Reinforcement Learning Coursework 2

Jamie Shing Him Ho CID: 01868630

November 23, 2023

Abstract

Contents

1 Q1 Tuning the DQN

In order to have the optimal hyperparameters, we will need to optimize them through tuning in order to fit the CartPole problem. The hyperparameters are listed in the table below, with the range of values we need to explore and their justification. The number of runs is set to 10 and the episodes to 250, this is because we are required to consistently achieve an average episode length of 100, and achieve the target of at least 50 consecutive episodes, where the total time needed to search for the optimal hyperparameters should be sufficient under 250 and due to computational constraint.

In order to find the base case amongst all these ranges of hyperparameters, I tried to use a hyperparameter optimization framework "Optuna" to perform this operation. I defined the optimal hyperparameters by getting the lowest score from all the combinations of the parameters, where the score is calculated as [250 - The number of Episodes with reward > 100], to get the batches where the rewards are all over 100. I randomly chose the batches, because this could reduce the variance when training the model. Running 50 runs for this search. After getting the base case, we further optimize it by testing out different combinations. However, due to computational power constraints, I only chose to test with a particular search space for the initial optimal result.

1.1 Q1.1 Hyperparameter

The first optimal base case after the 50 runs are:
$\epsilon = 0.7$, ϵ-decay = 0.99, Memory Buffer Size = 5000, Update Target Frequency Iteration= 2, Learning Rate = 0.001, Weight Decay = 0, Optimiser = RAdam, Activation Function = leaky relu, Policy Neural Network Architecture = 4, 64, 64, 2, Batch Size = 20.

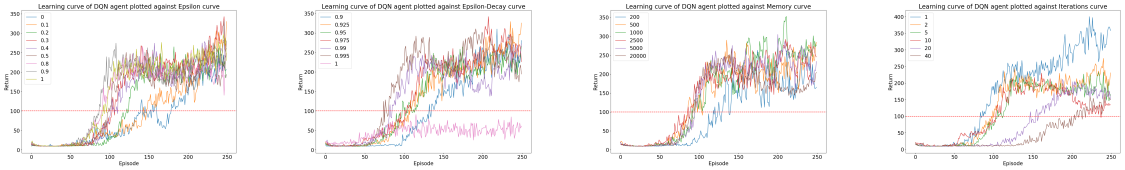
In order to further optimize it, we plot the graph to compare the learning of DQN against the hyperparameters in Figure 1. After selecting the potential better hyperperameter options, the second further optimized case are:
$\epsilon = 0.7$, ϵ-decay = 0.99, Memory Buffer Size = 1000, Update Target Frequency Iteration= 1, Learning Rate = 0.001, Weight Decay = 0, Optimiser = Adam, Activation Function = leaky relu, Policy Neural Network Architecture = 4, 128, 64, 2, Batch Size = 20.

The results can be seen in Figure 2, which is the first optimal base case and the second further optimized case are in orange and green respectively.

Hyperparameter	Value	Justification
Epsilon	[0.001, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0]	Higher initial value (e.g. 1.0 or 0.9) encourages more exploration. As training progresses, the agent gains more knowledge about the environment, and exploration decreases.
Epsilon Decay	[0.9, 0.925, 0.95, 0.975, 0.99, 0.995, 1]	Epsilon decay rate determines how quickly the exploration rate decreases, balancing between exploration and exploitation. When decay = 1, there is no decay. Slower decay (Close to 1) maintains a higher level of exploration for a longer period for more time to explore complex environments effectively. A faster decay (0.9) quickly shifts the agent towards exploiting.
Memory Buffer Size	[200, 500, 1000, 2500, 5000, 20000]	A larger replay buffer can store more experiences, which is crucial for learning from sets of past actions.
Update Target Frequency (Iteration)	[1, 2, 5, 10, 20, 40]	Determine how frequently the target network updates its weights from the policy network, affecting the stability of the learning process.
Learning Rate	[0.001, 0.01, 0.1]	A critical parameter in the optimization process, impacting how quickly the network converges to the solution, these are the standard numbers.
Weight Decay (λ)	[0, 0.05, 0.1, 0.2]	Regularization parameter to prevent overfitting, these are the standard numbers.
Optimizer	[SGD, Adam, RAdam]	Different optimizers can significantly impact the network's training dynamics and convergence speed, these are some of the common ones.
Activation Function	[ReLU, LeakyReLU, Sigmoid, Tanh]	Different activation functions can affect the network's ability to model complex patterns, these are some of the common ones.
Policy Neural Network Architecture	1 Layer: {4, 16, 2}, {4, 32, 2}, {4, 64, 2}, {4, 128, 2} 2 Layers: {4, 32, 16, 2}, {4, 64, 32, 2}, {4, 128, 64, 2} 3 Layers: {4, 64, 32, 16, 2}, {4, 128, 64, 32, 2}	Testing different network sizes to balance between model complexity and computational efficiency, testing all the possibilities.
Batch Size	[1, 2, 5, 10, 20]	Impacts the variance of the gradient estimates and the model's generalization ability. Smaller batches are used per run to ensure the quality and the amount of data we use is relatively small.

1.2 Q1.2 Learning Curve

From Figure 2, after trying out the first optimized base case using Optuna, we got the result of reaching 100 returns around 80 episodes, and the second further optimized case of fine-tuning through comparing the charts, successfully further optimizing it and reaching the return of 100 at under 30 episodes and continuously reaching higher than 100 return. The shaded areas are their standard deviations.

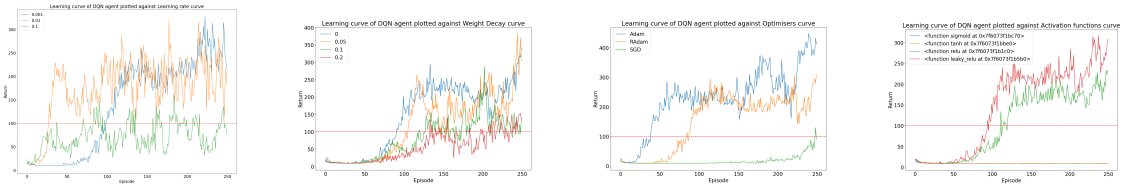


(a) Learning curve of DQN agent plotted against Epsilon curve

(b) Learning curve of DQN agent plotted against Epsilon-Decay curve

(c) Learning curve of DQN agent plotted against Memory curve

(d) Learning curve of DQN agent plotted against Iterations curve

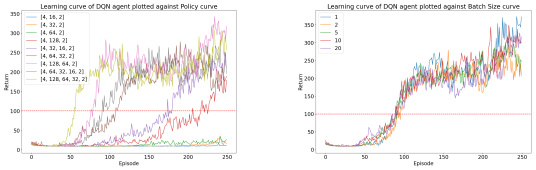


(e) Learning curve of DQN agent plotted against Learning rate curve

(f) Learning curve of DQN agent plotted against Weight Decay curve

(g) Learning curve of DQN agent plotted against Optimisers curve

(h) Learning curve of DQN agent plotted against Activation functions curve



(i) Learning curve of DQN agent plotted against Policy curve

(j) Learning curve of DQN agent plotted against Batch Size curve

Figure 1: Learning curve of DQN against hyperparameters

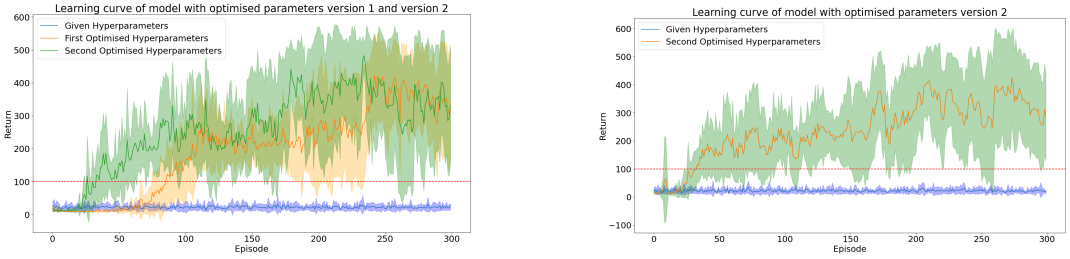


Figure 2: Learning curve of the model with optimized parameters version 1 and version 2

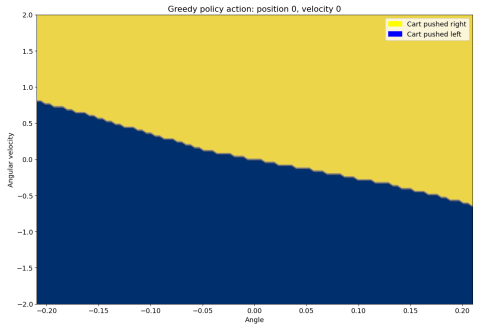
2 Q2 Visualize your DQN policy

2.1 Q2.1 Slices of the Greedy Policy

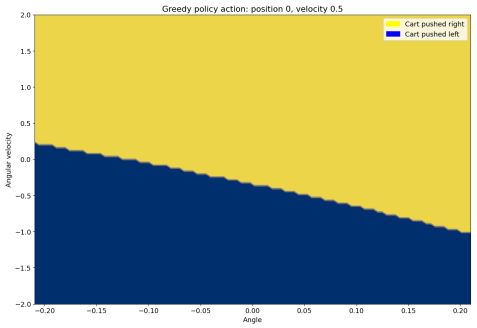
Figure 3 shows the pole angular velocity on the y-axis against the pole angle on the x-axis, showing the greedy policy action state changes where the pole is fixed to the cart position zero at different velocities. When the cart is pushed to the right, the pole undergoes an anti-clockwise motion acceleration and vice versa due to gravity.

When the cart has 0 velocities and the angular velocity is positive, the action taken is to push the cart to the right. This is because it will increase the opposite direction of circular motion to the anticlockwise motion to balance out the angular velocity to the right side, and vice versa where the action is to push the cart to the left for negative angular velocity. The larger the angular velocity and angle in the same direction, we are certainly taking the action of pushing the cart in the same direction as the pole is falling. The greedy policy shows this result in Figure 3a, low Q value when the action of pushing the cart to the left and a high Q value when the action of pushing the cart to the right.

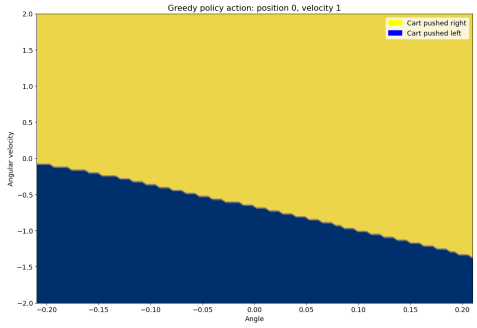
When the cart's velocity increases, the action needed to move the cart to the right increases. Since



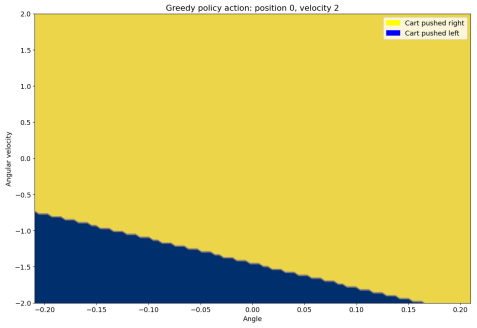
(a) Greedy policy action: position 0, velocity 0



(b) Greedy policy action- position 0, velocity 0.5



(c) Greedy policy action- position 0, velocity 1



(d) Greedy policy action- position 0, velocity 2

Figure 3: Greedy policy action on Angular Velocity against various Velocity at position 0

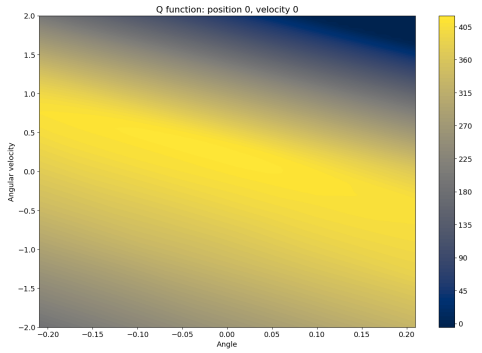
the cart is moving to the right with higher velocity, in order to counterbalance the pole, it requires force to move in the same direction to not let it have a greater clockwise angular acceleration motion, or else it will make the pole fall. This can be seen in Figure 10d, when we have a slightly negative angular velocity we are still going to push the cart to the right because it has a greater velocity, the action needed to push the cart to the right becomes more dominant, but the case of the most negative angular velocity with the negative angle remains the action to push the cart to the left to counterbalance for not to fall in the anticlockwise as the pole is falling in anticlockwise motion despite the cart velocity is at 2.

The separating line between the action to push the cart to the left or right has a negative gradient and is not perfectly straight. It is because the opposite magnitude of angular velocity and angle suggests moving the pole toward the center of angle 0. Positive angular velocity with a negative angle or negative angular velocity with a positive angle tends to move back to the center of angle 0. The exact value of this gradient changes according to the velocity of the cart.

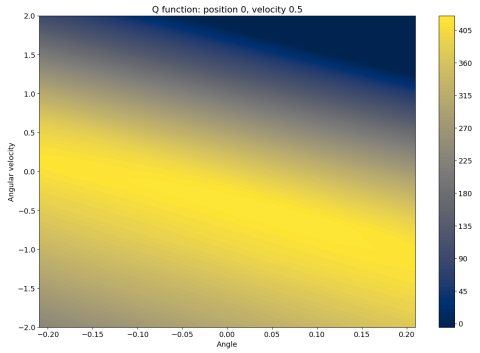
2.2 Q2.2 Slices of the Q function

If we are at the edge of falling, the chance to move the pole back to the origin of angle 0 is low. According to Figure 4a, when the cart velocity is 0, the edge case has a high magnitude of positive angular velocity and angle, and a high magnitude of negative angular velocity and angle, it has a low Q value close to 0 which is in blue color, the agent believes that it is not likely to move the pole back to the origin. Oppositely, when the angular velocity and angle are close to 0, with the high angular velocity with a negative angle or vice versa, the agent believes it is desirable and has a high Q value in yellow color. These combinations aim at moving the pole back to the origin.

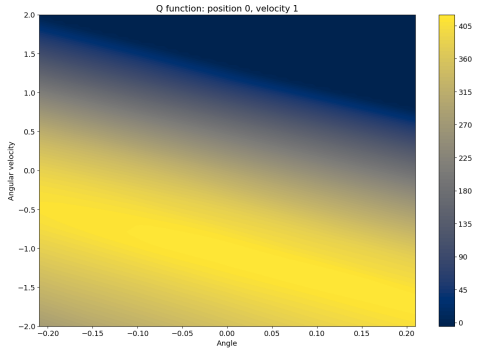
When the cart velocity increases to 2. The angular velocity and angle with a large positive magnitude have a low Q value close to 0. This is because moving to the right due to actions from Figure 3d increases the angular momentum to move in a clockwise direction while the angular velocity is positive to the right and the angle is at the max to the right, it is not likely to move the pole back to the origin. Oppositely, since the velocity is larger than 0, when both the angular velocity and angle are negative magnitude, and the decision made is to move the cart in the left direction, the agents are more likely to believe that these situations are desirable, because it is easier to counterbalance to



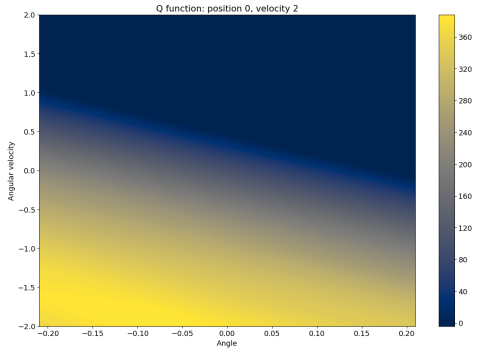
(a) Q function- position 0, velocity 0



(b) Q function- position 0, velocity 0.5



(c) Q function- position 0, velocity 1



(d) Q function- position 0, velocity 2

Figure 4: Q function on Angular Velocity against various Velocity at position 0

angular momentum to the left and the cart is already having a velocity to the right.