Reinforcement Learning : Coursework 2

Sakshi Singh [CID : 01925165]
Department of Bioengineering
MEng Biomedical Engineering

18 November 2023

Contents

1 Question 1 : Tuning the DQN

1.1 Question 1.1: Hyperparameters

An open source hyperparameter optimization framework to automate hyperparameter search called *Optuna* was used. Manually selecting optimal hyperparameters can be time-consuming. *Optuna* automates the process by systematically exploring different combinations of hyperparameters, making the search more efficient. To tune the agent, there was a wide range of hyperparameters that were explored:

1. Epsilon (ϵ) and ϵ-decay : Epsilon determines the trade-off between exploration (choosing random actions) and exploitation (choosing the best-known actions). The decay rate controls how quickly the agent shifts from exploration to exploitation. Slower decay rates allow for more extended exploration. Different rates were tested to observe their effects on learning stability.

Hyperparameter	**Search Space Explored**
ϵ	[0, 0.2, 0.4, 0.6,0.8,1.0]
ϵ-decay	[0.95, 0.98, 0.99, 1.0]

2. Learning Rate: Different learning rates were tested to find an appropriate balance between learning speed and stability. Lower values ensure slower, more stable learning, while higher values promote faster but potentially unstable learning.

Hyperparameter	**Search Space Explored**
Learning Rate	[0.0001, 0.001, 0.01, 0.1]

3. Optimizer Type: Multiple optimizer types were explored to assess their impact on training. Different optimizers have unique characteristics, and their performance can vary depending on the task.

Hyperparameter	**Search Space Explored**
Optimizer Type	['Adam', 'SGD', 'RAdam']

4. Weight Decay: Weight decay (L2 regularization) was applied to the optimizer. Different weight decay values were tested to study their impact on preventing overfitting.

Hyperparameter	**Search Space Explored**
Weight Decay	[0, 0.001, 0.01]

5. Activation Function: Different activation functions were tested within the neural network layers to observe how they affect the learning process and convergence.

Hyperparameter	**Search Space Explored**
Activation Function	['ReLU', 'Sigmoid', 'Tanh','LeakyReLu]

6. Neural Network Architecture: The NN Architecture hyperparameter includes configurations with 1, 2, and 3 hidden layers and varying numbers of neurons (32, 64, 128). This allows us to explore how the depth and width of the neural network impact the agent's performance in the CartPole environment.

Hyperparameter	**Search Space Explored**
Neural Network Architecture	[[4, 16, 2], [4, 32, 2], [4, 64, 2], [4, 128, 2]] , [[4, 32, 16, 2], [4, 64, 32, 2], [4, 128, 64, 2]] , [[4, 64, 32, 16, 2], [4, 128, 64, 32, 2]]

While running the *Optuna* study, a fixed value for batch size, buffer size, and update iterations was used.

Smaller batch sizes can introduce more randomness into the learning process, while larger batch sizes can slow down training due to increased computation requirements.[1] A batch size of 25 is used because it strikes a balance between computational efficiency and learning stability.

The buffer size determines how many experiences are stored for replay during training. A larger buffer size of 20000 allows the DQN to remember a diverse set of past experiences, which can help the agent generalize better and avoid forgetting valuable information.

In DQN, there are two neural networks - the **policy network** and the **target network**. The target network's parameters are updated less frequently to stabilize training. By using a value of 2 for update iterations, the target network is updated every two episodes, which can help smooth out the learning process and improve stability without updating too frequently, which might introduce variance.

For each hyperparameter and it's value in the search space, learning curves were also plotted (Return vs. Episode) and the final hyperparameter values obtained from this method also closely matched the ones from *Optuna*. The hyperparameter values found after doing the hyperparameter search are shown in Table 1 on the next page.

Hyperparameter	**Value**	**Justfication**
Epsilon (ϵ)	0.8	This value was selected to strike a balance between exploration and exploitation. It was found through empirical evidence (observing the learning curves) that an initial high exploration rate helps the agent explore the state space effectively in the early stages of training, which is crucial for learning.
ϵ-Decay	0.99	A high decay rate allows the exploration rate to decrease slowly over time, which extends exploration during training. This helps the agent discover optimal policies even in later episodes.
Learning Rate	0.001	This value is chosen for stability and controlled learning. This value was determined through looking at the learning curves to prevent the network from converging too quickly and overshooting optimal values.
Optimizer Type	Adam	The results of the Adam optimizer are generally better than every other optimization algorithm as seen through empirical evidence, have faster computation time, and require fewer parameters for tuning.
Weight Decay	0	No weight decay is applied as empirical observations through the learning curve indicate that regularization is not necessary for this relatively simple problem.
Activation Function	LeakyReLu	By looking at the learning curve for different activation while training, it was observed that leakyReLu had greater average return over 250 episodes than ReLu. It was also observed model does not learn how to solve the cart-pole problem when Sigmoid and Tanh were used.
Neural Network Architecture	[4, 128, 64, 2]	This architecture was chosen based on empirical observations, which demonstrated that it achieves better performance compared to other architectures. Overall, the architecture chosen achieves an average return of greater than 100 in just 50 episodes and converges quicker as compared to using architecture with just one layer.

Table 1: Hyperparamters used to tune the DQN.

1.2 Question 1.2: Learning Curve

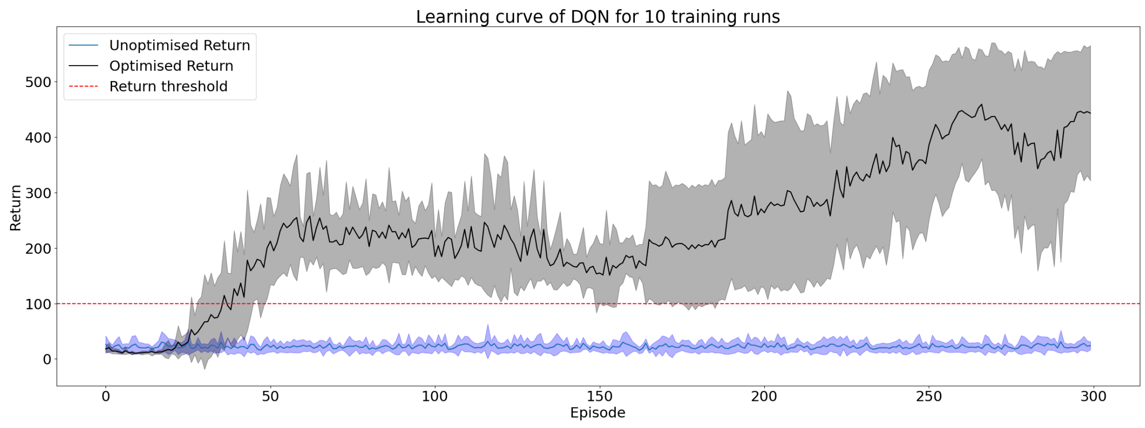


Figure 1: Learning curve of DQN with uoptimized and optimized parameters plotted as return per episode over time.

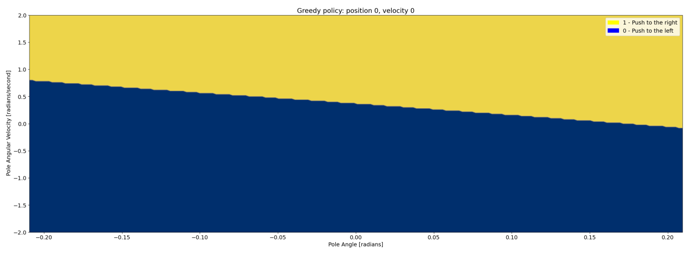
The graph in Figure 1 presents a comparative analysis between an unoptimized and an optimized Deep Q-Network (DQN) across 10 training runs. It is evident that the optimized DQN significantly outperforms its unoptimized counterpart. The unoptimized DQN, indicated by the blue line, shows minimal improvement and consistently scores below the threshold of 100, which serves as the benchmark for adequate performance. In contrast, the optimized DQN, represented by the black line, demonstrates a substantial increase in returns after an initial learning phase, consistently surpassing the return threshold of 100 in less than 50 episodes.

The shaded areas around both the unoptimized (blue) and optimized (black) return lines represent the standard deviation of returns across these runs.The wide shaded black region indicates that while the optimized DQN generally performs better than the unoptimized version, there is still considerable variability in its performance. Despite this variability, the optimized DQN's mean performance consistently exceeds the return threshold. This suggests that the optimizations have effectively improved the DQN's average performance, although individual runs may still experience significant fluctuations in returns.

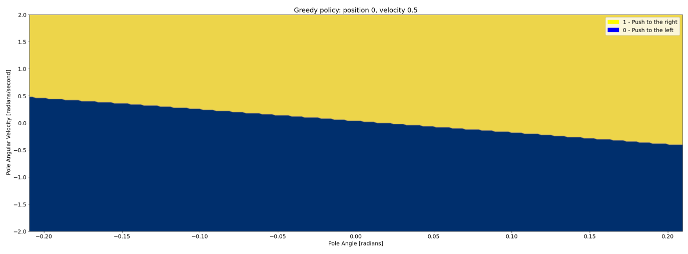
Overall, the graph suggests that while the optimizations applied to the DQN have led to a higher mean return that meets the target performance, the wide standard deviation for the optimized DQN suggests there's room for further stabilization of the learning process to ensure more consistent outcomes across all training runs.

2 Question 2 : Visualise your DQN policy

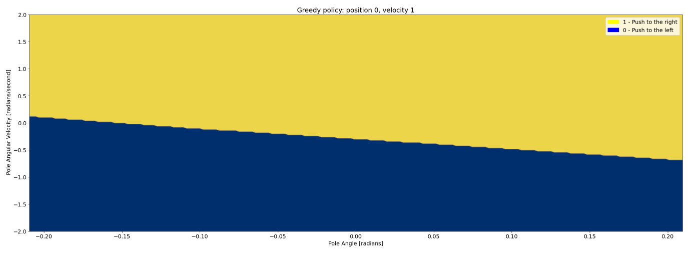
2.1 Question 2.1: Slices of the greedy policy action



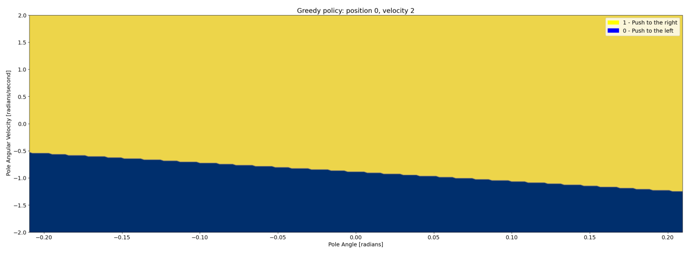
(a) Cart Velocity: 0



(b) Cart Velocity: 0.5



(c) Cart Velocity: 1



(d) Cart Velocity: 2

Figure 2: Greedy Policy Action Plot shown by Pole Angular Velocity vs. Pole Angle for different Cart Velocities and Cart position fixed to 0.

Figure 2 provides a visual representation of the interaction between the agent's greedy policy and the dynamics of the cart-pole system. It's essential to grasp the underlying mechanics of the system: when the cart is moved leftward, it imparts a clockwise angular acceleration to the pole, stabilizing it and vice versa.

1. Velocity and Angular Velocity Interaction:
   In scenarios where the pole exhibits zero velocity as seen in Figure 2(a), the policy's objective is to rectify its behavior based on angular velocity. When the angular velocity is positive (indicating the pole is tilting to the right), the optimal action is to nudge the cart to the right (action 1). This action amplifies the counterclockwise acceleration, aiding in stabilizing the pole. Conversely, when the angular velocity is negative (indicating the pole is tilting to the left), the

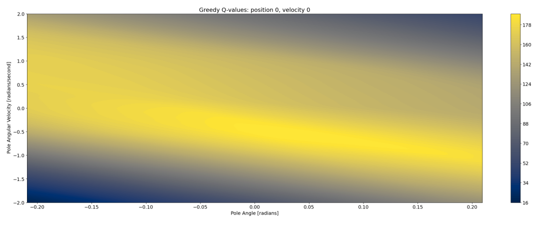
policy recommends pushing the cart to the left (action 0) to arrest further tilting and maintain equilibrium.

2. Impact of Increasing Velocity:
   As the cart's velocity increases, action 1 (rightward push) becomes the predominant strategy as seen in Figure 2(b),(c) and (d). This suggests that when the cart shifts leftward, the most effective course of action is to continue in that direction to prevent pole instability. The policy's primary aim here is to avert shifting the center of gravity in the opposite direction, a condition known to result in instability. Conversely, when the cart moves rightward, applying an opposing force (leftward push) generates a stronger clockwise angular acceleration, leading to pole instability.

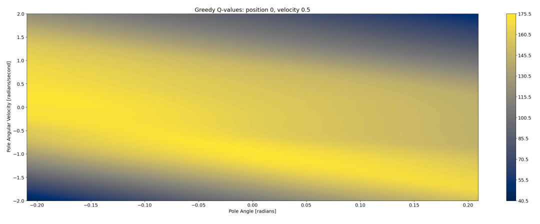
3. Rescuing the Falling Pole:
   Action 0, which involves pushing the cart to the left, is particularly crucial when the pole is in a state of falling to the left (negative angle and negative angular acceleration). This action effectively counteracts the fall, demonstrating the policy's ability to respond appropriately to critical situations.

4. Gradient of Separation Line:
   Notably, the division line in the visualizations does not adhere to a perfectly linear trajectory but exhibits a negative gradient. This arises because when the angle is negative and the angular velocity is positive, the pole gravitates toward the center (angle of 0), which aligns with the desired state. Similarly, when the angle is positive and the angular velocity is negative, the pole also tends to move toward the center (angle of 0). This behavior aligns with the policy's objective to maintain balance.

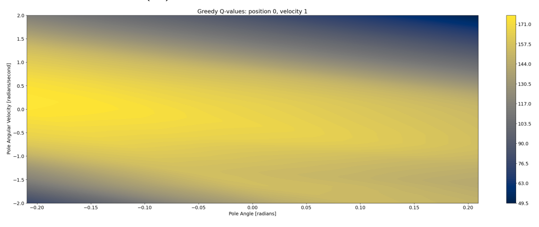
2.2 Question 2.2: Slices of the Q function



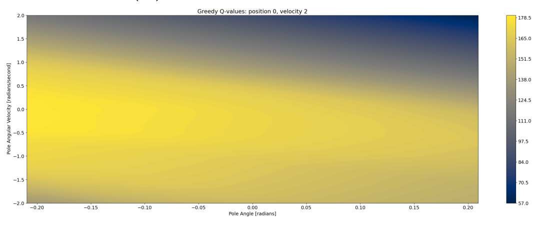
(a) Cart Velocity: 0



(b) Cart Velocity: 0.5



(c) Cart Velocity: 1



(d) Cart Velocity: 2

Figure 3: Greedy Q-Values Plot shown by Pole Angular Velocity vs. Pole Angle for different Cart Velocities and Cart position fixed to 0.

Visualizations of the greedy Q-values as seen in Figure 3 demonstrate the DQN's ability to learn and make informed decisions in response to different combinations of cart velocity, pole angle, and angular velocity. The transition from darker to brighter colors as cart velocity increases indicates the agent's increasing proficiency in stabilizing the pole under various conditions. This reflects the success of the trained DQN in achieving its goal of balancing the pole effectively in the cart-pole environment.

1. Stationary Cart at Center:
   In this scenario as seen in Figure 3 (a), when the cart is motionless and positioned at the center of the track, the Q-values exhibit interesting patterns. The bottom-left and top-right corners of the plot have Q-values close to 0. These states are considered undesirable because they indicate a precarious situation where the pole is almost falling. In these states, the pole's angle is near its maximum, and the angular velocity is pushing it further in that direction. This makes it challenging to control the pole's balance, leading to low Q-values.
   Conversely, the bright yellow patch from the top-left to the bottom-right represents desirable states. Here, the angular velocity opposes the angle's direction, causing the pole to move closer to the center (angle 0). The agent assigns high Q-values to these states because they lead to a more stable pole.

2. Increasing Cart Velocity:
   As the cart's velocity increases, there is a noticeable shift in the regions with high Q-values towards the bottom-left corner of the plot, as seen in Figure 3 (b),(c) and (d). The bottom-left corner corresponds to states where both the angular velocity and angle are negative, indicating that the pole is tilting to the left. As we know from Figure 2, the greedy policy suggests pushing the cart to the left when the pole is falling to the left. It's intuitive to assign high Q-values to these states because pushing the cart to the left in such situations increases the angular acceleration, helping to correct the pole's position and prevent it from falling.
   On the other hand, states where the pole is falling to the right while the cart moves rightward (top-right corner) receive lower Q-values. This is expected, as these are more challenging scenarios where preventing the pole from falling is difficult.

3 References

[1] Stooke, A., & Abbeel, P. (2018). Accelerated Methods for Deep Reinforcement Learning. ArXiv, abs/1803.02811.