PROJECT 4: TOM AND JERRY IN REINFORCEMENT LEARNING

Monisha Balaji
University at Buffalo
mbalaji@buffalo.edu

(This report includes the writing task for this project)

Abstract

Machine learning and AI have been changing the course of technological advancement for over a few decades now. It has led to the birth of many new techniques and the evolution of many others. The application of ML algorithms is growing exponentially on a daily basis. The classic problem categories that every ML and AI freshman has come across are the regression and classification problems. Regression problems, as we know, deal with real values and aim to produce a continuous output. Classification, on the other hand, focuses on predicting the which output class the test data belongs to. But there is so much more to be dealt with for the execution of more complex tasks. In this project, we aim to understand the working of Reinforcement algorithm and apply the mechanism to help the agent (Tom) achieve the goal (Jerry). This also involves the creation of a Deep Q-Network. This report records the implementation procedures of the game environment and the agent and also performance of the agent over multiple episodes.

1. INTRODUCTION

Reinforcement Learning is an area of Machine Learning that focuses on the agent, in an environment, that has to perform a set of tasks to achieve a goal. The agent, the environment and the goal are subjective; it depends on the application the developer is working on. The focus is to perform actions that aim to maximize the total reward. RI has proven to be effective in many other fields such as Operations Research, Optimal Control, Reward System and many more. This algorithm is more like trial and error method. Initially, the agent is unfamiliar with the environment and the consequences of the actions. So the agent performs random actions that helps with some learning experiences. The model works as: the reinforcement learning agent performs a particular action, from the set of all available actions, inside the environment; now the environment processes the action to evaluate the next state that the agent will move to and also the rewards that the agent acquires on moving into the consequent state; after processing, this information is sent to the agent. This procedure is carried out until the goal or a terminal state is reached. The target is to collect as many rewards as possible.

Reinforcement is modeled as a Markov Decision Process. This includes:

- A finite set of states that is given by S
- A set of actions given by A
- Probability (P) that the agent moves to the next state from the current state under a particular action
- Immediate rewards (R) that result due to the transition to the new state.

The agent and environment interactions are considered to happen in discrete time steps that is denoted by t. Each state is associated with a reward value. This helps ensure the achievement of an optimal solution i.e. acquiring maximum rewards.

Associating the algorithm with our problem:
The task is to employ Reinforcement Learning to help the agent reach the goal. With respect to this project, the agent is Tom, a cat, that tried to reach the goal that is Jerry, a mouse. Here, the environment is considered to be a 5x5 grid space. The main target is to find the shortest path to reach Jerry which will correspondingly ensure a comparatively higher cumulative reward count.

2. DEEP Q-NETWORK

The application of deep learning to reinforcement learning has resulted in a thriving model which is the Deep Q-Network (DQN). We implement the DQN model in our project to help make fitting choices of actions to help reach the goal. The DQN is ideally considered to be the brain of the agent as this makes the choices of movement. DQN is a neural network structure that is trained with the agent's past experiences to enable visualizing patterns for optimal solutions. Elaborating the training process, the agent consists of a memory buffer of a certain capacity. This memory buffer stores a tuple which contains the current state, the action performed, the next state that is reached and the immediate rewards from the transition. After every action, the tuple containing the corresponding information is stored in the buffer. During training, a small batch of the data from the memory is taken and fed to the NN. This helps the DQN understand the relationship between the states and actions and the related results; figure out patterns and get an optimal answer.

The agent either performs a random action or a reward based action based on the value of a parameter known as the exploration rate or epsilon. When the agent is not performing a random action, then the DQN comes into play. It computes the immediate reward values for all the available actions when performed on the current state and it returns the action that causes the higher reward value.

3. TOM AND JERRY SPORT IMPLEMTATION

A python script helped implement the Tom and Jerry game. The implementations involved executing an environment, random actions, the DQN, the memory buffer and the agent.

Report #1: The parts of the code implemented by me were the DQN model which consisted of 3-layer neural net; two hidden layers and an output layer. Then the exponential decay formula for epsilon that governs the random action count and finally the calculation of Q-values based on the steps.

Report #2: The role of the respective features in training the agent is explained below:

ENVIRONMENT:

The environment was a 5x5 grid world space inside which resides the agent and the goal. The environment is responsible for performing the action specified by the agent; which in turn means that the environment is also responsible for updating the states after a transition. Thus, it consists of a method called "step()" which accepts the action, mentioned by the agent, as a parameter, then executes the action and returns the next state and the immediate rewards for that action. Now it updates the state to the new state reached. This process is carried out until the goal is reached. Once the ultimate state is reached, it marks the end of one episode. To begin with a new episode, the states have to be set back to the initial positions. This is employed by a method called “reset()” which returns the agent to its initial position.

RANDOM ACTIONS:

We know that the agent is initially unaware of the surrounding environment. So in the beginning of the process, the agent chooses random actions for transition. The agent passes on the available actions which in our case is: LEFT, RIGHT, UP, DOWN. The environment executes this action and returns the rewards and the next state. This is achieved for a few episodes and the results are stored in the buffer. These experiences come in handy for the training of the DQN. Random actions help explore the environment and get familiar with it.

BRAIN:

The DQN otherwise called the Brain of the agent is the most vital component of the entire structure. This comprises of a neural network that trains on the data from the buffer to make better action predictions in future. The NN consists of a dense 3-layer network; 2 hidden layers and an output layer. The activation functions for the hidden layers are ReLU and for the last layer we use the Linear activation. The DQN takes as input a tuple with 4 values; these represent the coordinates of the agent and the goal. Every time the agent is required to make a proper action chose, the DQN outputs the Q-values for all the actions for the given state; the agent gets to pick the one that gives the highest reward.

MEMORY BUFFER:

This helps store the experiences of the agent. The memory capacity is limited. Every time an action is performed its corresponding observation tuple is sent to the memory. Once the limit is reached, to make space for the new tuples, the oldest ones are removed. This functions in a first-in-first-out (FIFO) way.

AGENT:

An agent is considered to be a computer program that performs tasks on behalf of the user. The agent involves a method called “act()” which returns the action to the performed. This is the action that is passed to the step function of the environment. Based on the value of the exploration rate, the agent either chooses a random action and requests the DQN to predict an action based on the rewards. We know that the results of every action is given to the buffer for training purposes. For this, we use a method called “observe()” that collects the tuple and gives it to the memory. Finally, we use the “replay()” method. This method is the start of the actual training process. This ensures the phenomenon of Experience Replay i.e. it ensures that the agent learns from the recent as well as the past observations.

4. HYPERPARAMETERS

Report #3: The influence the hyperparameters have on the performance of the agent is shown below:

There are a few parameters that influence the working of the structure. They are:

- Epsilon
- Discount factor
- No of Episodes

The hyperparameters are interrelated to one another. Adjusting just one of the parameters to an ideal value makes no big difference in the performance of the model. Tuning of hyperparamters is an essential and crucial task that needs to be performed with careful observation and extreme scrutiny.

4.1 EPSILON or EXPLORATION RATE:

The agent is required to perform a number of random actions which helps explore the environment. Here, the agent performs actions without working towards reaching the goal. It examines the different actions and the variety of results they return. This gives the agent a wider knowledge about the environment and thus aid making better action choices. This is governed by the exploration rate. And in order to not perform a notably large number of random actions a decay factor is applied to epsilon to help control the random action count. The exploration rate is given by,

$$\boldsymbol{\varepsilon = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) * e^{-\lambda|S|}}$$

where
ε_{max} is the maximum rate at which an agent randomly decides its action
ε_{min} is the minimum rate at which an agent randomly decides its action
λ is the rate of decay. This value (lambda) is a very important factor that determines the random action count.

Min_Epsilon	Max_Epsilon	Lambda	Mean Rewards
0.05	1	0.00005	6.301
0.5	1	0.0005	4.287
1	1	0.00005	-0.160

The table shows that with significant changes to the epsilon values, the mean rewards decreases significantly. The drop has been clearly noted for extreme values of epsilon.

4.2 DISCOUNT FACTOR (γ):

Discount factor is a parameter used in the calculation of the Q-values whose value lies between 0 and 1. It is considered to give the difference between the present reward and the future reward; it corrects the future rewards. It is also used in the calculation of Return (G_t) where Return gives the total discounted reward for a particular time.

Discount Factor(γ) Value	Mean Rewards
0.99	6.301
0	6.401

This shows that larger the gamma value, the resultant discount is smaller. Thus the agent focuses on maximizes the rewards long-term.

4.3 NO OF EPISODES:

An episode is considered to be the time frame that comprises of a sequence of actions until a terminal stage is reached, that is the progress from the initial state to the final state (goal or a state after which movement is not possible). The no of episodes plays a vital role in the training of the DQN. As the agent proceeds through the episodes, it gets familiar to the environment and starts to figure out patterns to help achieve the shortest path to reach the goal.

No of episodes	Mean Rewards
10000	6.301
4000	4.603
1000	1.494

This shows that with decrease in the number of episodes, the mean rewards decreases. This shows that a sufficient amount of training is required to achieve high rewards.

The correlation between Epsilon and the number of episodes is given below:

The graph shows the relationship between the Episode count and the Epsilon or Exploration Rate value. The graph shows a significant drop through the episodes. Initially the value of epsilon is high as the agent is unfamiliar with the environment and we require the agent to explore different options. And as the number of episodes' increase, we require the agent to perform less random actions and more goal-oriented.

This is why we have added a decay factor to help progressively increase the epsilon value. In our code, we have initialized the decay factor to 0.00005.

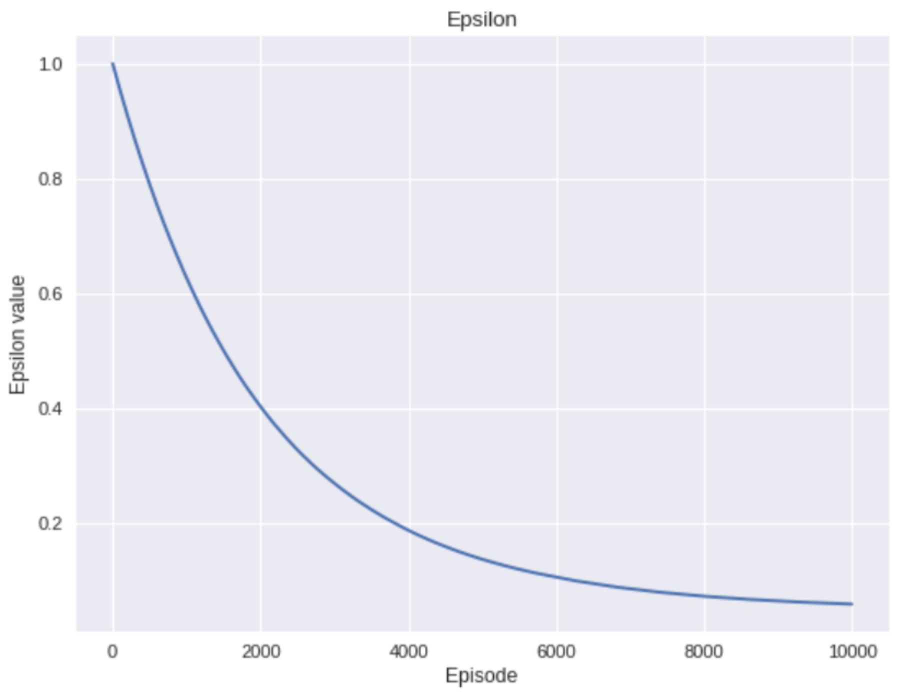


Fig 1: Episode VS Epsilon Graph

The correlation between Reward and the number of episodes is given below:

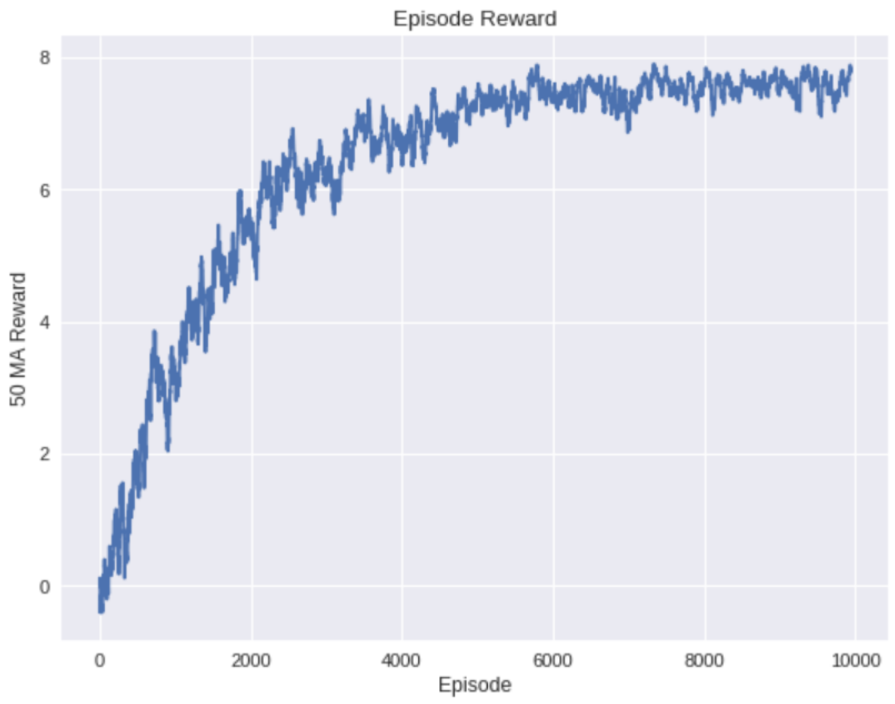


Fig 2: Episode VS Reward Graph

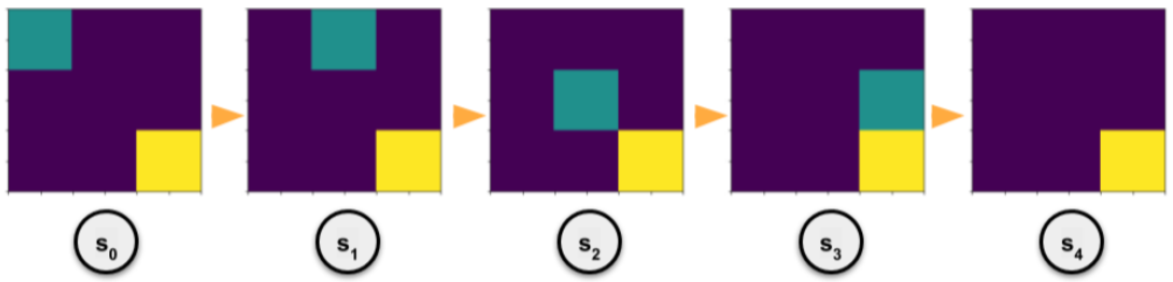
The graph shows the relationship between the Episode count and the Rewards value. The graph shows that as the no of episodes increase the cumulative rewards also increase.

WRITING TASK:

Question 1: Explain what happens in reinforcement learning if the agent always chooses the action that maximizes the Q-value.

Considering that the agent always chooses actions that maximize the rewards. This way, if the agent finds a particular path to reach the goal, the agent sticks to that particular path. Elaborating on this with our project case, Tom chooses every action based on the immediate rewards. It follows the path that this approach leads to and might end up reaching Jerry. Now that we have made an assumption that Tom chooses reward-based actions, after every reset, Tom will execute the same path to reach Jerry. Although the goal i.e. Jerry is reached we need it to the shortest path to Jerry. In some cases, reward-based movements cause exploiting only the closest sources of rewards, even if there are higher rewards at a distance.

Considering the state transitions from writing task question #2,

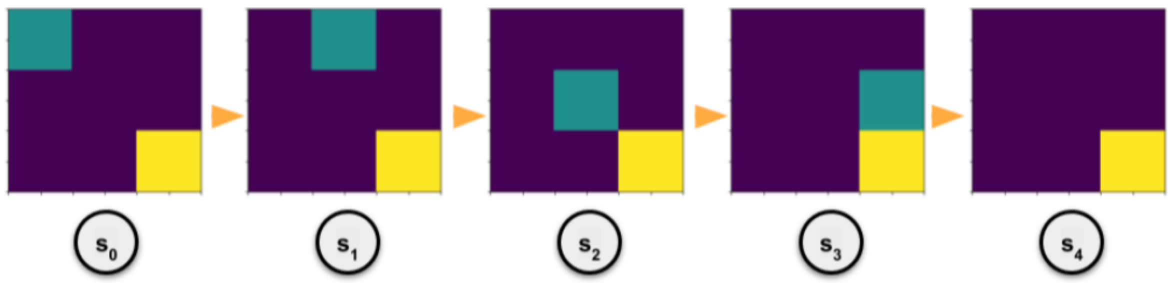


Here, the agent moves based on rewards. It moves RIGHT- DOWN- RIGHT- DOWN and it successfully reaches the goal. In this case, this is an optimal solution i.e. the shortest path to Jerry. But this is not the only optimal path for this problem. There are multiple choices and Tom will not be able to explore them under our current assumption.

To help overcome this, we introduce the parameter called the Exploration Rate or Epsilon. We know that the agent chooses an action based on comparing the randomly generated value with epsilon. If the random value is greater than epsilon, it implies that the agent knows enough to predict the right action. Otherwise, the agent performs a random action. We know that this helps exploring other possibilities. So we start with a big epsilon value that enables the agent to explore the environment and then progressively reduce the value once the agent has learned about estimating Q-values. To help with this reduce in the epsilon value over time, we use a decay factor 'lambda'. This ensures that the agent does not just stick to the initially acquired local maximum but due to the familiarity of the environment, it can achieve the global maximum by considering all the variety of possibilities. We know that epsilon is given by,

$$\boldsymbol{\varepsilon = \varepsilon_{min} + (\varepsilon_{max} - \varepsilon_{min}) * e^{-\lambda|S|}}$$

Question 2: Calculate Q-value for the given states and provide all the calculation steps.
Given the states,



We need to calculate the Q-values for the specified set of states. Q-table or the Quality-table holds the information to choosing the best action for the respective state. The procedure for calculating the values of the Q-Table are:

- Initialize the values of the Q-table arbitrarily for all state-action pairs.
- Choose an action to be performed for the current state.
- Perform the action and observe the resultant state and the rewards.
- Update the Q-value for the particular action.
- Repeat the steps until the goal is reached or till the end of an episode.

For our problem,

- Green title: Agent (Tom)
- Yellow title: Goal (Jerry)
- The actions performed for the given states: RIGHT- DOWN- RIGHT- DOWN
- Goal: Shortest path to reach Jerry
- Reward allocation: 1-when agent moves closer to the goal; -1 –when agent moves away from the goal; 0- when agent does not move at all.
- Discount factor: 0.99
- Grid: 3x3
- Possible States: 9
- Available Actions: LEFT, RIGHT, UP, DOWN

Q-Table- Arbitrarily initialized to zero

STATE	**UP**	**DOWN**	**LEFT**	**RIGHT**
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0

3	0	0	0	0
4	0	0	0	0

This our case, the agent chooses RIGHT action. Since this is not the end of the episode the Q-value is given as,

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

So for the first state,

- Q (s0, right) = 1 + 0.99 * maxQ(s_{t+1}, a)

Then the second action is DOWN. This is given by,

- Q (s1, down) = 1 + 0.99 * maxQ(s_{t+1}, a)

Then the third action is RIGHT. This is given by,

- Q (s2, right) = 1 + 0.99 * maxQ(s_{t+1}, a)

Then the final action is DOWN. This is given by,

- Q (s3, down) = 1

Once the final state s4 is reached, the agent has reached the goal and there are no more movements to perform. So the Q-values for all the actions at all the states is 0. Now we backtrack through the states to complete the Q-values. So,

- Q (s3, down) = 1, as per the formula for Q-values, if the episode terminates at the next step then Q-value = the current reward. So, s3 = 1.
- Q (s2, right) = 1 + 0.99 * 1 = 1.99, as we know the path the agent follows we can say that the maximum Q is Q (s3, down). So substituting that we get s2 = 1.99.
- Q (s1, down) = 1 + 0.99 * 1.99 = 2.9701
- Q (s0, right) = 1 + 0.99 * 2.9701= 3.9404

Therefore, updating the Q-table we get,

STATE	**UP**	**DOWN**	**LEFT**	**RIGHT**
0	0	0	0	3.9404
1	0	2.9701	0	0
2	0	0	0	1.99
3	0	1	0	0

4	0	0	0	0

5. CONCLUSION

This project helped learn about one of the most advanced algorithms of machine learning which is Reinforcement Learning. This project assisted in comprehending the conventions of deep reinforcement learning algorithms such as the Deep Q-Network. We implemented the Tom and Jerry game by using the DQN network. The goal of finding the shortest path was achieved successfully. Thus this project has helped gain a precise understanding of RL.

REFERENCES

[1] Reinforcement Learning - https://medium.freecodecamp.org/an-introduction-to-reinforcement-learning-4339519de419
[2] Q-Learning - https://medium.freecodecamp.org/diving-deeper-into-reinforcement-learning-with-q-learning-c18d0db58efe
[3] Reinforcement Learning - https://en.wikipedia.org/wiki/Reinforcement_learning
[4] Agent - https://en.wikipedia.org/wiki/Software_agent