Project 4: Tom and Jerry in Reinforcement learning

Arinjoy Bhattacharya
Department of Physics
University at Buffalo, SUNY
Buffalo, NY 14214
Person # 50168806
arinjoyb@buffalo.edu

Abstract

A simple Tom and Jerry game was implemented using Reinforcement learning and Deep Q-Learning to iteratively perfect out the rewards.

1 Introduction

Training a computer to beat humans in a game has been a fascination since the onset of machine learning. Through reinforcement learning, a model can be created which can be trained with the help of multiple experiences which can been used to perfect its own rewards stochastically.

2 Model

2.1 Model: Environment description

The environment is a grid world with an agent and a goal. All possible movements are allowed and the agent gets a reward of +1 if it moves closer to the target using the shortest path. Each episode consists of 100 steps and the agent tries to achieve as much higher of a reward as possible. After each episode it is then reset and thus the agent learns an optimal path after a certain number of episodes.

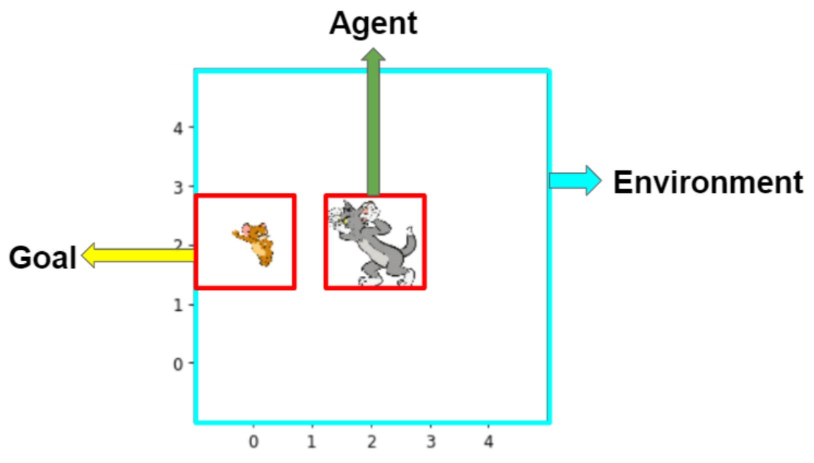


Figure 1: Environment for Deep Q learning.

2.2 Model: Reinforcement Learning

It is a method in Machine Learning where an agent learns how to behave in an environment by performing actions and seeing the results. In reinforcement learning, an agent learns from trial-and-error feedback rewards from its environment, and results in a policy that maps states to actions to maximize the long-term total reward as a delayed supervision signal. Reinforcement learning combining with the neural networks has made great progress recently, including playing Atari games and beating world champions at the game of Go.

Important parameters in reinforcement learning include:

- S is a finite set of states where the agent can move around.
- A is a finite set of actions which the agent can take, like UP, DOWN, LEFT, RIGHT.
- P_a (s; s_0) = P_r (s_{t+1} = s_0 | s_t = s; a_t = a) is the probability that action 'a' in state 's' at time t will lead to state s_0 at time t + 1.
- R_a (s; s_0) is the immediate reward (or expected immediate reward) received after transitioning from state 's' to state 's_0' due to action 'a'.
- $\gamma \in [0;1)$ is the discount factor, which represents the difference in importance between future rewards and present rewards.

2.2.1 Model: Reinforcement Learning: Discounting factor (γ)

It is used to penalize the rewards in the future. Reward at time k worth only γ^{k-1}. It is good for the system since it helps in more learning since the future rewards might have more uncertainty. Also it provides mathematical convenience by not providing immediate benefits. It is used later on in the project for Q-value calculation in Deep-Q Learning.

2.3 Model: Deep Q-Learning

It implements experience replay to randomly train the network based on its previous learning experience. This in turn increases speed of learning and also avoid forgetting between experiences. Calculating the Q-value is shown below:

$$Q_t = \begin{cases} r_t, & if\ episode\ terminates\ at\ step\ t+1 \\ r_t + \gamma\ max_a Q(s_t, a_t; \Theta) & otherwise \end{cases}$$

where γ is the discount factor, r_t is the reward and Qt is the Q-value.

2.4 Model: Exploration vs Exploitation

The agent selects an action at random and implements it for the reward. Once it has the reward it will use the exploration rate or "epsilon" (ϵ) to process the next reward.

The exponential decay of ϵ is given by

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda |s|}$$

2.5 Model: Neural Networks

Neural networks are trained iteratively using optimization techniques like gradient descent. After each cycle of training, an error metric is calculated based on the difference between prediction and target. The derivatives of this error metric are calculated and propagated back. They are then adjusted relative to how much they contributed to the total error. This process is repeated iteratively until the network error drops below an acceptable threshold.

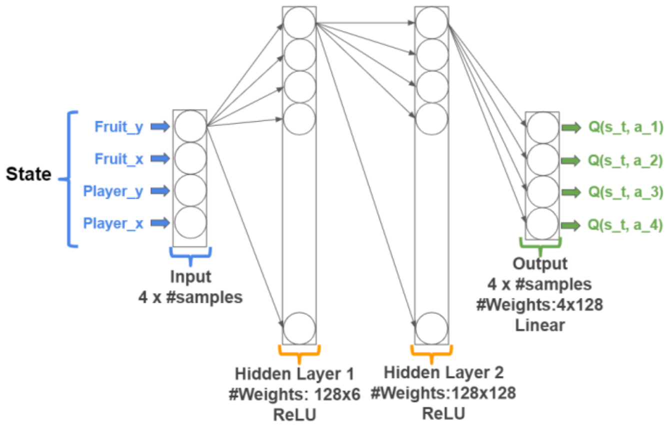


Figure 2: Neural network implemented in the provided Deep Q learning code.

2.5.1 Model: Neurons

The inputs to each node are combined using a weighted linear combination. The result is then modified by a nonlinear function before being output. Mathematically, it can be shown by the equation.

$$z_j = b_j + \sum_{i=1}^{n} w_{i,j} x_i$$

, where $w_{i,j}$ represents the weight vector, x_i represent the input vector from each neuron and z_j gives the values in the hidden layer.

2.5.2 Model: Layers

There are mostly speaking three layers in the network, the input layer, the hidden layer and the output layer. The provided code has two hidden layers, so essentially speaking there are total of four layers. In the hidden layer, this is then modified using a nonlinear function such as a relu,

$$f(x) = x^+ = \max(0, x)$$

to give the input for the next layer. This tends to reduce the effect of extreme input values, thus making the network somewhat potent to arbitrary values. The parameters $b_1, b_2, b_3, w_{1,2}$ ... are "learned" by training the data. The values of the weights are often confined to prevent them from blowing up. The parameter that restricts the weights in the code is known as the 'learning rate', which varies from 0.1 to 1.

2.5.3 Model: Weights

The weights take arbitrary values to commence with, and these are then updated utilizing the observed data. Thereupon, there's a part of unpredictability within the predictions made by a neural network. Therefore, the network is typically trained many times considering totally different random beginning points, and also the results area unit averaged. The number of hidden layers, and the number of neurons in each hidden layer, must be specified in advance.

2.5.4 Model: Activation functions

Activation functions takes the input from the hidden layer and modify the values to tune the model. Activation functions give neural networks to model complex non-linear relationships. By modifying inputs with non-linear functions neural networks can model highly complex relationships between features. Popular activation functions include relu and leaky relu. The model's structure is: LINEAR → RELU → LINEAR → RELU → LINEAR.

2.5.5 Model: Loss functions

After the hidden layer, a loss function is implemented to check how well the model performs for a particular set of parameters. In the provided code, it is a mean square error function whose slope gives us how the model actually is performing. So, depending on the performance of the model, the parameters are modified by the loss function.

2.5.6 Model: Optimization Methods

Optimization algorithms are used to back process the model and reduce the error in accuracy rate of the model. The method used in the provided code is RMSProp with a learning rate of 0.00025.

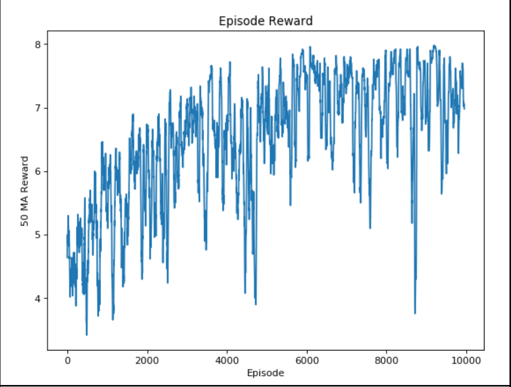
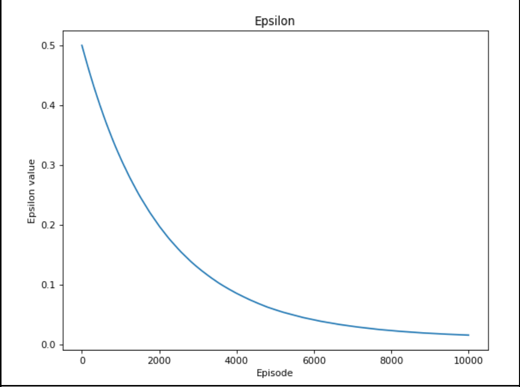
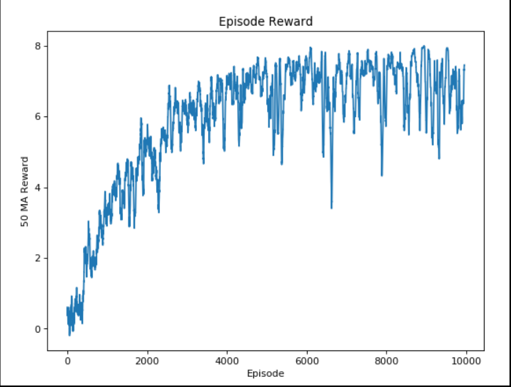
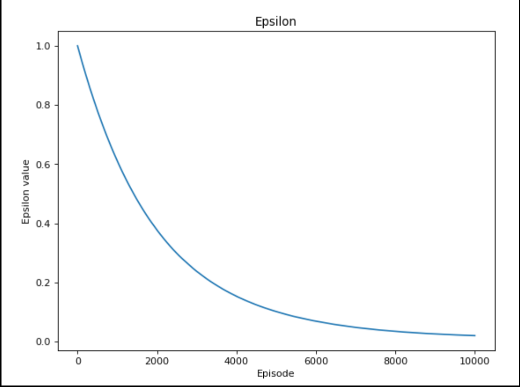
2.5.7 Model: Forward and Backward Propagation

Essentially speaking, the two methods of propagation are complimentary to one another. The forward propagation involves calculating weights and multiplying them with the input, and finally using the activation function to pass it to the output layer. In case of backpropagation each weight in the network are adjusted in proportion to how much it contributes to overall error. The error function if differentiated with respect to particular hyperparameters and equated to zero to get our best probable accuracy.

3 Experiment

There are a couple of parameters which can be tuned in the agent environment which includes the maximum and minimum values of epsilon or what is known as the "exploration rate", discounting parameter or what is termed here as 'γ' and the number of episodes used in training.

3.1 Experiment: Changing the maximum value of epsilon (ϵ_{max}).

a)		
b)		

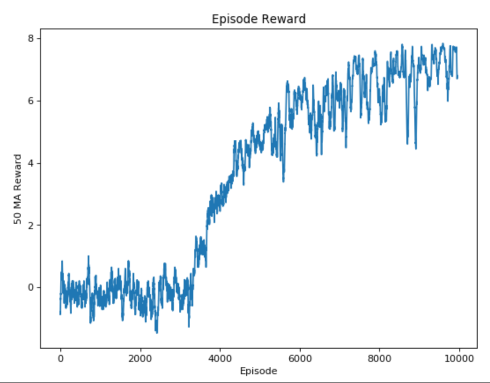
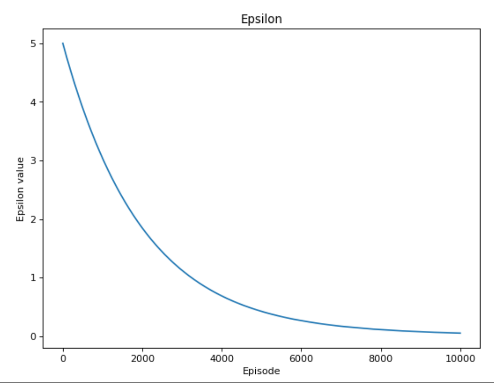
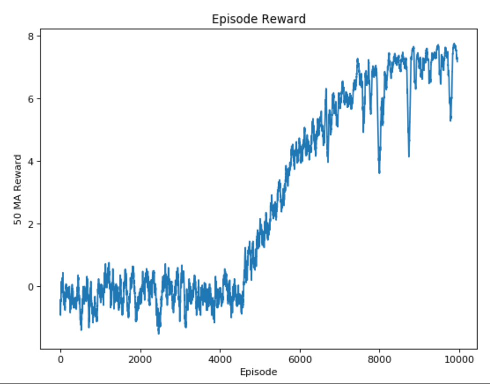
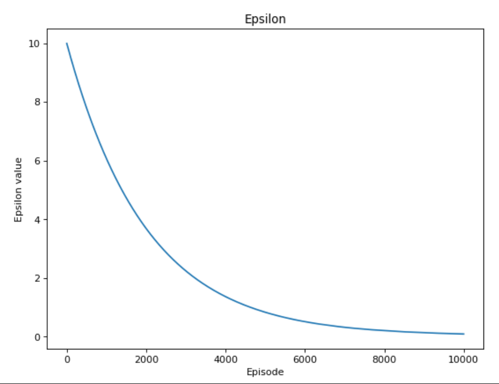
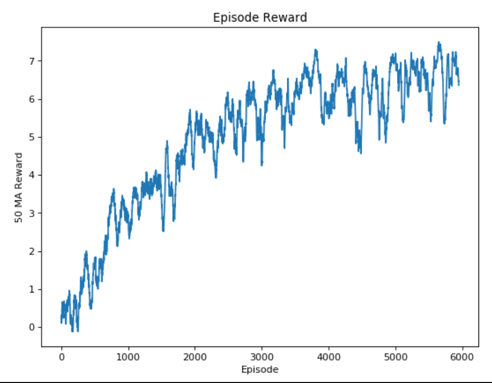
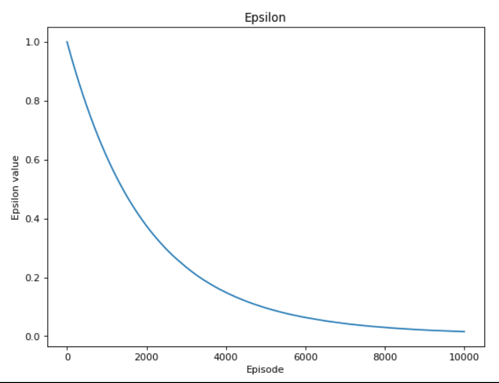
c)		
d)		

Figure 3: Rewards and Epsilon value for four different cases: a) $\epsilon_{max} = 0.5$, b) $\epsilon_{max} = 1$, c) $\epsilon_{max} = 5$, d) $\epsilon_{max} = 10$. Remaining parameters were kept constant at $\epsilon_{min} = 0.01$, episodes = 10000 and $\gamma = 0.99$.

From the plots it is easily understood that with more range of epsilon, meaning higher maximum values of epsilon gives more stable straining ensuring the mean rewards are more.

3.2 Experiment: Changing the minimum value of epsilon (ϵ_{min}).

a)		

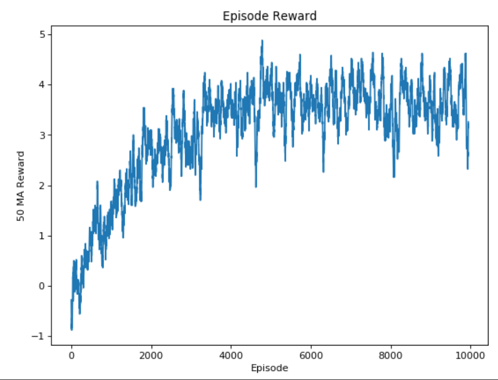
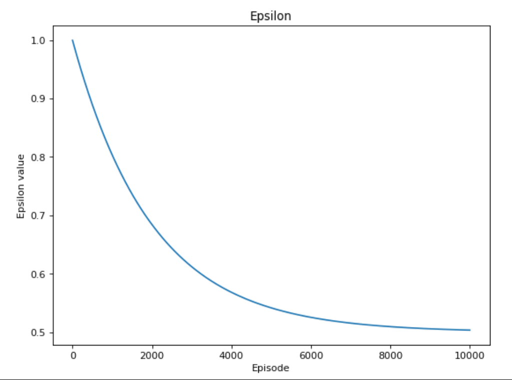
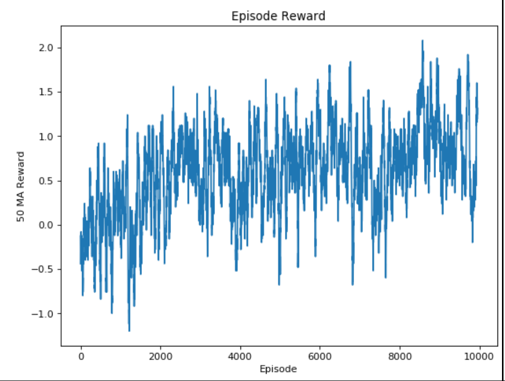
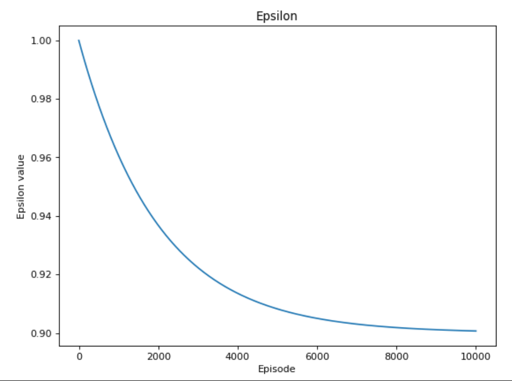
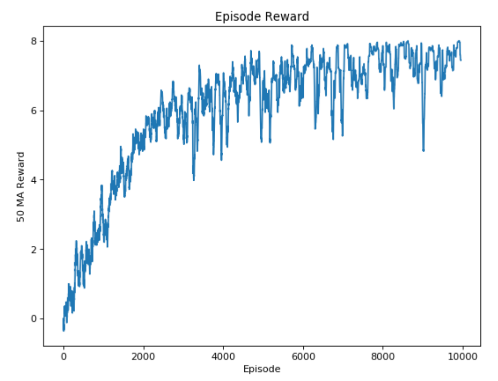
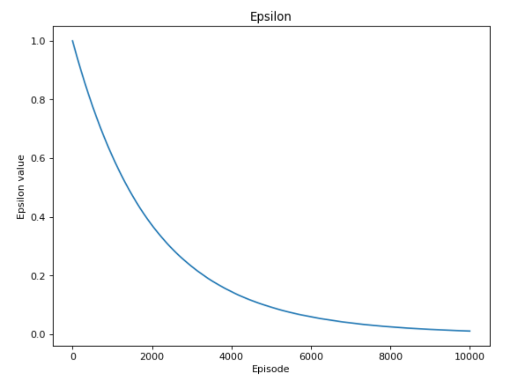
b)		
c)		
d)		

Figure 4: Rewards and Epsilon value for four different cases: a) $\epsilon_{min} = 0.005$, b) $\epsilon_{min} = 0.5$, c) $\epsilon_{min} = 0.9$, d) $\epsilon_{min} = 0$. Remaining parameters were kept constant at $\epsilon_{max} = 1$, episodes = 10000 and $\gamma = 0.99$.

From the plots it is easily understood that with more range of epsilon, meaning higher maximum values of epsilon gives more stable straining ensuring the mean rewards are more. Too high minimum value gives a noisy rewards curve, since there is too less range of epsilon.

3.3 Experiment: Changing the discounting parameter (γ).

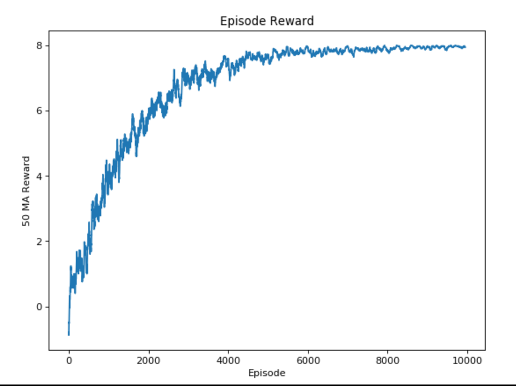
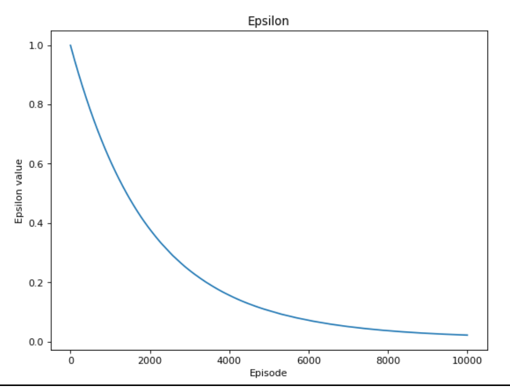
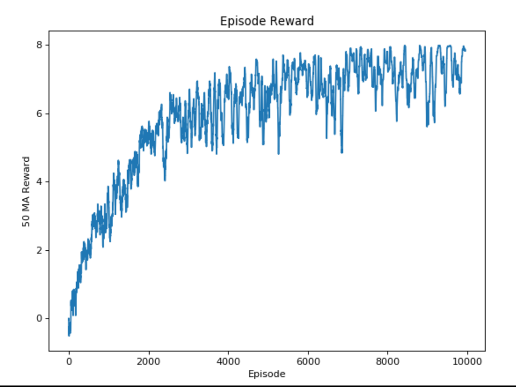
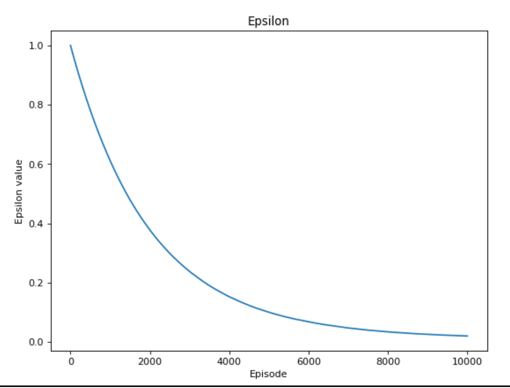
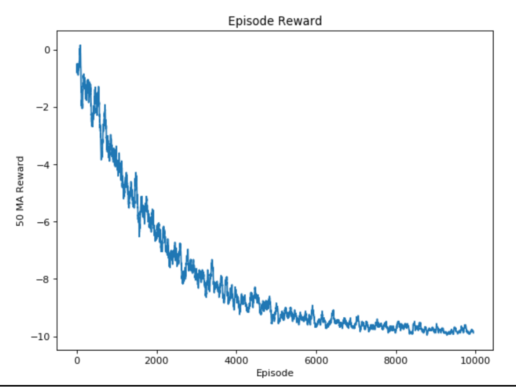
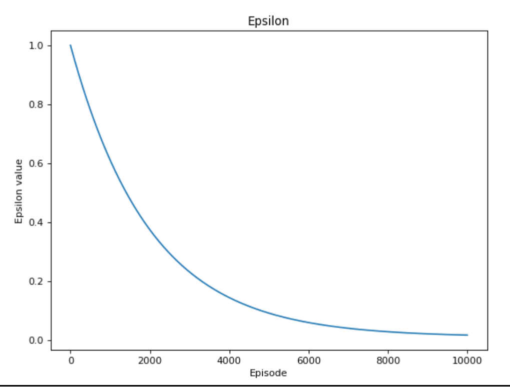
a)		
b)		
c)		

Figure 5: Rewards and Epsilon value for three different cases: a) $\gamma = 0.1$, b) $\gamma = 0.99$, c) $\gamma = 5$. Remaining parameters were kept constant at $\epsilon_{min} = 0.01$, episodes = 10000 and $\epsilon_{max} = 1$.

From the plots it is easily understood that γ should always be less than 1. When the value of γ is more than 1 then the rewards are seen to decay down rather than increase meaning the agent seems to be moving further away rather than coming nearby. Also, lower values of γ seems better since the rewards plot is less noisy and we have a clean value of reward.

3.4 Experiment: Changing the number of episodes.

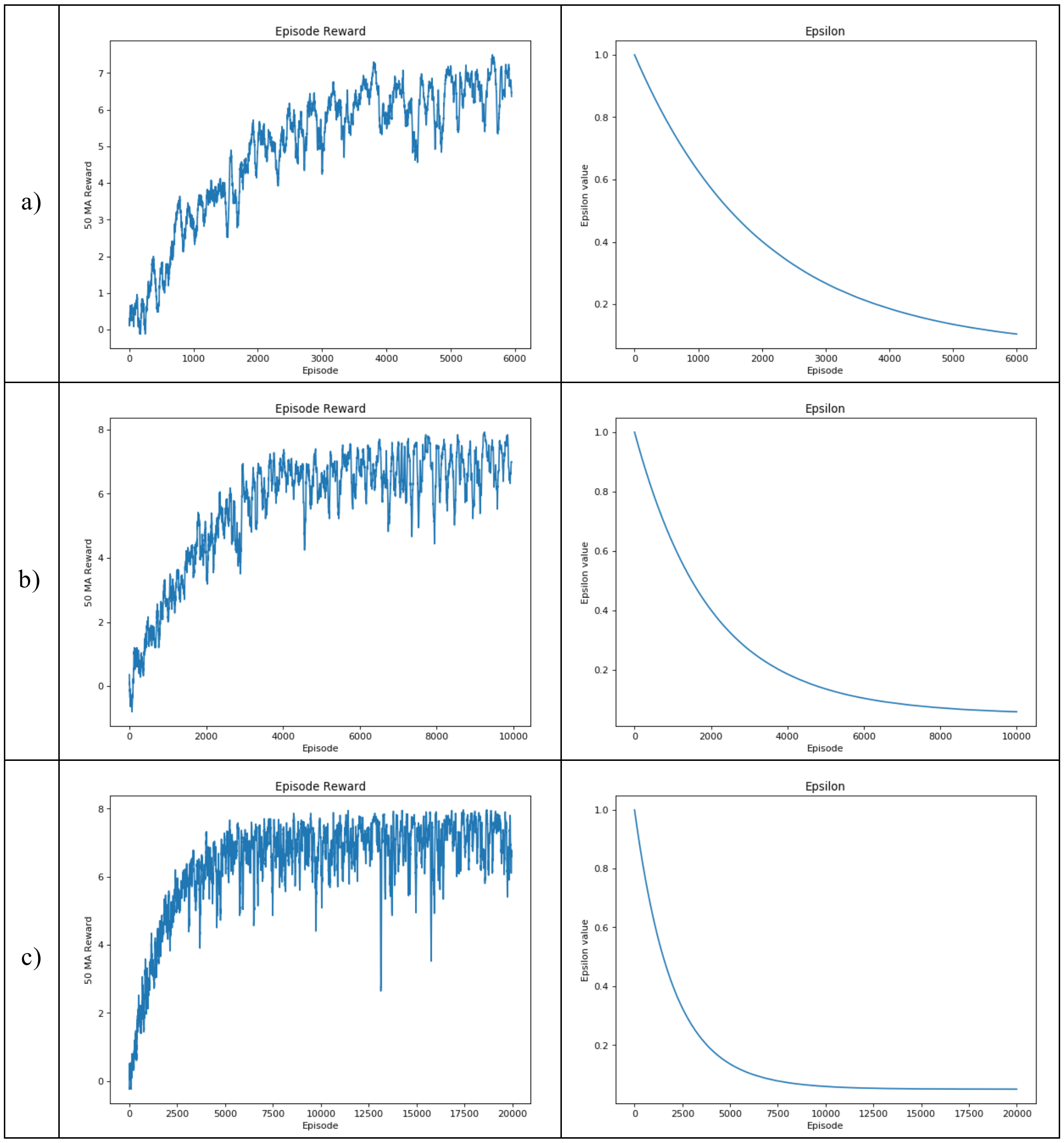


Figure 6: Rewards and Epsilon value for four different cases: a) episodes = 6000, b) episodes = 10000, c) episodes = 20000. Remaining parameters were kept constant at ϵ_{min} = 0.01, γ = 0.99 and ϵ_{max} = 1.

The number of episodes enhances the rewards directly. More is the number of episodes ensures longer and better training and hence a better rewards curve.

3.5 Experiment: the perfect parameter

For the best reward curve the parameters to be put are:

- Number of episodes = 20000 or higher.
- 2 or more number of layers in the neural network.
- ϵ_{min} = 0.001 or lesser.
- ϵ_{max} = 1 or more.
- γ = 0.1 or higher.

4 Writing Task

4.1 Question number 1:

If the agent always chooses the way of maximizing Q-value, then it chooses such states which moves farther away from the target. It will tend to get stuck around the corners of the environment grid which is provided and will return lower rewards. Using the example in question number 2, it can be easily shown that the maximum Q-value states are the farthest away from the target.

To force the agent to explore more, one can implement the epsilon greedy algorithm. In this case, with each iteration, with a probability of epsilon, the agent takes a random action as opposed to action with the highest quality.

- An exploration rate "epsilon," is set to 1 in the beginning. In the beginning a lot of exploration is done since nothing is known about Q-values.
- Then a number is taken random and checked if it is greater than epsilon, then it is switched to "exploitation" (selecting the best action is each step). Else, exploration is continued.
- The idea is to bring down the value of epsilon when the agent becomes confident about estimating Q-values.

Also, Boltzmann policy can be a very important way to do exploration by stochasticity. It uses the Q-values actually to assign different probabilities to each action using a softmax function. There is also a parameter β apart from the Q values, which is called temperature to control the dependence.

4.2 Question number 2:

It is considered that all the states in the deterministic environment (the provided grid) to be labelled as follows.

S_{11}	S_{12}	S_{13}
S_{21}	S_{22}	S_{23}
S_{31}	S_{32}	S_{33}

Also, the provided formula for Q-values calculation is:

$$Q(s_t, a_t) = r_t + \gamma * max_a Q(s_{t+1}, a)$$

and $\gamma = 0.99$ and the target is S_{33}.
For each state there are a total number of four actions: UP, DOWN, LEFT, RIGHT which are possible and thus returns 4 different Q values.

For obvious reasons we can clearly initialize the Q-values for our target to be 0.

Thus, for **S_{33}:**
$q\ (s_{33}, up) = q\ (s_{33}, down) = q\ (s_{33}, left) = q\ (s_{33}, right) = 0$

For **S_{32}:**
$q\ (s_{32}, up) = r_t + \gamma * \max_a (q\ (s_{22}, a))$
$= -1 + 0.99*(\max_a (q\ (s_{22}, a)))$

Now, for $\max_a (q\ (s_{22}, a))$, the agent can be either moved DOWN then RIGHT or RIGHT then DOWN in either of which cases the Q-value is the same.

$\max_a (q\ (s_{22}, a)) = 1 + (0.99*1) = 1.99$

So, $q\ (s_{32}, up) = -1 + (0.99*1.99) = 0.9701$

Now for $q\ (s_{32}, left)$ it's the same number of steps to the target, it's just in this case the agent goes RIGHT twice for the target.

So, $q\ (s_{32}, left) = q\ (s_{32}, up) = 0.9701$

In case of $q\ (s_{32}, right)$, it enters the target directly so, it's = 1 + (0.99*0) = 1

For the case of $q\ (s_{32}, down)$ it just stays in its actual position getting no reward after the first action, so it's = 0 + (0.99*1) = 0.99

Similarly, all the other actions and states are enunciated below with associated calculations and substitutions.

For **S_{31}:**
$q\ (s_{31}, up) = -1 + 0.99*(\max_a (q\ (s_{21}, a))) = -1 + 0.99*(1 + 0.99 + 0.99^2) = 1.9403$
$q\ (s_{31}, right) = 1 + 0.99*(\max_a (q\ (s_{32}, a))) = 1 + 0.99*(1) = 1.99$
$q\ (s_{31}, left) = 0 + 0.99*(\max_a (q\ (s_{31}, a))) = 0 + 0.99*(1 + 0.99) = 1.9701$
$q\ (s_{31}, down) = 0 + 0.99*(\max_a (q\ (s_{31}, a))) = 0 + 0.99*(1 + 0.99) = 1.9701$

For **S_{23}:**
$q\ (s_{23}, up) = -1 + 0.99*(\max_a (q\ (s_{13}, a))) = -1 + 0.99*(1 + 0.99) = 0.9701$
$q\ (s_{23}, right) = 0 + 0.99*(\max_a (q\ (s_{23}, a))) = 0 + 0.99*(1) = 0.99$
$q\ (s_{23}, left) = -1 + 0.99*(\max_a (q\ (s_{13}, a))) = -1 + 0.99*(1 + 0.99) = 0.9701$
$q\ (s_{23}, down) = 1 + 0.99*(\max_a (q\ (s_{33}, a))) = 1 + 0.99*(0) = 1$

For **S_{22}:**
$q\ (s_{22}, up) = -1 + 0.99*(\max_a (q\ (s_{12}, a))) = -1 + 0.99*(1 + 0.99 + 0.99^2) = 1.9403$
$q\ (s_{22}, right) = 1 + 0.99*(\max_a (q\ (s_{23}, a))) = 1 + 0.99*(1) = 1.99$
$q\ (s_{22}, left) = -1 + 0.99*(\max_a (q\ (s_{21}, a))) = -1 + 0.99*(1 + 0.99 + 0.99^2) = 1.9403$
$q\ (s_{22}, down) = 1 + 0.99*(\max_a (q\ (s_{32}, a))) = 1 + 0.99*(1) = 1.99$

For **S_{21}:**
$q\ (s_{21}, up) = -1 + 0.99*(\max_a (q\ (s_{11}, a))) = -1 + 0.99*(1 + 0.99 + 0.99^2 + 0.99^3) = 2.9009$
$q\ (s_{21}, right) = 1 + 0.99*(\max_a (q\ (s_{22}, a))) = 1 + 0.99*(1 +0.99) = 2.9701$
$q\ (s_{21}, left) = 0 + 0.99*(\max_a (q\ (s_{21}, a))) = 0 + 0.99*(1 + 0.99 + 0.99^2) = 2.9403$
$q\ (s_{21}, down) = 1 + 0.99*(\max_a (q\ (s_{31}, a))) = 1 + 0.99*(1 + 0.99) = 2.9701$

For **S_{13}:**
$q\ (s_{13}, up) = 0 + 0.99*(\max_a (q\ (s_{13}, a))) = 0 + 0.99*(1 + 0.99) = 1.9701$
$q\ (s_{13}, right) = 0 + 0.99*(\max_a (q\ (s_{13}, a))) = 0 + 0.99*(1 + 0.99) = 1.9701$
$q\ (s_{13}, left) = -1 + 0.99*(\max_a (q\ (s_{12}, a))) = -1 + 0.99*(1 + 0.99 + 0.99^2) = 1.9403$
$q\ (s_{13}, down) = 1 + 0.99*(\max_a (q\ (s_{23}, a))) = 1 + 0.99*(1) = 1.99$

For **S_{12}:**
$q\ (s_{12}, up) = 0 + 0.99*(\max_a (q\ (s_{12}, a))) = 0 + 0.99*(1 + 0.99 + 0.99^2) = 2.9403$
$q\ (s_{12}, right) = 1 + 0.99*(\max_a (q\ (s_{13}, a))) = 1 + 0.99*(1 + 0.99) = 2.9701$
$q\ (s_{12}, left) = -1 + 0.99*(\max_a (q\ (s_{11}, a))) = -1 + 0.99*(1 + 0.99 + 0.99^2 + 0.99^3) = 2.9009$
$q\ (s_{12}, down) = 1 + 0.99*(\max_a (q\ (s_{22}, a))) = 1 + 0.99*(1 + 0.99) = 2.9701$

For **S_{11}:**
$q\ (s_{11}, up) = 0 + 0.99*(\max_a (q\ (s_{11}, a))) = 0 + 0.99*(1 + 0.99 + 0.99^2 + 0.99^3) = 3.9009$
$q\ (s_{11}, right) = 1 + 0.99*(\max_a (q\ (s_{12}, a))) = 1 + 0.99*(1 + 0.99 + 0.99^2) = 3.9403$

$q\ (s_{11},\ left) = 0 + 0.99*(\max_a\ (q\ (s_{11},\ a))) = 0 + 0.99*(1 + 0.99 + 0.99^2 + 0.99^3) = 3.9009$
$q\ (s_{11},\ down) = 1 + 0.99*(\max_a\ (q\ (s_{21},\ a))) = 1 + 0.99*(1 + 0.99 + 0.99^2) = 3.9403$

Putting them all in a Q-value value returns the table as shown below

	UP	DOWN	LEFT	RIGHT
S_{11}	3.9009	3.9403	3.9009	3.9403
S_{12}	2.9403	2.9701	2.9009	2.9701
S_{13}	1.9701	1.99	1.9403	1.9701
S_{21}	2.9009	2.9701	2.9403	2.9701
S_{22}	1.9404	1.99	1.9404	1.99
S_{23}	0.9701	1	0.9701	0.99
S_{31}	1.9403	1.9701	1.9701	1.99
S_{32}	0.9701	0.99	0.9701	1
S_{33}	0	0	0	0

Table 1: Q-values for all different combinations of actions and states.

From the table above we only need the Q-values for a few number of states which are referred as s_0, s_1, ... s_4.

	UP	DOWN	LEFT	RIGHT
S_0	3.9009	3.9403	3.9009	3.9403
S_1	2.9403	2.9701	2.9009	2.9701
S_2	1.9404	1.99	1.9404	1.99
S_3	0.9701	1	0.9701	0.99
S_4	0	0	0	0

Table 2: Q-values for the particular actions and states given in the question.

5 Conclusion

Reinforcement learning can be tuned by using the above said parameters and using Deep Q-learning can be made better by calculating the Q-values of the agent one can find most favorable actions.

Acknowledgments

I thank the professor Dr. Srihari and the TAs for helping me complete this report.

References

[1] CM Bishop, TM Mitchell. (2014) *Pattern Recognition and Machine Learning*, Springer.

[2] Richard S. Sutton, Andrew G. Barto. (1998) *Reinforcement Learning: An Introduction*, MIT Press.