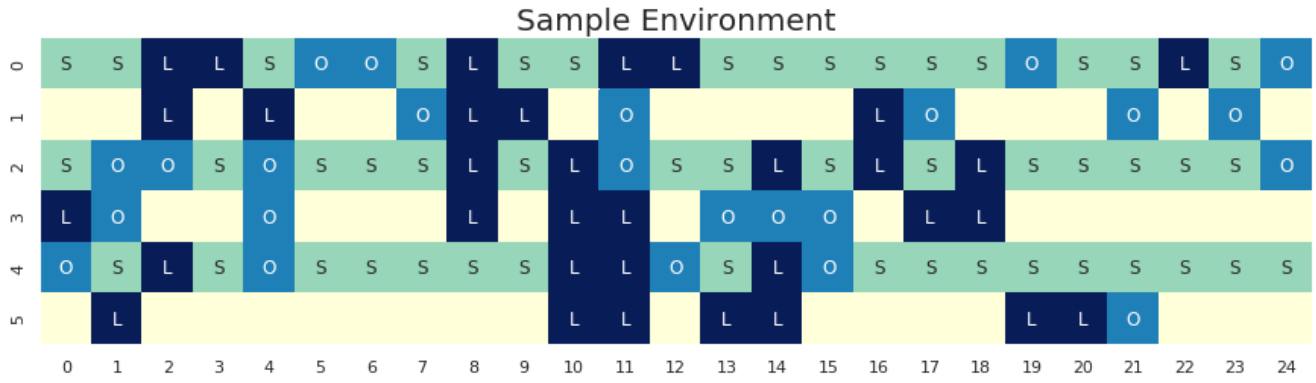

Figure 1. Grid world environment where 'S' represents the sidewalk (it continues for the whole y axis even if it's not apparent). 'O' represents an obstacle and 'L' represents litter

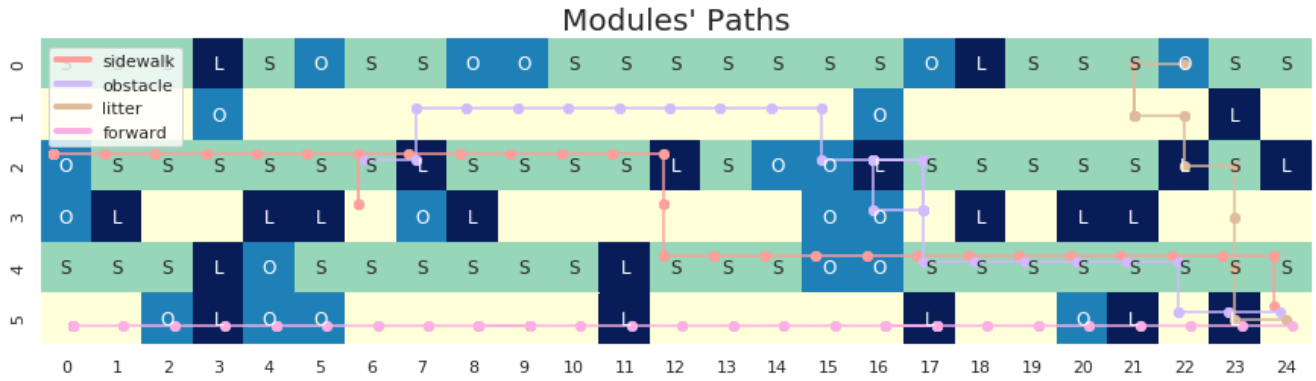

Figure 2. The four modules' generated paths in the environment labeled

direction, we end up having a state space of size $2^4 = 16$. If no modular approach was used, the state space would have been $3^4 = 27$, which is large and would increase faster if more grids were used.

There exists 4 types of rewards for the 4 modules that are going to be built. The first one incentivizes the agent to remain on the sidewalk, the second one avoids obstacles, the third picks up litter, and the fourth one encourages reaching the end of the sidewalk (x=0).

The states for a sidewalk are defined as if the adjacent cells are sidewalks, the relevant bits that represent that direction is set to 1. The staying on sidewalk goal's rewards are if it takes a left onto a sidewalk, it gets a reward of +10, and a right on a sidewalk is -1. This is to provide a slight incentive to going towards the left. In addition, a reward of -10 is received if the action takes the agent off of the sidewalk.

If an obstacle is in an adjacent cell, the bits that represent that state are set. The obstacle sidewalk is defined as follows: if the agent stumbles on a sidewalk, a reward of -10 is received. If the agent is not on a sidewalk and takes a left, a positive reward of +3.5 is received, whereas a left gives a reward of -2.5. This is to encourage leftward behavior.

For picking up litter, if the agent stumbles on litter, the litter is removed from the environment. Others had concerns that removing an item would cause the state space to explode; however, if the state space is only defined as what the agent sees in it's local vicinity, so there is no global state explosion we need to worry about. The reward for picking up litter while moving left is +0.5, while moving right is -2.5. If no litter is picked up, a reward of -3 is received.

The forward moving module is interesting. The state is continually 0, and the agent is meant to realize that moving forward is the only good action to take. This last module is a forward moving module that causes the agent to reach the end of the sidewalk. The reward for this gives a reward of +10 for moving left and -10 for moving right. If the end of the sidewalk is reached, a reward of +50 is used.

It is clear that these rewards are all in significantly different scales, which effects the scales of each of the Q functions. Due to this, we normalize each of the states by their L2-norm in each modules Q function.

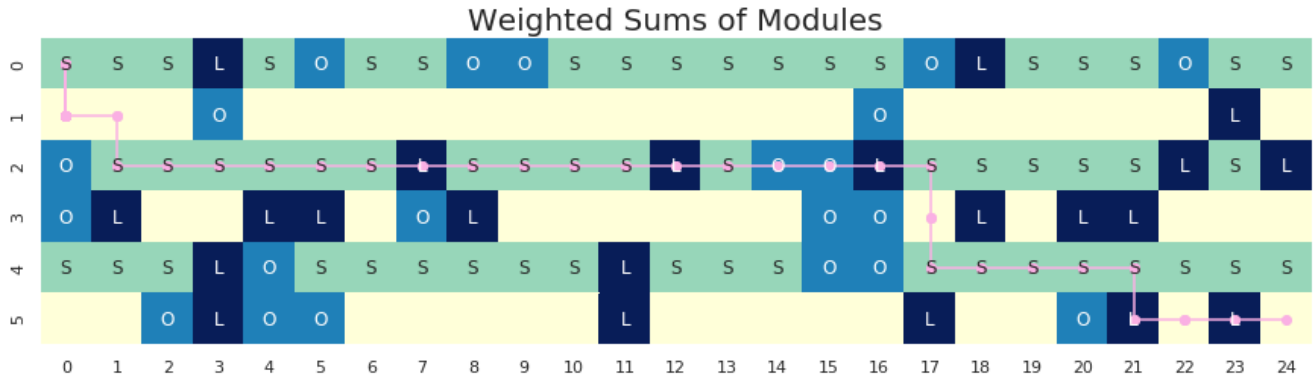

Figure 3. Path of the actions generated by a weighted combination of the modules with weights [0.2,0.4,0.3,0.1] for sidewalk, obstacle, litter, forward respectively

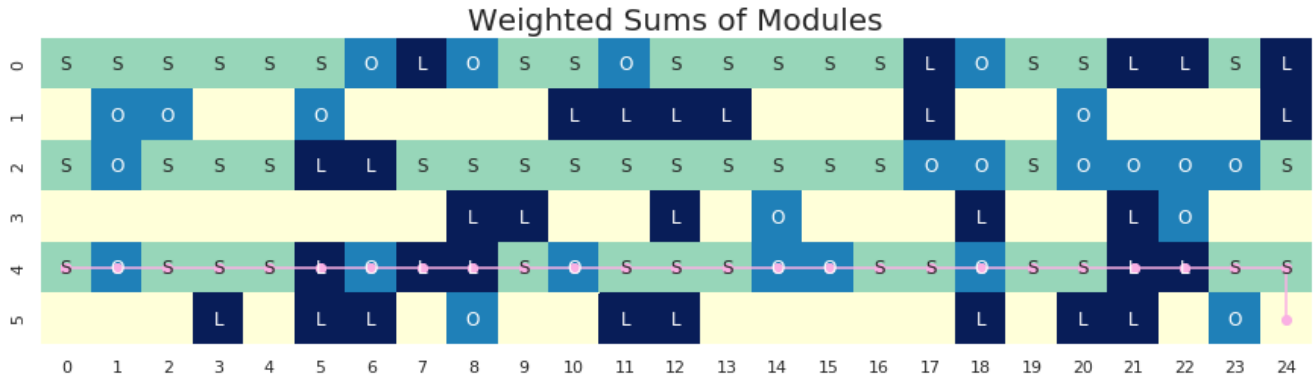

Figure 4. Issues seen by path of the actions generated by weighted combination of the modules with weights [0.4,0.2,0.3,0.1] for sidewalk, obstacle, litter, forward respectively

3. Results

4. Individual Modules

Figure 2, the sample generated paths of each of the modules. Each of the modules show the specific goal it is trying to achieve. We can see the forward module is continues to propell the agent forward towards the end goal, as expected. The sidewalk agent follows the sidewalk pretty regularly. Random motions away from the sidewalk occur not due to issues in the Q function, but rather the fact an epsilon of 0.9 is kept throughout, causing the agent to make random actions at a probability of 10%. This is due to the agent potentially getting stuck in some areas, specifically for the obstacle module. We can see the obstacle module, which is always incentivized to avoid obstacles, goes through some obstacles at around 17 in the figure. This is likely due to a random action taken to get it through there. If no random action was taken, the agent would have repeated in circles indefinitely. Lastly, the litter agent is the most interesting. It is able to find litter, but due to a lack of good incentive to move forward, it finds litter through random exploration; however, it can get stuck, especially on the boundaries of the environment, which are not defined as anything. It continually tries to make motions through the boundaries on occasion.

The issues with each of the modules are meant to be solved through a weighted combination of all the modules.

5. Weighted Modules

As seen in the previous section, each module is unable to succeed in its goal. A weighted combination of the modules can be seen in Figure 3. The weights are as follows for the policy in the figure: .2 for staying on the sidewalk, .4 for avoid obstacles, .3 for picking up litter, and .1 for moving forward. This combination is meant to put an importance in avoiding obstacles; however, it will at the same time pick up litter, and stay on the sidewalk. We can see in around 17, it picks up the litter, and continues to stay on the sidewalk as it continues on to its goal. Also, the litter module is able to function properly without much of a desire to spread randomly due to the module that prefers forward motion, along with all the other modules having rewards for forward motion. The interaction between all these modules provide an interesting benefit, as the goals of all of them can be met at

once.

This method however is not perfect. As we see in Figure 4, for a specific path being generated in a different environment, a straight path through all the obstacles was generated. This is because a large weight was decided to be put on the sidewalk (0.4) and a lesser one on the obstacles (0.2). It appears it becomes easy for one type of policy to dominate the weighted combination of policies, thus the weights need to be fine-tuned manually.

6. Summary

Although a regular Q-learning example could easily represent this model due to the state space, it shows a proof of concept of the modularization of different goals into independently trained modules. The combination of these modules provide behaviors that accomplish all the goals that we want, and the weights on each module can show a preference to one goal over another. This is good when the weights are finetuned and small; however, this approach may be difficult to scale to a large number of modules due to the fact it seems like it's easy for one module to dominate over others, even if the Q-values of each module are all normalized.