Reinforcement Learning

Priya Rao
Department of Computer Science
University at Buffalo
Buffalo, NY 14214
prao4@buffalo.edu

Abstract

This paper describes building a reinforcement learning agent to navigate the classic 4x4 grid-world environment. This agent learns an optimal policy through Q-Learning which will allow it to take actions to reach a goal while avoiding obstacles. The initial code template was already provided as part of the project requirement and the policy, update and the training algorithm was to be designed in order to successfully train this reinforcement learning agent.

1 Introduction

Reinforcement learning (RL) is basically where an agent learns by interacting with an environment. An RL agent learns from the consequences of its actions, rather than from being explicitly taught and it selects actions based not only on its past experiences (exploitation) but also by the new choices exposed by the environment (exploration), which is essentially trial and error learning. The reinforcement signal that the RL-agent receives is a numerical reward, which encodes the success of an action's outcome, and the agent seeks to learn to select actions that maximize the accumulated reward over time.

RL can be broadly broken down into three main sections:

1. The exploration-exploitation dilemma
2. Markov Decision Process
3. Q-Learning and policy learning

The next two subsections explain the first two sections of RL and Q-Learning and Policy Learning is described in detail in section 2.

1.1 The Exploration-Exploitation dilemma

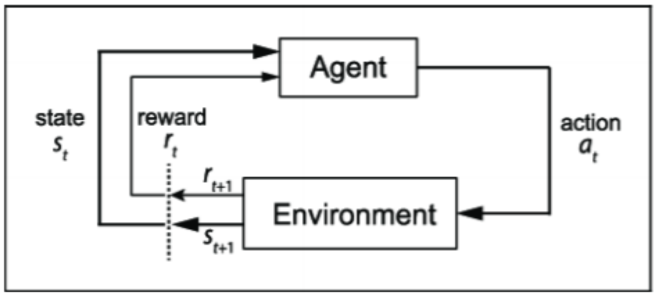
Exploration is the process by which the agent analyzes the tentative moves it can take while exploitation utilizes the best moves made so far. But the balance between the two determines how effectively the agent arrives at the best reward. One simple strategy for exploration would be for the agent to take the best-known action most of the time but occasionally explore a new, randomly selected direction even though it might be walking away from known reward.

This strategy is called the epsilon-greedy strategy, where epsilon is the percent of the time that the agent takes a randomly selected action rather than taking the action that is most likely to maximize reward given what it knows so far. We usually start with a lot of exploration (i.e. a higher value for epsilon). Over time, as the agent learns more about the environment and which actions yield the most long-term reward, it would make sense to steadily reduce epsilon to 10% or even lower as it settles into exploiting what it knows.

1.2 Markov Decision Process

The agent's wandering through the grid can be formalized as a **Markov Decision Process**, which is a process that has specified transition probabilities from state to state. MDPs include:

1. **A finite set of states.** These are the possible positions of the agent within the grid.
2. **A set of actions available in each state.** This is set of actions (up, down, left, right) that can be taken up by the agent
3. **Transitions between states.** For example, if the agent goes down in the grid it ends up in a new position. These can be a set of probabilities that link to more than one possible state.
4. **Rewards associated with each transition.** The reward to moving towards the goal is +1 and the reward to moving away from the goal is -1.
5. **A discount factor γ between 0 and 1.** This quantifies the difference in importance between immediate rewards and future rewards.



2 Q-Learning

Q-learning is the reinforcement learning algorithm most widely used for addressing the control problem because of its off-policy update, which makes convergence control easier. The Q-learning algorithm essentially evaluates which action to take based on the action-value function that determines the value of being in a certain state and taking a certain action at that state. The goal of Q-learning is to learn a policy, which tells an agent what action to take under what circumstances. It does not require a model (hence the connotation "model-free") of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.

To determine whether the agent will choose exploration or exploitation at each step, we generate a random number between 0 and 1. If this number is greater than epsilon, then the agent will choose its next action via exploitation, i.e. it will choose the action with the highest Q-value for its current state from the Q-table. Otherwise, its next action will be chosen via exploration. The action will be chosen randomly since our exploration rate is set to 1 initially. So, after the agent takes an action, it observes the next state, the reward gained from its action, and updates the Q-value in the Q-table for the action it took from the previous state.

Thus, the Q-learning algorithm iteratively updates the Q-values for each state-action pair using the Bellman equation until the Q-function converges to the optimal Q-function. This approach is called value iteration.

$$q_*(s,a) = E\left[R_{t+1} + \gamma \max_{a'} q_*(s',a')\right]$$

We want to make the Q-value for the given state-action pair as close as we can to the right-hand side of the Bellman equation so that the Q-value will eventually converge to the optimal Q-value q∗.

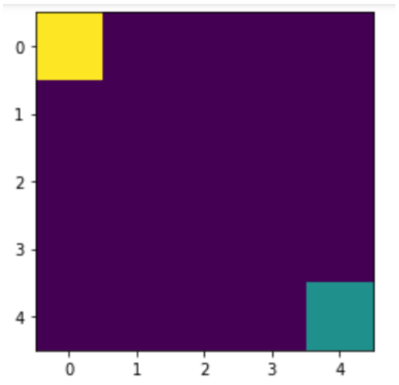
This will happen over time by iteratively comparing the loss between the Q-value and the optimal Q-value for the given state-action pair and then updating the Q-value over and over again each time we encounter this same state-action pair to reduce the loss.

where the reward is received upon moving from one state to the next and the learning rate is a value between 0 and 1. An episode of this algorithm ends when the final state is the next state. The learning rate is a number between 0 and 1, which can be thought of as how quickly the agent abandons the previous Q-value in the Q-table for a given state-action pair for the new Q-value.

$$q^{new}(s,a) = (1-\alpha)\ \underbrace{q(s,a)}_{\text{old value}}\ + \alpha \overbrace{\left(R_{t+1} + \gamma \max_{a'} q(s',a')\right)}^{\text{learned value}}$$

3 Environment

The environment provided is a basic deterministic n × n grid-world environment (the initial state is shown below) where the agent (shown as the green square) has to reach the goal (shown as the yellow square) in the least amount of time steps possible. The agent is the yellow square which can perform actions at each state in order to earn rewards. Rewards are positive or negative weights that are assigned when the agent moves closer to the goal or away from the goal respectively.



4 Model/Training

The Q-Learning algorithm basically can be broken down into the following steps:

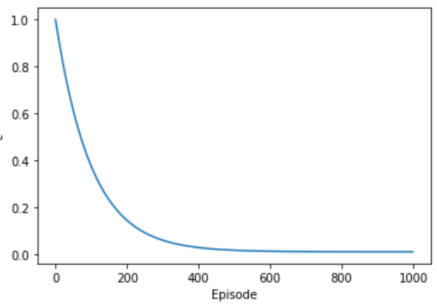
1. For each step in each episode:
   1.1 Choose an action by taking into considering the exploration-exploitation tradeoff as being done in the policy method
   1.2 Observe the reward and record the next state
   1.3 Update the Q-value function using the formula described under the section of Q-Learning. This, over time, will make the Q-value function converge to the right-hand

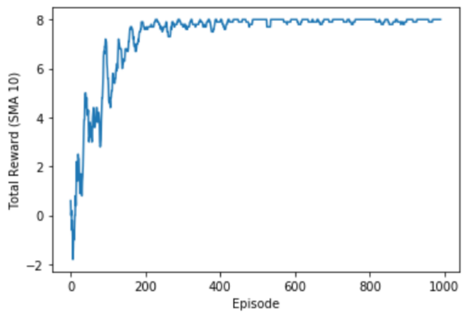
side of the Bellman equation given above

We define the maximum exploration rate and the minimum exploration rate as bounds to how large or small our exploration rate can be. Also, the exploration rate is represented with the symbol ϵ. Lastly, we have the exploration decay rate to determine the rate at which the exploration rate will decay.

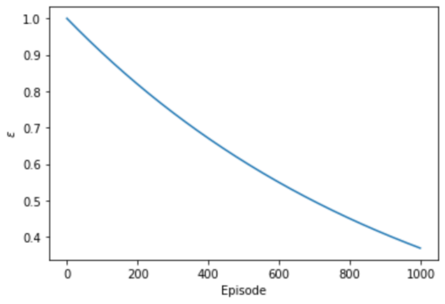
The model was trained by tuning the minimum exploration rate, episodes and the exploration decay rate. The episodes were fixed at 1000 and by keeping the learning rate as the default given rate in the code template and only fine tuning the other two parameters, there were 2 scenarios encountered:

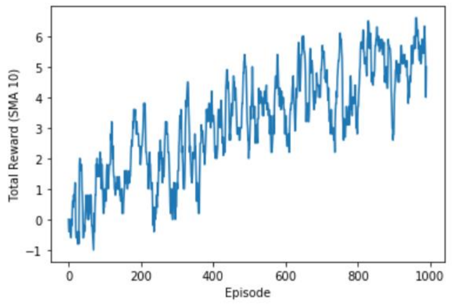
1. The below graphs were obtained when the minimum exploration rate as well as the exploration decay was set to 0.01. This shows that we should've stopped updating epsilon by the 500th episode since it essentially means that the agent is done exploring.





2. The below graphs were obtained when the minimum exploration rate as well as the exploration decay was set to 0.001. From this it's evident that the exploration was not yet completed as the local minima was still not found (epsilon has not converged) which is why the total rewards graph depicts that the agent is still exploring. Even after increasing the learning rate from 0.5 to 0.9, the convergence is not reached.





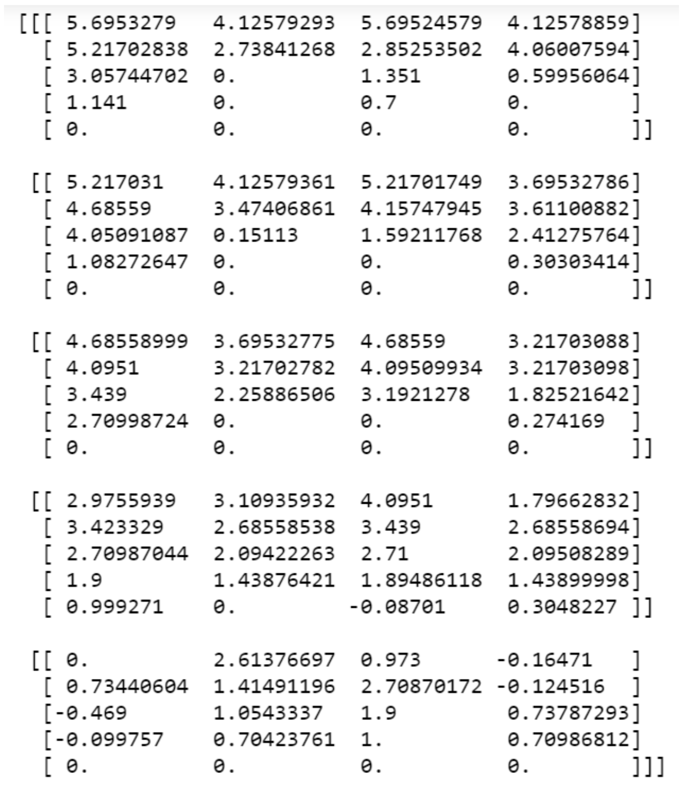
5 Challenges

One of the challenges that were faced was that since the number of episodes were given in the problem statement as at 1000 episodes, the hyper parameters were tuned to fit within this window. This could've been avoided by decreasing the number of episodes and increasing the hyper parameters of exploration decay rate and learning rate in order to achieve the goal quicker.

6 Results

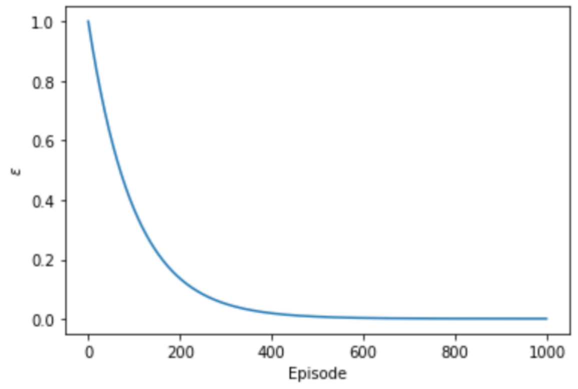
The results of the experiment were drawn based on tuning of the hyperparameters. The epsilon versus number of episodes was plotted and the total rewards versus episodes graph is also plotted as part of training the agent. *The optimal result was obtained by keeping the episodes as 1000 itself, increasing the learning rate to 0.7, keeping the discount rate as the default of 0.9 and having the* *minimum exploration rate as 0.001, the exploration decay rate 0.01.*

The q-table after training this agent generated was calculated and is displayed below:

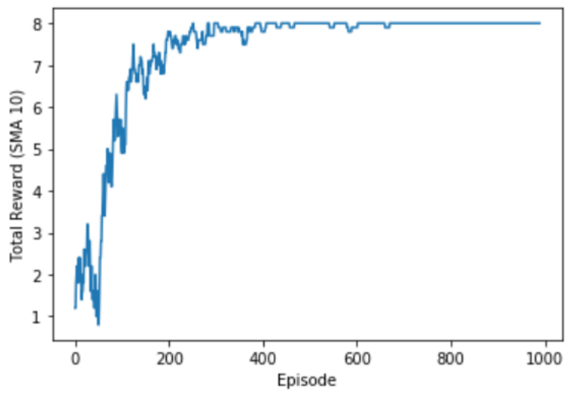


```
[[[ 5.6953279   4.12579293  5.69524579  4.12578859]
  [ 5.21702838  2.73841268  2.85253502  4.06007594]
  [ 3.05744702  0.          1.351       0.59956064]
  [ 1.141       0.          0.7         0.        ]
  [ 0.          0.          0.          0.        ]]

 [[ 5.217031    4.12579361  5.21701749  3.69532786]
  [ 4.68559     3.47406861  4.15747945  3.61100882]
  [ 4.05091087  0.15113     1.59211768  2.41275764]
  [ 1.08272647  0.          0.          0.30303414]
  [ 0.          0.          0.          0.        ]]

 [[ 4.68558999  3.69532775  4.68559     3.21703088]
  [ 4.0951      3.21702782  4.09509934  3.21703098]
  [ 3.439       2.25886506  3.1921278   1.82521642]
  [ 2.70998724  0.          0.          0.274169  ]
  [ 0.          0.          0.          0.        ]]

 [[ 2.9755939   3.10935932  4.0951      1.79662832]
  [ 3.423329    2.68558538  3.439       2.68558694]
  [ 2.70987044  2.09422263  2.71        2.09508289]
  [ 1.9         1.43876421  1.89486118  1.43899998]
  [ 0.999271    0.         -0.08701     0.3048227 ]]

 [[ 0.          2.61376697  0.973      -0.16471   ]
  [ 0.73440604  1.41491196  2.70870172 -0.124516  ]
  [-0.469       1.0543337   1.9         0.73787293]
  [-0.099757    0.70423761  1.          0.70986812]
  [ 0.          0.          0.          0.        ]]]
```

The epsilon versus episode graph was generated as follows:



The total rewards versus episodes graph was generated as follows:



Printing the average rewards per 100 episodes we get:

```
********Average reward per hundred episodes********

100 :  2.49
200 :  6.540000000000002
300 :  7.580000000000005
400 :  7.820000000000006
500 :  7.960000000000005
600 :  7.980000000000006
700 :  7.970000000000006
800 :  8.000000000000005
900 :  8.000000000000005
1000 :  8.000000000000005
```

7 Conclusion

From the above results we can conclude that for the 1000 episodes given, the hyperparameters are tuned (learning rate of 0.7, decay rate of 0.01, minimum exploration rate of 0.001 and discount rate of 0.9) in such a way that we have the goal reached with the maximum total reward and by balancing the exploration and exploitation trend.

References

[1] Medium – Reinforcement Learning https://medium.com/machine-learning-for-humans/reinforcement-learning-6eacf258b265

[2] Christopher M. Bishop ©2006 Pattern Recognition and Machine Learning. Springer-Verlag Berlin, Heidelberg

[3] StackExchange - https://stats.stackexchange.com/

[4] Kaggle - https://www.kaggle.com/

[5] Scholarpedia – Reinforcement Learning http://www.scholarpedia.org/article/Reinforcement_learning

[6] Wikipedia – Q-Learning https://en.wikipedia.org/wiki/Q-learning

[7] DeepLizard - https://deeplizard.com/learn/