ML 574 Project 4
(with Bonus)

Daniel Amirtharaj

damirtha@buffalo.edu
UB Person Number 50291137

1 Projrect overview

This project aims to develop an agent that is able to learn to take actions that would maximize reward based on its interaction with an environment. Here an agent tom has to learn to catch jerry in a grid environment that rewards tom as it gets closer to jerry in the shortest path possible. This is an application of reinforcement learning whereby the program learns on its own how to make choices that would benefit it based on a set criteria (reward or regret).

Objective The objective here is to implement snippets that were left unimplemented in the given program. The following components had to be added,

1. A deep neural network to learn the non-linear mapping between past observations and Q-values for a given state and action.
2. A parameter Epsilon to determine a trade-off between exploration (taking random actions to try new experiences) and exploitation (using previously known information to maximize reward).
3. The Q-value function itself for each state and action combination of a previously observed experience. This gives a cumulative reward that the agent might receive if it takes that state and action.

And finally tune hyper-parameters such as epsilon, gamma, number of episodes etc., to see which setting the program performed best in.

2 Analysis

Some of the key concepts used in the project have been described briefly here.

2.1 Reinforcement Learning

Reinforcement learning is a machine learning technique that helps an "agent" learn by interacting with its "environment", considering possibilities in actions that may help maximize its benefits in terms of positive reinforcements such as rewards or negative reinforcements such as regrets. Unlike traditional machine learning algorithms which are provided with the data to learn from, here the agents collect data from the environment by interacting with it and then learn from it.

It is a direct application of how living beings interact with their environment and learn from it without an external party's intervention. Such agents should be able to model human learning and will be able to acquire general intelligence to perform any task laid out to it, although such advancements are yet to be seen.

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

2.2 Q-learning

The concept of Q-learning is an extension of the Q-function computed for finite Markov decision processes. The Q-function gives the maximum expected reward that can be achieved through all future states and optimum actions when the agent starts from a particular state and executes a certain action. Future optimal Q values are weighted by a discount factor gamma. A high value of gamma makes the agent value all future rewards as much as the present reward, and a low value of gamma makes the agent value current rewards over future rewards.

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

Q-learning is all about learning the Q-values for a state and action given some observations regarding that state and action from past experiences from interacting with the environment. Initially its value is not known, but as the agent starts learning and collecting experience, its value starts to converge and using Q-values for all possible actions in a state, a good idea of what action is best (that maximizes Q) is obtained.

2.3 Exploitation vs. exploration (ϵ)

As the agent starts interacting with its environment, it has no idea what action will result in good rewards or the best Q value, so it must take random actions initially, and see what rewards it gets and then using this information, later make informed optimal decisions. This is the idea behind the parameter ϵ. It is a factor that determines whether or not the agent takes a random action or an informed decision.

There several possible ways to implement this, and one method calculates ϵ from 3 parameters set to constant values in the beginning, ϵ_{min}, ϵ_{max} and a step factor λ. ϵ_{min} gives the minimum value of ϵ or minimum probability that a random action will be chosen at any time and ϵ_{max} gives the maximum probability that a random action will be chosen. λ is a factor that determines how quickly the probability of random actions should decrease with the number of steps (|S|) that the agent has taken so far.

$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) * e^{-\lambda |S|}, \;\; \epsilon_{min}, \epsilon_{max} \; in \; [0, 1] \tag{1}$$

2.4 Deep Q networks

As the name suggests, deep Q networks combine the concepts of Q-learning with deep learning to form a powerful agent that is able to interact with its environment and get experiences and learn to take the best action from these experiences. Here, the Q values are learned both from observations (current rewards) and the non-linear mapping of states and actions obtained by learning from past observations. This non-linear mapping is achieved by the use of DNNs. The ability to model complex functions and non-linear data allows DNNs to model complex environments whose interactions are not easily predictable.

2.5 Deep Neural Networks

Neural networks is a highly efficient and robust algorithm used widely today. It emulates the human brain in terms of how we learn, and process data. Neural networks can represent complex non-linear functions and have applications in complex learning problems such as computer vision. Deep neural networks are an extension of neural networks, in that they model networks with lots of hidden layers. These hidden layers provide multiple layers of abstraction between the input data and the output label, enabling the network to model complex mappings accurately.

3 Method

The code-snippet implementations have been described and explained here (in order of appearance in the source code).

3.1 DNN

The DNN implemented was the "Brain" of the agent. Since its objective was to learn and predict Q-values for all possible actions, given an observation from an experience of the agent interacting with the environment,

- Its input was the observation, containing the current state s_t, action taken a_t, the state resulting from this action in the next time step s_{t+1} and the reward obtained as a result of this action r_t.
- The output and final layer was set to be the Q values for all 4 possible actions, and was given a linear activation.
- The 2 hidden layers were built with 128 neurons and a RELU activation function.

3.2 Parameter ϵ

The parameter ϵ was implemented in the "observe" function of the agent using the method described in the previous section. This function would be invoked repeatedly in the main function to allow the agent to observe its interaction with the environment and retain it in its memory.

A step counter is incremented every time an observation is made to ensure that ϵ decreases gradually with the number of steps into the learning process (or number of observations). This would allow the agent to take more random actions in the beginning and better optimal actions based on what it has learned as time progresses.

3.3 Q-function

The Q-function was implemented in the "replay" function of the agent using the method described in the previous section. This function would be invoked immediately after an observation to update its understanding of the Q values, for a sample set from its past experiences in memory.

Here predictions from the agent's current understanding of the environment are updated with the current observation, and this is fed to train the DNN to update the agent's understanding of the environment. Thus after every observation, the agent would have a slightly better idea of what action would result in a better outcome, ensuring that the agent learns eventually after a number of times of its interaction with the environment (number of episodes/time steps).

4 Results

4.1 Setting 1

On running the program with the hyper-parameters, ϵ_{min} = 0.05, ϵ_{max} = 1, λ = 0.00005, and number of episodes = 10,000. (Given parameters in the given code base). The agent was able to attain a mean reward of over 6 per 100 episodes. The model converged to a mean reward of 6 at 7900 episodes and continued to slowly increase till about 6.34 at 10,000 episodes.
The following table shows the mean reward (per 100 episodes) achieved over the number of episodes the agent has trained for.

Mean Reward	Number of episodes
1	800
2	1400
3	2100
4	3100
5	4700
6	8000

Table 1: Mean Reward (per 100 episodes vs. Number of episodes taken to reach the mean reward.)

Since the agent is able to learn to take the action that maximizes Q as the number of episodes and its interaction with the environment increases, its reward continues to increase. The following figure shows the reward received by the agent as the number of episodes increases.

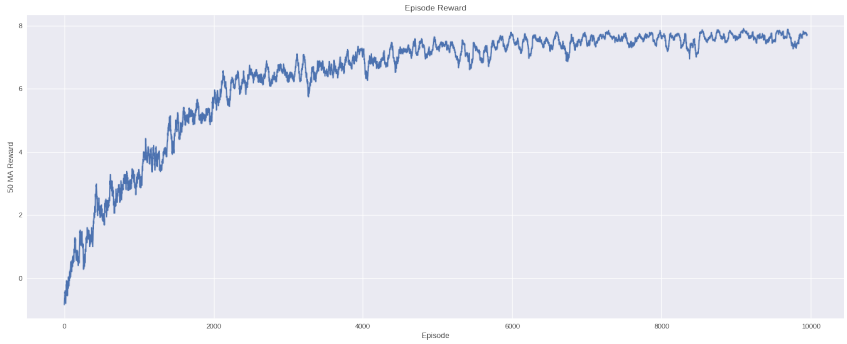

Figure 1: Reward per episode vs. Number of episodes.

The following figure shows the gradual decrease in ϵ over the number of episodes. The agent initially has a high ϵ that gradually decreases making the agent explore less and take actions based on its learned experiences as the number of episodes/time steps increase.

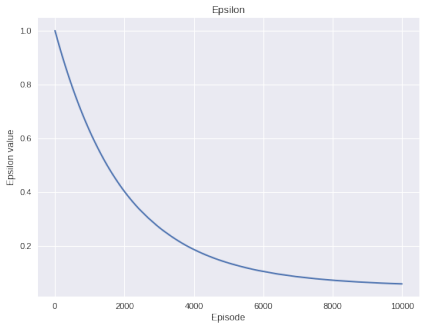

Figure 2: ϵ vs. Number of episodes.

4.2 Setting 2

On running the program with the hyper-parameters, ϵ_{min} = 0.05, ϵ_{max} = 0.5, λ = 0.00005, and number of episodes = 9,000. The agent was able to attain a mean reward of over 7 per 100 episodes. The model converged to a mean reward of 7 at 8,300 episodes. The following table shows the mean reward (per 100 episodes) achieved over the number of episodes the agent has trained for.

Mean Reward	Number of episodes
1	Nil
2	Nil
3	Nil
4	300
5	800
6	2500
7	8300

Table 2: Mean Reward (per 100 episodes vs. Number of episodes taken to reach the mean reward.)

The following figure shows the reward received by the agent as the number of episodes increases. It can be observed that this setting converges quicker to a higher reward compared to setting 1. This can be attributed to a lower ϵ_{max} = 0.5, forcing the agent to explore less and rely on its own experiences in relatively fewer episodes. Since the environment is quite simple, the agent is able to understand it without having to explore too much.

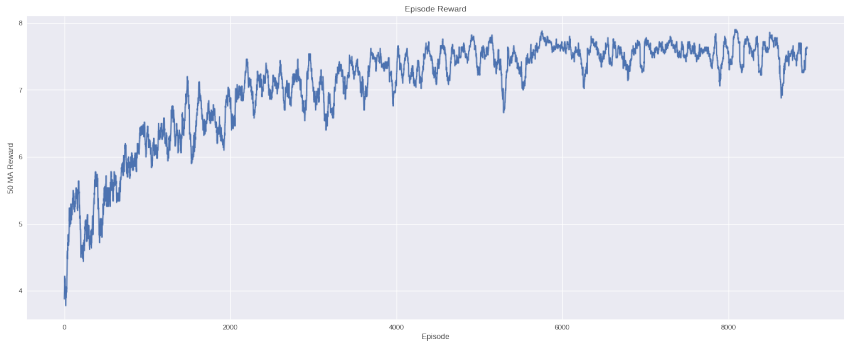

Figure 3: Reward per episode vs. Number of episodes.

The following figure shows the gradual decrease in ϵ over the number of episodes.

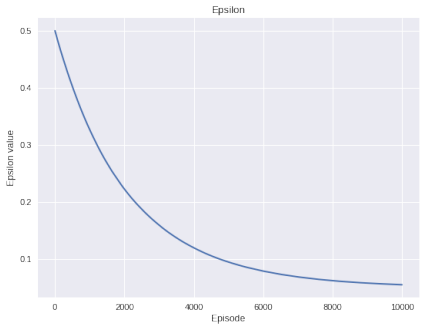

Figure 4: ϵ vs. Number of episodes.

4.3 Setting 3

On running the program with a lower λ = 0.005, keeping other hyper-parameters unchanged, ϵ_{min} = 0.05, ϵ_{max} = 0.5, and number of episodes = 2,000. The agent was able to attain a mean reward of over 7 per 100 episodes. The model converged quickly to a mean reward of 7.75 at 2000 episodes. The following table shows the mean reward (per 100 episodes) achieved over the number of episodes the agent has trained for.

Mean Reward	Number of episodes
4	Nil
5	Nil
6.8	300
7	400
7.75	2000

Table 3: Mean Reward (per 100 episodes vs. Number of episodes taken to reach the mean reward.)

The following figure shows the reward received by the agent as the number of episodes increases. It can be observed that this setting converged much quicker than the earlier settings. This can be explained again by the fact that we were dealing with a simple environment that did not require much exploring before the mean reward started to converge. The smaller λ resulted in epsilon values falling rapidly, resulting in minimal exploration.

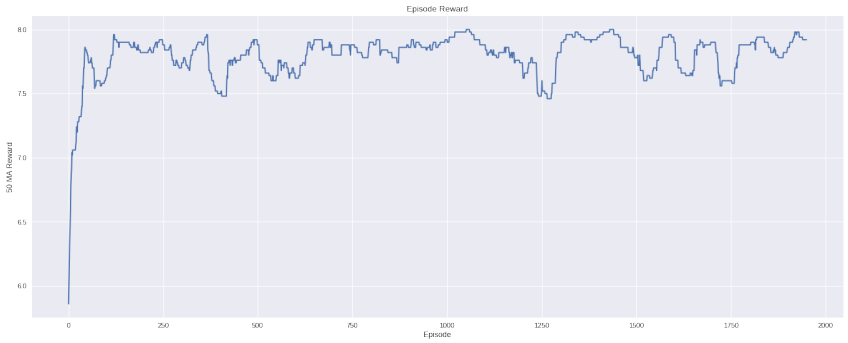

Figure 5: Reward per episode vs. Number of episodes.

The following figure shows the gradual decrease in ϵ over the number of episodes.

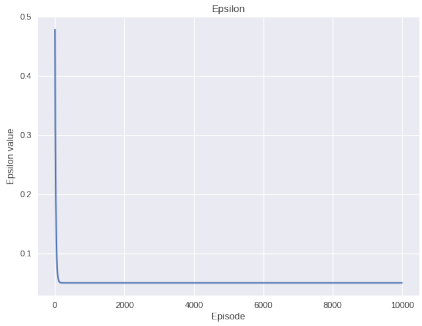

Figure 6: ϵ vs. Number of episodes.

4.4 Conclusion

It can be concluded that reducing the exploration parameter ϵ using ϵ_{max} and λ, resulted in faster learning and convergence of mean reward. This is a consequence of the simple environment which is learned easily by taking few random actions. This may not work for complex environments with complex reward schemes.

5 Written tasks

5.1 Question 1

Answer: If the agent always chooses to maximize its Q-value, it will not explore and instead will only exploit the information it already has about the environment. This is a scenario with ϵ set to zero always. In the beginning since the agent has no information about the environment it cannot make optimal decisions, simply because it has never interacted with the environment before. This will result in a situation where the agent tries to make an optimal decision from the very beginning having no knowledge about the environment itself.

In simple environments the agent may get away by learning something by taking an action it thinks is optimal without prior knowledge (randomly since the knowledge it has in the beginning is also random) and then learning how that affects its rewards, without any exploration. In complex

environments however, "random" knowledge will not help and may result in the agent making actions that it may think is optimal due to its limited knowledge but which may really be a bad decision. Thus, exploration is necessary to enable the agent to understand the environment and gain some knowledge about it.

There are many ways in which a trade-off between exploration and exploitation can be achieved.

- Random actions can be taken occasionally by the agent by introducing a decaying parameter ϵ. When deciding the most optimal action to be taken based on the Q-values of all actions, this parameter can be used to define a probability that the agent will take a random action or use the most optimal action. This will ensure that the agent will always explore with a probability of ϵ that decays as the agent gains experience, ensuring a balance between exploration and exploitation is maintained.
- By modelling the certainty or uncertainty that an action will result in an expected reward and using this information to choose an action. This can be used to explore actions that may result in higher expected rewards with higher uncertainties, allowing the agent to observe new experiences that may increase its Q-value. This maintains a good balance between exploration and exploitation, since exploration is done only when there is uncertainty in the result of an action.

5.2 Question 2

Answer: Calculation of Q-values for the given model with γ = 0.99,

When the agent reaches the goal, its Q-value will be zero, since it will not receive any reward after that. Thus,

Q(S_4 , a_{down}) = 0
Q(S_4 , a_{up}) = 0
Q(S_4 , a_{right}) = 0
Q(S_4 , a_{left}) = 0

At S_3, the agent will receive the following Q-values. Some values have been calculated using results from S_2, and the fact that some actions resulted in symmetric outcomes.

Q(S_3 , a_{down}) = 1 + 0.99(max_aQ(S_4 , a)) = 1
Q(S_3 , a_{up}) = -1 + 0.99(max_aQ(S_2 , a)) = 1.9701
Q(S_3 , a_{right}) = 0 + 0.99(max_aQ(S_3 , a)) = 0.99
Q(S_3 , a_{left}) = -1 + 0.99(max_aQ(S_2 , a)) = 1.9701

At S_2, the agent will receive the following Q-values. Some values have been calculated using results from S_1, and the fact that some actions resulted in symmetric outcomes.

Q(S_2 , a_{down}) = 1 + 0.99(max_aQ(S_3 , a)) = 1.99
Q(S_2 , a_{up}) = -1 + 0.99(max_aQ(S_1 , a)) = 1.9403
Q(S_2 , a_{right}) = 1 + 0.99(max_aQ(S_3 , a)) = 1.99
Q(S_2 , a_{left}) = -1 + 0.99(max_aQ(S_1 , a)) = 1.9403

At S_1, the agent will receive the following Q-values. Some values have been calculated using results from S_0, and the fact that some actions resulted in symmetric outcomes.

Q(S_1 , a_{down}) = 1 + 0.99(max_aQ(S_2 , a)) = 2.9701
Q(S_1 , a_{up}) = 0 + 0.99(max_aQ(S_1 , a)) = 2.9403
Q(S_1 , a_{right}) = 1 + 0.99(max_aQ(S_2 , a)) = 2.9701
Q(S_1 , a_{left}) = -1 + 0.99(max_aQ(S_0 , a)) = 2.9008

At S_0, the agent will receive the following Q-values, and the fact that some actions resulted in symmetric outcomes.

$Q(S_1, a_{down}) = 1 + 0.99(max_a Q(S_1, a)) = 3.9403$
$Q(S_1, a_{up}) = 0 + 0.99(max_a Q(S_0, a)) = 3.9008$
$Q(S_1, a_{right}) = 1 + 0.99(max_a Q(S_1, a)) = 3.9403$
$Q(S_1, a_{left}) = 0 + 0.99(max_a Q(S_0, a)) = 3.9008$

The Q-table is given by,

STATE	UP	DOWN	LEFT	RIGHT
0	3.9008	3.9403	3.9008	3.9403
1	2.9403	2.9701	2.9008	2.9701
2	1.9403	1.99	1.9403	1.99
3	0.9701	1	0.9701	0.99
4	0	0	0	0

Table 4. Q-table for the given environment, agent and goal.

6 Bonus

Two DQN environments were imported from the OpenAI gym library (with one ATARI environment) and 2 agents were trained on each environment.

1. The first environment used was the CartPole-v1 environment. On training the agent provided by the DQN library from stable-baselines using MLP (multi-level perceptron), the following results were obtained.

```
--------------------------------------
| % time spent exploring  | 34       |
| episodes                | 100      |
| mean 100 episode reward | 16.9     |
| steps                   | 1668     |
--------------------------------------
--------------------------------------
| % time spent exploring  | 2        |
| episodes                | 200      |
| mean 100 episode reward | 30.2     |
| steps                   | 4692     |
--------------------------------------
--------------------------------------
| % time spent exploring  | 2        |
| episodes                | 300      |
| mean 100 episode reward | 131      |
| steps                   | 17820    |
--------------------------------------
```

Figure 7: Mean reward increasing over time steps for the CartPole-v1 environment.

The CartPole-v1 environment, being relatively simple was easy for the agent to learn, this is evident by the good improvement in rewards over episodes.

2. The second environment used was the Breakout-v0 ATARI environment. On training the agent provided by the DQN library from stable-baselines using MLP (multi-level perceptron), the following results were obtained.

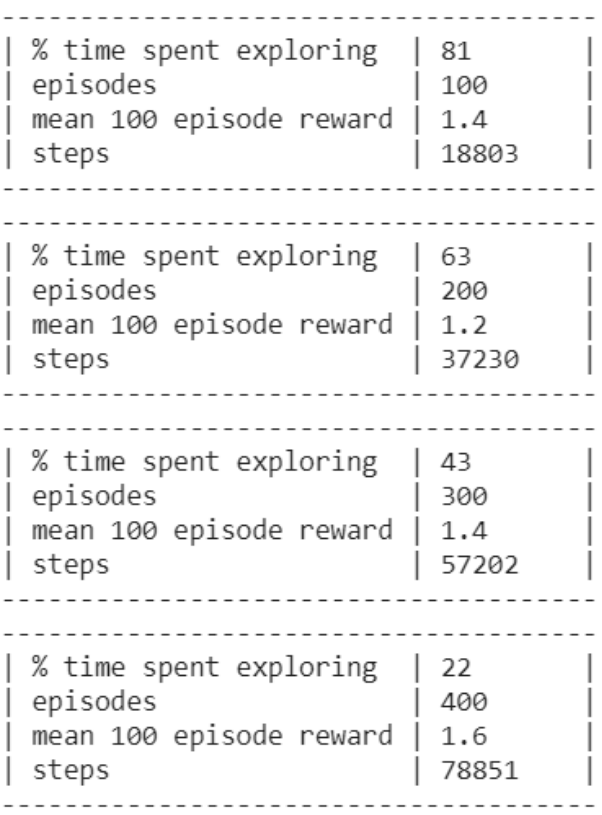

Figure 8: Mean reward increasing over time steps for the Breakout-v0 ATARI environment.

The ATARI environment being much more complex took a lot of time to train without large improvements in mean reward, although a mild improvement over episodes can be observed. The agent was not able to learn to maximize rewards over 100,000 time steps. Such complex environments would require millions of episodes for an agent to effectively understand it, due to the wide range of states, actions and rewards possible.

References

[1] Reinforcement learning
`https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html`

[2] Keras documentation
`https://keras.io/`

[3] Stable-baselines documentation
`https://stable-baselines.readthedocs.io/en/master/modules/dqn.html`

[3] Code and other materials posted in UB learns.