FIT3080

Artificial Intelligence

Assignment 3

Reinforced Learning (RL) & Machine Learning

Lim Zheng Haur

32023952

31st October 2023

Table of Contents

Part 1: Reinforcement Learning

Question 1: MDP value iteration

Implementation:

The implementation for Question 1 is a MDP value iteration. MDP value iteration is basically an iterative algorithm used to find the optimal value function and optimal policy in an MDP. The algorithm works by iteratively calculating the Q-value of each possible action for each state and storing the highest Q-value as the value for that state. The iterations are computed until convergence where there are no changes to the values of each state. Finally, the policy is chosen according to the values of each state.

The value of each state is calculated using the formula:

$$V_{i+1}(s) = \max_a \sum_{s'} T(s,a,s') \left[R(s,a,s') + \gamma V_i(s') \right]$$

This formula is computed by the function *registerInitialState()* and *computeQValueFromValues()* in the code.

The optimal policy is chosen using the formula:

$$\arg\max_a Q^*(s,a)$$

This formula is computed by the function *computeActionFromValues()* in the code.

Parameter Exploration:

The parameters that we could control in MDP value iteration are the discount factor, and the number of iterations. The discount factor is the value that controls the agent preference of having an earlier or later reward, a lower discount factor increases the agent preference for an earlier reward, and vice versa. The number of iterations controls the number of times the algorithm loops and recalculates the values of each state. There is a number of iteration where the values converge, and no changes are made even in future iterations, and ideally, this number of iterations should be used for the algorithm.

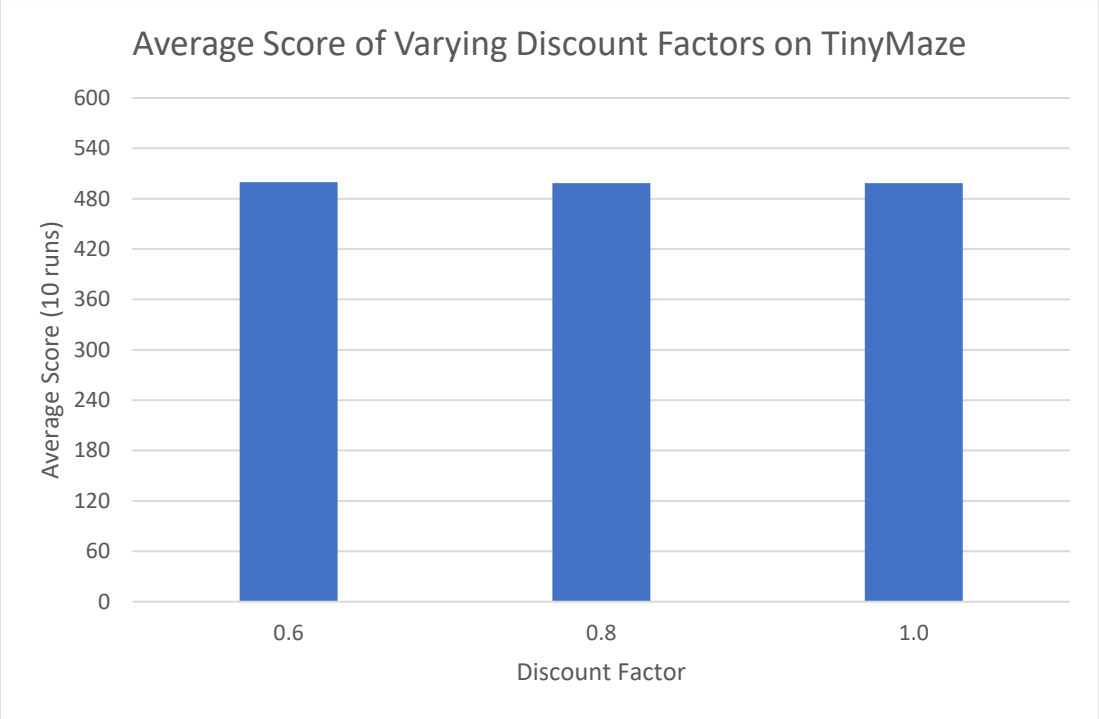

Figure 1: Average score for varying discount factors on TinyMaze

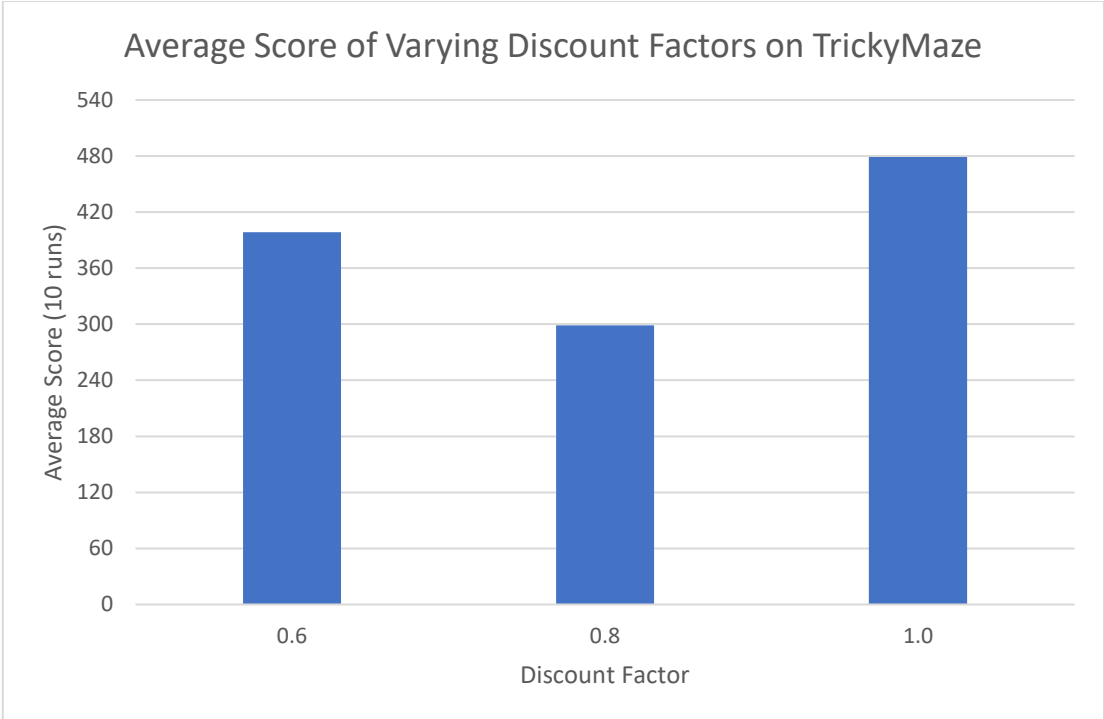

Figure 2: Average score for varying discount factors on TrickyMaze

As observed in Figure 1, the average score for different discount factors is the same. This might imply that discount factors do not affect the score, but this is not true as shown in Figure 2 where the average score greatly fluctuates when the discount factor is manipulated. The reason for this behaviour is that

discount factors do not significantly affect the values of layouts where there are 1 clear path from the agent to the goal, such as TinyMaze. On the other hand, on layouts where there are different options of paths for the agent to take, such as in TrickyMaze, then the discount factor plays a role in deciding the path of the agent. In layouts that have different options of path to take, the discount factor affects the decision of the agent to take the nearer reward (with a lower discount factor), or to take the safer but longer path reward (with a higher discount factor). The death of Pacman significantly affects the average score, hence although the longer path might result in lower scores, the average scores is still higher than taking the higher risk but shorter path.

This theory is used to decide the discount factor of each layout. There also exists certain examples such as BigMaze, where there are many options of food but the path to the nearest food is relatively safe compared to the other foods. In this situation, a lower discount factor encourages the Pacman to take the shorter path that is safer.

The number of iterations is decided by incrementally increasing the number of iterations until convergence is observed. Convergence is observed by including a short if-statement in the code to check if there are any updates in the values of the state in that specific iteration. If the sum of the update for each value of each state do not exceed 0.01, then convergence has occurred, and the iteration number is printed onto the terminal. This value is then used as the number of iterations for that layout.

Conclusion:

In implementing MDP value iterations, there are two parameters that could be adjusted to ensure the optimal results of the algorithm. These two parameters are the discount factor and the number of iterations. The discount factor is selected by observing the characteristics of each layout, for Pacman to take the shortest path, or to take a longer but safer path. On the other hand, the number of iterations is chosen by incrementally increasing the number of iterations until convergence is observed.

In this assignment, the number of iterations is defined at the start of each execution of the algorithm. However, a more optimal approach would be to set a limit for the number of iterations, and to use a condition to break the loop when convergence is observed. This would remove the inconvenience of restarting the execution of the code to modify the number of iterations.

Question 2: Q-Learning

Implementation:

The implemented algorithm for this question is Q-Learning. Q-Learning is a reinforcement learning algorithm to learn the optimal policy for an agent in an MDP. The algorithm mainly revolves around learning the Q-values through exploration or exploitation. The algorithm is similar to value iteration by using the Bellman Equation to update the Q-value, but rather than computing all the values for every state, the agent computes and stores the Q-value for the states that is explored. For each iteration, which is known as an episode, the agent will explore the map until it reaches a terminal state, and this is repeated for the number of training episodes selected. Finally, the values obtained from the training episodes are used by the agent to select the optimal policy to reach the goal state.

Parameter Exploration:

The parameters that could be adjusted in Q-Learning are the greedy action selection value (epsilon), discount factor (gamma), the learning rate (alpha), and the number of training episodes.

- Greedy action selection value (epsilon):

  The greedy action selection value is the variable that determines the greediness of the algorithm. Exploration and exploitation trade-off is a characteristic of reinforcement learning, where agents decide to exploit existing knowledge which results in a greedy algorithm, or to explore more of the map. A higher epsilon value would result in an algorithm that prefers exploration, and a lower epsilon value would result in a greedier algorithm that prefers exploitation.

  The selection of this value based on each of the layout. In smaller layouts, it is less expensive computationally for the agent to explore most of the map, and the computational cost increases exponentially on larger layouts. Hence, a higher epsilon value is generally preferred on layouts that are smaller. Larger layouts would require a lower epsilon value, but the value must not be so low that the agent refuses to explore the map completely. This value is mostly derived from a magnitude of tests conducted on each layout.

- Discount factor (gamma):

  The discount factor is practically the same as the discount factor from Question 1's value iteration algorithm. A lower discount factor prioritises immediate rewards while a higher discount factor does not. The reasons for selecting the discount factor is also the same as observed from Figure 1 and Figure 2.

- Learning rate (alpha):

  The learning rate is the value that determines how much the algorithm learns from the past experience. A higher alpha value would increase the significance of the new information that it receives, causing the Q-values of the states to increase rapidly. On the other hand, a lower alpha value would cause the information to affect the values of each state more conservatively. Since the Q-values increase rapidly with a high alpha value, the learning speed would also increase with a high alpha value as the values would converge quicker. The alpha value would also affect

the greediness of the algorithm as a higher alpha value would affect the values of the state more and cause the algorithm to rely on information learnt rather than exploring more.

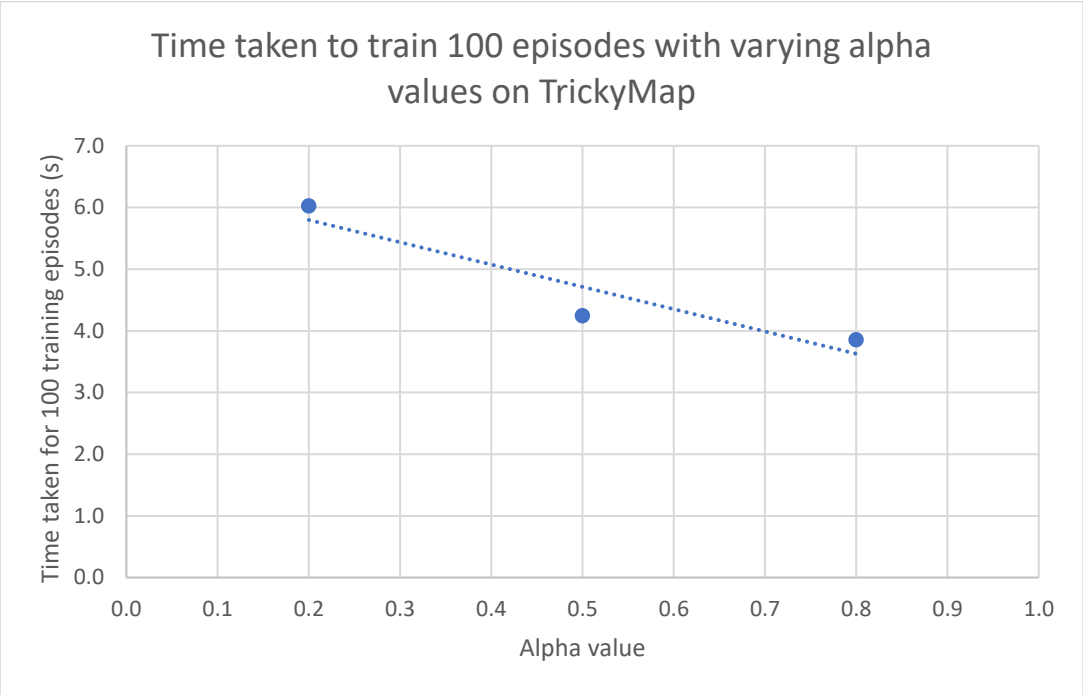

Figure 3: Time taken to train 100 episodes with varying alpha values on TrickyMap

As shown in Figure 3, the time taken to train 100 episodes with higher alpha values would decrease as a higher alpha value causes the Q-values to converge faster and leads to lesser exploration that would increase the training time.

The alpha value is chosen depending on various factors but mainly the size of the layout. The reasoning behind this is the same as the epsilon value, where it is computational cost is relatively cheaper on smaller maps which allows more exploration and learning, and vice versa on larger maps. However, the value should also not exceed a certain amount that would hinder the learning capabilities of the algorithm. Inescapably, the final alpha value is chosen through trial and error that maximises the performance of the algorithm.

- Number of training episodes:

  The number of training episodes is chosen with the same reasoning as the number of iterations from Question 1's value iteration algorithm. However, the values of the state in Q-Learning are unlikely to reach convergence on larger maps. Therefore, the number of training episodes is chosen with a mentality of "until the model is good enough" which is subject to experimentation.

Conclusion:

Due to the high number of parameters that could be adjusted for this Q-Learning algorithm, there might exist a more optimal combination of parameters. However, the parameters chosen for each layout are meticulously tested to reach the optimal combination. The main consideration in testing is to ensure a shorter training duration while maintaining the ability of the agent to reach the goal state. To achieve a highly accurate model, a low alpha value with high number of episodes would be preferred, but due to the lack of computational power, I have opted for a high alpha value low number of episodes approach.

Part 2: Machine Learning

Question 3: Single Layer Perceptron

Implementation:

The implementation for this question is a Single Layer Perceptron. The Single Layer Perceptron is the most basic form of a neural network which consists of a singly layer of nodes. Each node receives an input, processed, and the output from the activation function is the decision made by the perceptron.

The activation function of this algorithm could be selected by us. Therefore, the activation function that I have chosen is the sigmoid activation function. The sigmoid activation function is smooth and continuously differentiable. The gradient could be calculated at any point and introduces non-linearity into the network. Last but not least, sigmoid activation function is simple and easy to implement into the algorithm.

The loss function is also allowed to be selected by us. The loss function that I have chosen is derived from the Mean Squared Error (MSE) for training the perceptron. The MSE is commonly used in regression problems, but in this case, I have use it along with the activation function for this classification of best action task. The formula for the loss function is:

$$\frac{2}{N}\sum_{i=1}^{N}(\hat{y}_i - y_i) * \hat{y}_i(1-\hat{y}_i) * x_{i,j}$$

Parameter Exploration:

The parameters that we could adjust for the Single Layer Perceptron are the number of iterations and the learning rate.

- Number of iterations:

  The number of iterations represents the number of iterations the algorithm will iterate over the training dataset. The importance of obtaining the right number of iterations is because a lesser number of iterations might result in the model not converging, while a greater number of iterations might result in overfitting.

- Learning rate:

  The learning rate controls the step size of the weights when updating during training. If the learning rate is too high, the model might converge too quickly and be suboptimal. On the other hand, if the learning rate is too low, the number of iterations required for the model to converge would be very high as the model converges slower.

Based on this knowledge, I have conducted an experimental test for various combinations of learning rate and number of iterations to find the optimal combination. The results are as shown in Table 1 and Figure 4:

Learning rate	Number of iterations								
	20	40	60	80	100	120	160	200	250
0.01	163.60	357.95	206.95	411.25	338.65	196.75	375.15	180.23	262.53
0.1	210.50	196.95	218.10	236.25	299.60	179.65	419.90	281.47	221.40
1	287.70	237.80	190.36	365.63	322.56	236.40	222.77	266.87	205.57
3	297.60	243.07	247.37	240.30	241.40	322.50	301.67	290.53	296.93

Table 1: Average score of 30 runs with for each learning rate and number of iterations

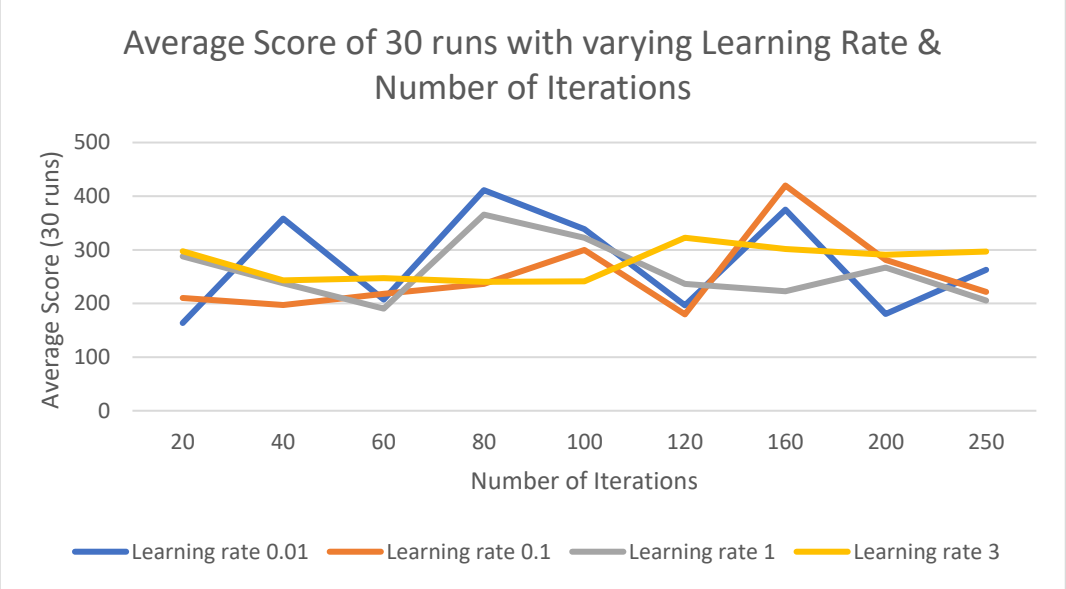

Figure 4: Average score of 30 runs with varying learning rate and number of iterations

From Figure 4, we could observe that the average score data has a lot of noise, and the data is very irregular. However, we could generally see a pattern where the line spikes at certain points where the number of iterations is the most suitable for the respective learning rate. Observing the graph, I have chosen the learning rate of 0.01 and 80 number of iterations for the maximum average score.

Conclusion:

The experimental data of the combinations of parameters is quite random and contains lots of noise. This is suboptimal as the chosen parameters might not be the optimal parameters. A larger number of tests is required to find the most optimal parameters. However, due to the time constraints and computational limitations, I have decided to use the best data that I could obtain at the current moment, Figure 4, and used it to determine my final parameters. Nonetheless, despite the low win-rates of the model, it is the model that I believe to output the highest average scores given the current circumstances.