IAR

Chetanveer GOBIN, Alexandre BACH, Esteban AVILA, Kenza BENLAMLIH

October 2022

1 Introduction

This project's objective is to study Reinforcement Learning and to gain a valuable insight about value iteration, Q learning, and to apply it to the game of PacMan. We followed the homework's instructions and got to implement the algorithms given by the UC Berkeley. We also scaled our Q learning implementation to approximate Q learning and engineered additional features for approximate Q learning. We also demonstrate that our additional engineered features give better performance than the predefined ones.

2 Modelling the problem as an MDP

In this setting, we model our problem as a Markov Decision Process(MDP). In this model, this means that the transition probability to the next state depends only on the current state and not on the history.

- **State space** In this first example, the state space that is represented by a grid world and this can be interpreted as a system of Cartesian 2D plane. The state of an agent is its position that is equivalent to a Cartesian coordinate (x, y). Its initial state and position do not change. We denote the state space by S.
- **Action space** The actions are the directions the agent can move. In our case these actions are North, East, South and West. The action space is denoted by A.
- **Transition probabilities** When we move from state s to state s', when taking action a, we denote the transition probability as $P(s, a, s')$. There are two cases. Either, we have access to a transition probability matrix and we have all the information needed. This is given by the MDP. Or we don't have access to the transition probabilities and in that case we need to estimate them using a learning method such as Q learning.
- **Reward function** The reward function depends entirely on how we set the problem and how we want to achieve our desired objective. It is denoted by $R(s, a, s')$.

Note that for the rest of this technical report we use the same notation.

3 Value Iteration

The first algorithm that we studied is Value Iteration algorithm. This approach is used when we want to find the value of a state. That is the total expected discounted reward that one can expect if they are at a particular state s and they act optimally from there. We note this as $V_{\pi^*}(s)$, where π^* denotes the optimal policy.
The value iteration algorithm, falls under the paradigm of dynamic programming and can only be used when we have full information about transition probabilities in the MDP. The optimal value of a state s can be computed by using the Bellman equation (1).

$$V(s) = \max_a \left(R(s, a) + \gamma \sum_{s'} P(s, a, s')V(s') \right) \tag{1}$$

This value is given by equation (1). These state values can be computed by the algorithm 1

3.1 Gridworld Toy Example

Before testing out value iteration algorithm on a Pacman game directly, we first try it on the toy gridworld example. The implementation can be seen in the attached python program.

Value Iteration

Algorithm parameters: discount $\gamma = 0.9$, number of iterations
Initialize V^π as an arbitrary value, $V_T^\pi = 0$
for Number of iteration **do**
    Initialize t = 0 and s_0
    for $t = 0, 1, ..., T-1$ **do**
        Choose a_t as an action given by π for s_t
        Take action a_t, observe r_t and s_{t+1}
        $V_t^\pi(s_t) = V_t^\pi(s_t) + \alpha(\mathrm{r}_t + \mathrm{V}_{t+1}^\pi(s_{t+1})$ - $V_t^\pi(s_t)$
return $V_T^\pi(s_{T-1})$

3.1.1 Observation

When we let our value iteration run for a relatively large number of iteration. For example, with 100 iterations (which is good enough for a gridworld example), we could observe that the value of each state had a fixed value, which proves that our algorithm converges to the optimal value. Figure 1 shows the end result.

Figure 1: Value functions for 100 iterations.

4 Q learning

Q learning, is the first Reinforcement Learning(RL) algorithm that we studied. This algorithm is used when the transition probabilities of the MDP are not available to us. In that case, we need a new approach called Q learning. It consists of finding the Q-values. A Q-value is the total expected reward from state s and taking action a, and there is the optimal Q-value which is acting optimally from that state s. The computation of Q-values is given by the equation and algorithm below.

$$Q(S, A) = Q(S, A) + \alpha[R + \gamma Q(S', A) - Q(S, A) \tag{2}$$

Q learning Algorithm

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
input: ϵ, Number of episodes, Number of steps
initialization: $\forall s \in S, \forall a \in A$ initialise $Q(s, a) = 0$
for Number of episodes **do**
    Initialise S
    for Number of steps **do**
        Choose A from S using policy derived from Q, e.g ϵ- greedy
        Take action A, observe R, S'
        $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma Q(S', A) - Q(S, A)$
        $S \leftarrow S'$
return Q

4.1 Gridworld Q learning Toy example

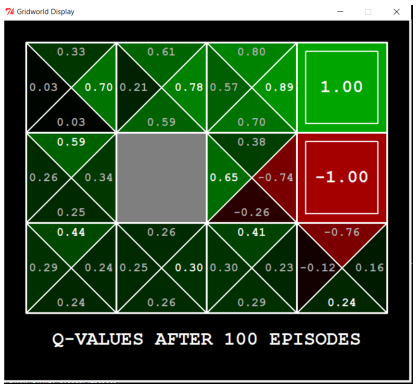

Figure 2: Q table after 100 iterations.

The figure below shows the gridworld environment. Each state is represented by the coordinates (x, y). For each state we can see that there are 4 actions, each with a different Q values. The figure shows that our implementation is correct since we passed all the tests of the autograd.py. Furthermore, the Q values doesn't change after increasing the number of episodes, which proves that the algorithm has converged.

4.2 ϵ -greedy

In Q learning, we have a balance to find: i.e exploration v/s exploitation. A well-known strategy for balancing exploration v/s exploitation is the ϵ - greedy method. In this method, the agent chooses random actions an epsilon fraction of the time, and follows its current best Q-values otherwise. This means that with a certain probability p, the agent will take a random action rather than the current optimal action. The higher the probability p, the higher will the exploration parameter. This is implemented in the question 5 of the material given to us.

4.3 Q learning on Pacman

Now that we have a working Q learning algorithm on a gridworld example, we are ready to test it on a Pacman environment. We first tested the Pacman agent on a small grid as shown in figure 3

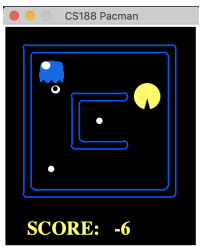

Figure 3: Pacman small grid

Several parameters were tested and we found the most effective to be the parameters as follows. $\epsilon = 0.05$, $\alpha = 0.2$, $\gamma = 0.8$.

The gameplay shows our agent playing on a small grid.

4.4 Observations

We played 2000 training games in total. As expected, when the number of training games exceeded 1500, our pacman agent was able to win almost all the games. We obtained an average rewards for last 100 episodes to be 256.84, which is quite high.

However, we tested the same Q learning agent on a medium grid. We run the 2000 training games as the small grid, but the average Rewards for last 100 episodes were -471.39, which is negative. This means that the the agent fails to win on larger layouts because each board configuration is a separate state with separate Q-values. He has no

way to generalize that running into a ghost is bad for all positions. Obviously, this approach will not scale. Hence, another approach was tested and its called Approximate Q learning.

5 Approximate Q learning vs tabular Q learning

In tabular Q learning, we are interested in finding a lookup table having specific Q values for each (state,action) pair. However, the limitation is that there might be similarities among the states, and the classical tabular Q learning has no way of distinguishing between similar states. This is when features come into play. A set of weighted features determine how good a particular state is. Then we describe the state using a vector of weighted features. For example the features could be the distance to the closest ghost or the number of ghosts present.

We can represent the values of states or the Q values as a set of weighted features. In this project we represented the Q function as a set of weighted features.

We use linear combination of features. Given by the equation below.

$$Q(s,a) = \sum_{n=1}^{n} f_i(s,a) w_i \tag{3}$$

We can't update the Q values as in equation (2). Instead we update a set of weights w_i. You will update your weight vectors similarly to how you updated Q-values:

$$w_i \leftarrow w_i + \alpha.difference - f_i(s,a) \tag{4}$$

where, $difference$ is given by equation (5).

$$difference = [R + \gamma Q(s', A)] \tag{5}$$

5.1 Pacman example

The approximate Q learning was applied to the Pacman game. However, this time, the huge advantage was that we could play on a bigger grid and we were not limited to the small one.

In our pacman example, the Q function was represented by the 2 following features.

- The number of ghosts who are 1 step away from the Pacman agent
- A feature related to the closest food pellet to the Pacman. The condition to execute this feature is that there should not be any ghost near the food pellet.

The gameplay(click here) shows the pacman agent playing. However, we notice that the agent is avoiding the ghosts even when the ghosts are scared. We would like the agent to eat the ghosts when they are scared.

5.2 With additional feature engineering

In this part, we presented our own feature. Our feature works as follows:
We only return the positions of the ghosts only if they are active. In case they are scared, the Pacman agent treats them as if they don't exist. This has the effect of eating the scared ghosts. The game play(click here) shows that the Pacman agent eats the scared ghosts while still managing to win the game with an average score of 1745.

6 Conclusion

In this project we over viewed the basics of reinforcement learning and the most popular RL algorithm, Q learning. We started with Value iteration algorithm and implemented the Q learning in a toy gridworld example. When we confirmed that our algorithm worked in the simple gridworld case, we applied it to the game of Pacman. However, with only Q learning we were limited to the small grid only. We scaled the game of Pacman to a larger grid by using approximate Q learning. The results show that we achieved an average score of 1745 on a large grid while obtaining the desired behaviour of the agent.