CS6482 Deep Reinforcement Learning

Assignment 2: DQN Classic Control

Sem2 AY 23/24

Name: Muhammad Khubaib Akram

StudentID: 23098929

Why Reinforcement Learning is the Machine Learning Paradigm of Choice for This Task

Reinforcement Learning (RL) is particularly suited for environments where an agent must learn to make decisions through trial and error, optimizing its actions based on feedback in the form of rewards. This paradigm is distinct from supervised learning, which relies on pre-labeled data, and unsupervised learning, which seeks to find patterns without any labeled responses.

In tasks like the "MountainCar-v0" and "Pendulum-v0" from OpenAI's Gym, an agent learns to solve challenges through trial and error, receiving feedback in terms of rewards or penalties. This setup mimics the learning process in natural and artificial intelligence systems, making RL ideal for such problems where the objective is to maximize the cumulative reward (Sutton & Barto, 2018).

For "MountainCar-v0", the goal is to drive an underpowered car up a steep hill which it can only accomplish by building up momentum from the opposite hill. In "Pendulum-v0", the task is to swing up and stabilize a pendulum at its highest point. Both tasks have continuous state spaces and discrete/continuous actions, which are well-modeled by RL's framework where an agent iteratively improves its policy of actions based on feedback from the environment (Brockman et al., 2016).

The Gym Environment

OpenAI's Gym provides a standardized, easy-to-use suite of environments for developing and comparing reinforcement learning algorithms. Gym environments encapsulate complex decision-making tasks with a focus on achieving maximal cumulative rewards. In the "MountainCar-v0" environment, the agent is provided with a simple, two-dimensional state space representing the position and velocity of the car, and it must learn to navigate a policy that maximizes its reward: reaching the mountain top (Brockman et al., 2016).

The "MountainCar-v0" problem within OpenAI's Gym framework represents a classic control task designed to test the limits of reinforcement learning algorithms. In this environment, an agent must learn to control a car situated between two hills. The agent's goal is to drive the car to the top of the right hill within the least amount of time possible. This seemingly straightforward task is complicated by the car's lack of power, making it impossible to ascend the slope in a direct manner.

The Challenge

The car's engine is too weak to climb the hill without a run-up, and therefore, the agent must learn to leverage the car's momentum by driving back and forth to build up enough speed to reach the goal—a flag located at the peak of the right hill. This setting creates a problem space with a continuous, low-dimensional, two-variable observation space but a discrete action space.

Action Space

The agent has three possible actions to choose from: accelerate to the left, don't accelerate, or accelerate to the right. These actions are deterministically applied, with the car's underpowered engine and the gravity pulling it back into the valley playing significant roles in the car's resulting movement.

Observation Space

The state of the environment is described by a two-dimensional observation space: the horizontal position and the velocity of the car. These continuous variables are constrained by their respective physical limits, with the position ranging between -1.2 and 0.6 meters, and the velocity ranging between -0.07 and 0.07 meters per step.

Transition Dynamics

The dynamics of the car follow a simple, deterministic model. The velocity is influenced by the chosen action, subtracting a factor for gravity, which is proportional to the cosine of three times the current position (mimicking the effect of slope on the car's acceleration). This ensures that the car's behavior is consistent and predictable, allowing for an interpretable learning process.

Reward Structure

The agent receives a reward signal which, in this case, is structured as a penalty. For every timestep taken to reach the flag, the agent incurs a reward of -1. This negative reward

compels the agent to find the quickest path to the goal, as minimizing the number of steps directly maximizes the cumulative reward.

Starting State

Each episode begins with the car's position randomly initialized between -0.6 and -0.4 meters, ensuring that the agent does not start too close to the goal and must learn to maneuver from different starting conditions. The car's velocity is set to 0 at the start of each episode.

Episode Termination

An episode ends when the car reaches the goal position, which is when its position is greater than or equal to 0.5 meters. However, to prevent the agent from driving indefinitely without success, an episode is also truncated after 200 timesteps if the goal has not been achieved.

The "MountainCar-v0" problem thus presents a balance between exploration and exploitation. Agents must explore the momentum-building strategy while exploiting their knowledge to reach the goal efficiently. The environment serves as a benchmark for reinforcement learning techniques, testing an algorithm's ability to learn delayed gratification, backtracking, and exploration strategies.

3. Implementation for Mountain-v0

In the realm of reinforcement learning, three distinct strategies stand out in their approach to solving the classic "MountainCar-v0" challenge, a problem where an agent must learn to drive a car up a steep hill—a task it cannot achieve through direct acceleration due to gravity being stronger than the car's engine. The strategies are Q-Learning, Fixed Q-Learning, and Deep Q-Learning (DQN), each presenting its own methodology and nuanced benefits.

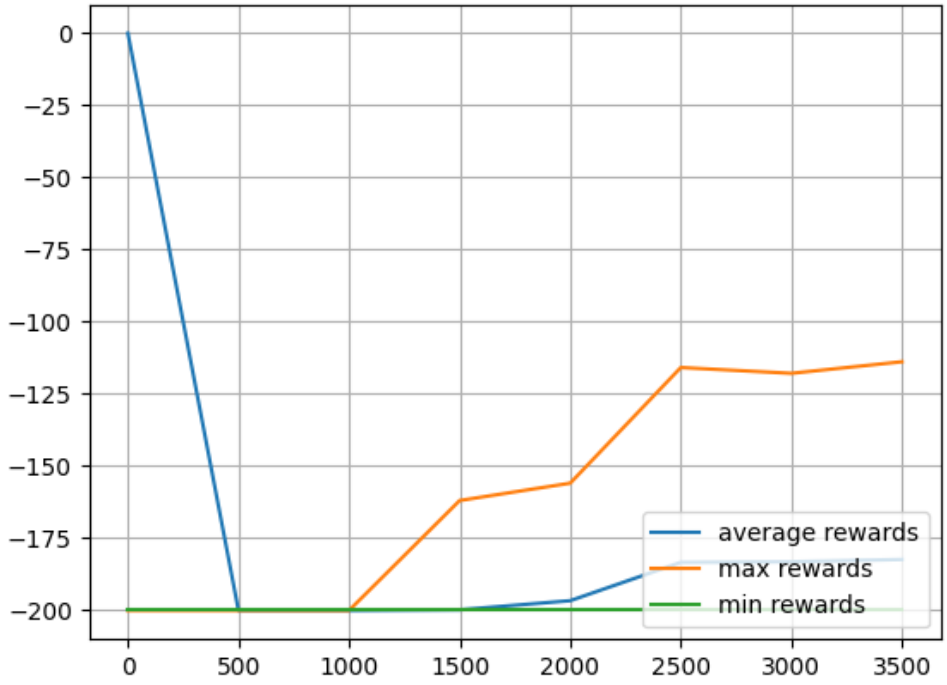
Q-Learning is a straightforward strategy that employs a Q-table—a grid-like structure where each cell represents the expected reward for taking an action in a given state. This method incrementally improves its strategy as it learns the consequences of actions taken in various states. In the provided Q-Learning graph, one observes an initial steep

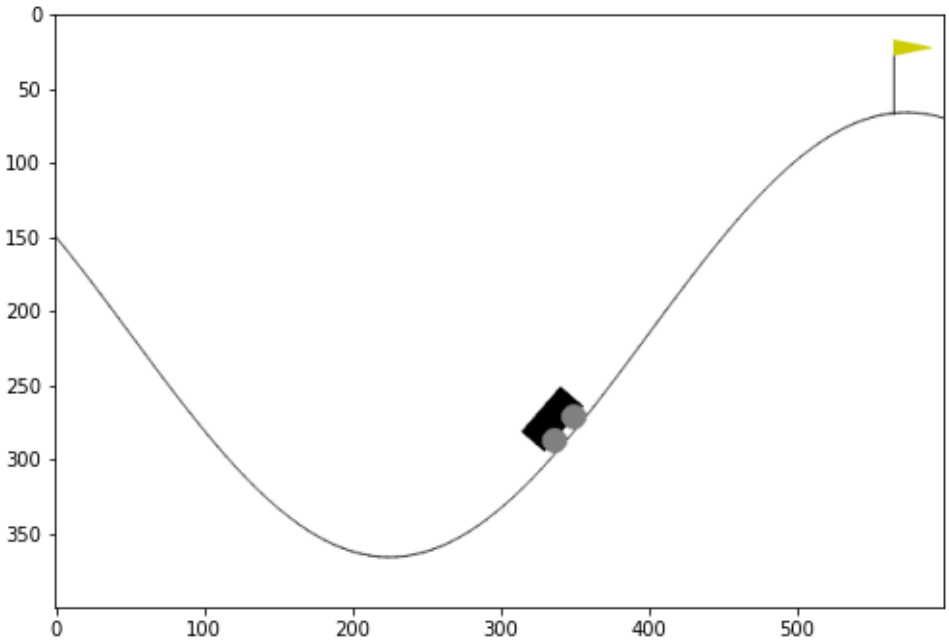
improvement in rewards, indicating the agent quickly learns from its naive initial strategy. As episodes progress, the average reward per episode improves, albeit with fluctuations. This depicts a learning curve where the agent gradually refines its policy to achieve the objective.

Fixed Q-Learning is a refined version of Q-Learning. It introduces a stability mechanism that addresses the "moving target" problem by utilizing two Q-tables—one for active learning and the other as a stable target for updates. This technique helps smooth out the learning updates, aiming to produce a more stable convergence to an optimal policy. The Fixed Q-Learning graph would ideally show a smoother progression in learning compared to standard Q-Learning, with less fluctuation in the rewards over episodes. However, the approach still suffers from limitations in handling large state spaces due to its tabular nature.

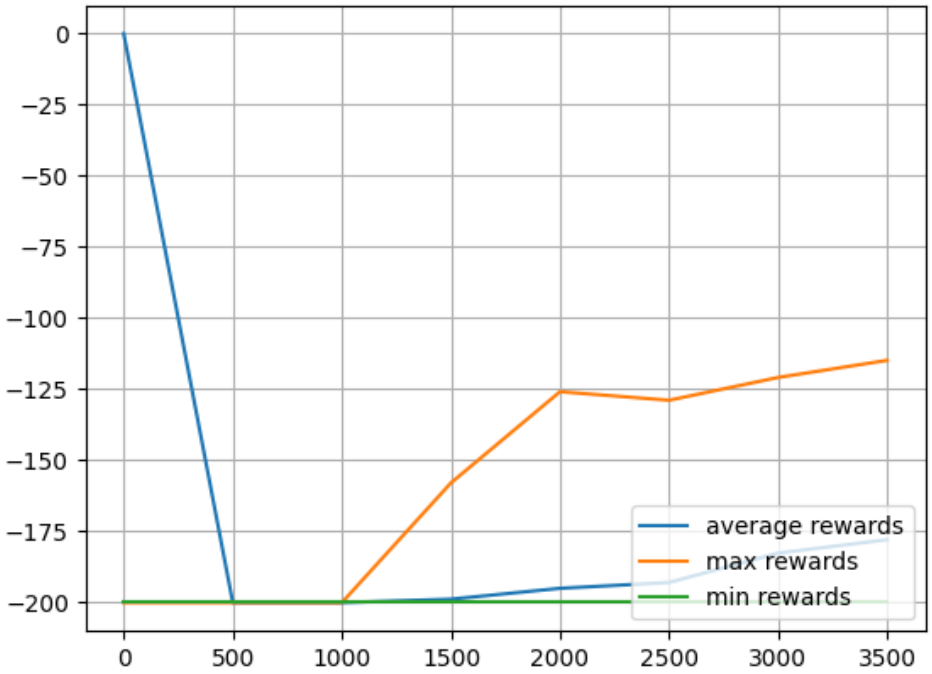
Deep Q-Learning (DQN) significantly expands the capabilities of the Q-Learning paradigm by replacing the Q-table with a deep neural network. This network is capable of generalizing across a vast, continuous state space, which is a common characteristic of many real-world problems, including "MountainCar-v0". DQN also employs techniques like experience replay and fixed Q-targets to further stabilize and optimize the learning process. The DQN graph should typically showcase a steadily improving performance curve, reflecting the neural network's ability to better generalize the task at hand as it learns from diverse experiences. The graph associated with DQN in the "MountainCar-v0" challenge would illustrate this progressive mastery—fewer steps to reach the goal as the episode count increases, thanks to the network's capability to approximate the complex relationship between states and actions.

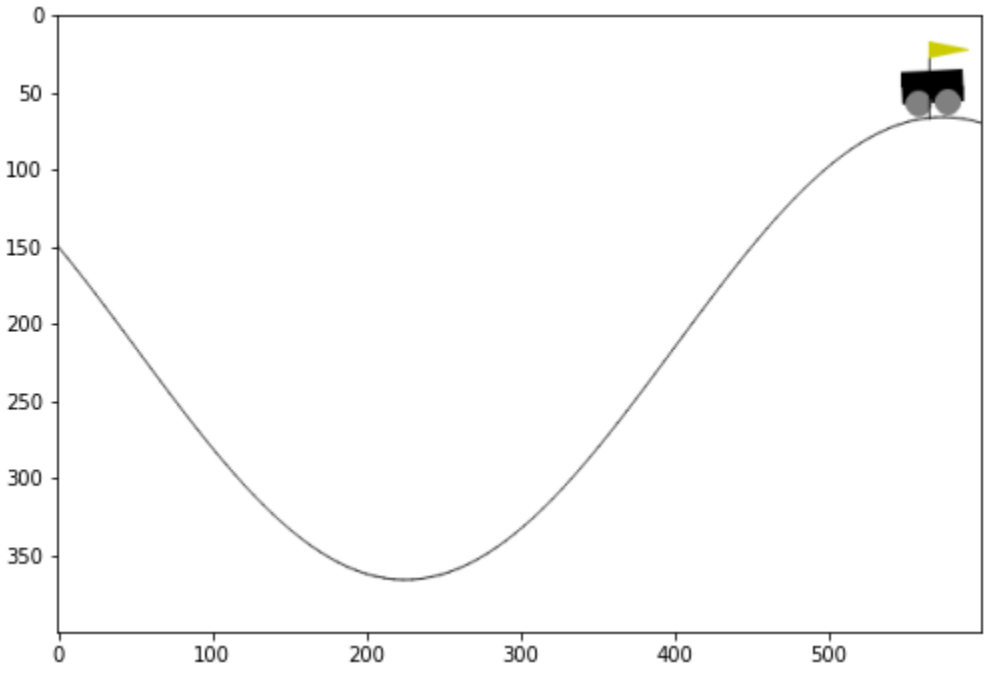
The graphs accompanying each method illuminate the learning dynamics at play. For Q-Learning, the graph indicates initial rapid learning, which tapers off as the agent begins to saturate the potential of what can be learned through a simple Q-table.



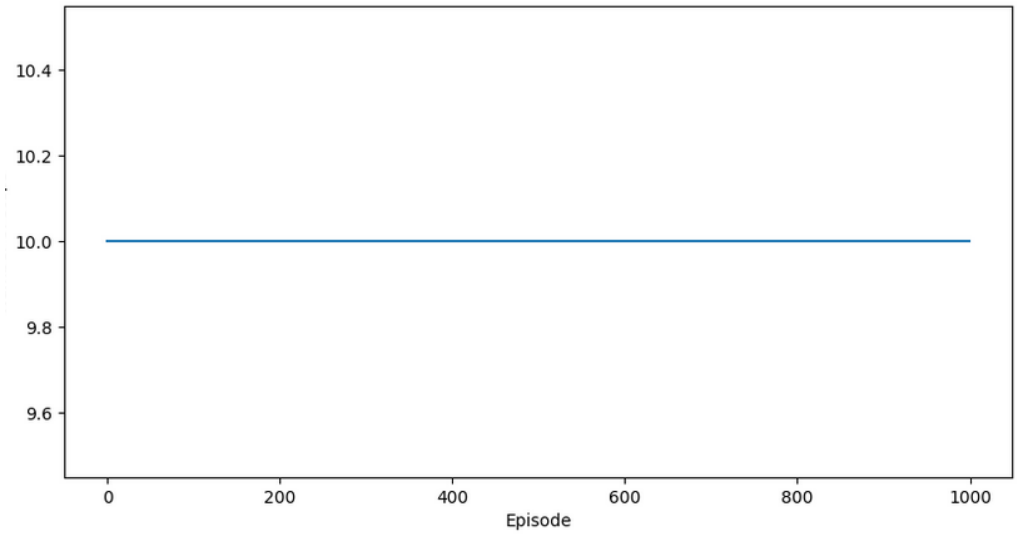


For Fixed Q-Learning, one would expect a less erratic improvement, reflecting the stabilizing effects of the dual Q-table approach.





The DQN graph stands as a testament to the power of neural networks in reinforcement learning, showcasing a consistent trend towards fewer steps needed to solve the episode—although, in this case, the plot provided does not reflect significant improvements as it kept getting crashed if I increased the steps or episodes, which might suggest the need for further hyperparameter tuning, additional layers or neurons in the network, and more resources.



The number of neurons in a neural network layer (as in the Deep Q-Learning model) significantly impacts the model's capacity for learning. More neurons provide a more substantial capability to capture complex patterns in data but also bring the risk of overfitting. The selected 64 neurons per layer in the Deep Q-Learning model are a moderate number, offering a balance between learning complexity and the risk of overfitting.

The decay of epsilon is crucial for balancing the exploration-exploitation trade-off. An initially high epsilon encourages exploration, allowing the agent to gather diverse experiences. Over time, as epsilon decays, the agent increasingly exploits its learned policy to achieve better results. The rate of decay for epsilon influences how quickly the agent shifts from exploration to exploitation. If decayed too quickly, the agent may not explore enough to learn an optimal policy. Conversely, if decayed too slowly, the agent might continue exploring suboptimal actions for too long, delaying the learning of an optimal policy.

In summary, the comparison between Q-Learning, Fixed Q-Learning, and Deep Q-Learning highlights the evolving complexity of reinforcement learning strategies, from simple tabular methods to sophisticated neural network-based approaches. Each has its strengths and weaknesses, with the simpler Q-Learning approaches being more transparent and easier to implement, while Deep Q-Learning offers greater potential in complex environments at the cost of increased complexity and computational demand.

Pendulum-v1

The GYM environment

The `Pendulum-v1` environment from OpenAI Gym is a classic reinforcement learning problem that represents a continuous control task. The objective in this environment is to swing a frictionless pendulum up so it stays upright.

Environment Details:

Observation Space: The observation is a 3-dimensional vector consisting of the cosine and sine of the pendulum's angle from vertical, and its angular velocity. These continuous values give a complete state representation of the pendulum.

- `**cos(θ)**`: The cosine of the angle can be used to determine the pendulum's position relative to the upright position.
- `**sin(θ)**`: The sine of the angle provides the direction of the pendulum's deviation from the upright position.
- `**θ˙**`: The angular velocity denotes how fast the pendulum's angle is changing at any given time.

Action Space: The action space is also continuous and 1-dimensional, which represents the torque applied to the pendulum. Actions can range from applying a maximum torque in one direction to a maximum torque in the opposite direction.

Reward Function: The reward function is designed to encourage the pendulum to stay upright with the least amount of energy (torque) expended. It penalizes both the angle from vertical (wanting it to be zero) and the torque applied. It is defined as:

$$R(t) = -(\theta^2 + 0.1 * \dot{\theta}^2 + 0.001 * torque^2)$$

This means that a smaller angle deviation from the vertical, lower angular velocity, and less torque applied all contribute to a higher reward.

Episode Termination: An episode does not end in a traditional sense, as the pendulum environment is designed to be continuous. However, for the purpose of training, episodes are often capped at a fixed length (e.g., 200 timesteps).

Task Goal: Unlike some other environments, there is no defined goal state to be reached. The task is ongoing, and the goal is to maximize the reward, which happens when the pendulum swings up to the highest point (vertical) and stays there with minimal movement and effort (torque).

Challenges:

Continuous Control: This environment is significantly challenging due to its continuous action space. This makes it more complex than environments with discrete action spaces, as the action to be taken at each step is not a simple choice from a finite set but requires selecting a value within a continuous range.

Complex Dynamics: Since the pendulum is affected by gravity and requires building momentum to swing up, the control policy needs to learn not just to balance the pendulum but also to effectively swing it up, which involves understanding the pendulum's dynamic physics.

Sparse and Misleading Rewards: The reward structure can be sparse and, at times, misleading. Small rewards for not doing the correct action can accumulate and mislead the learning algorithm unless carefully managed.

The Pendulum-v1 environment, with its continuous action and state spaces, poses a significant challenge for traditional reinforcement learning algorithms like Q-Learning. Let's delve into the implementation and results of Q-Learning for this problem, based on the provided code and graph.

Q-Learning for Pendulum-v1

Implementation:

The Q-Learning code for Pendulum-v1 involves discretizing the continuous state space into a finite number of bins to create a Q-table. This Q-table then guides the agent's action selection, with each entry corresponding to the expected cumulative reward of taking an action in each state. The learning process updates this table as the agent explores the environment.

2.1 Capture and Sampling of the Data:

The state of the pendulum is discretized, and for each time step, the agent records the current state, action taken, reward received, and the new state. The actions are determined by an ε-greedy policy, which either selects the best-known action or explores a new action, with the exploration rate decaying over time.

2.2 Network Structure and Hyperparameters:

There isn't a network structure in the traditional neural network sense since Q-Learning here uses a table-based approach. However, key hyperparameters in this context include:

- The learning rate (α) that controls how much new information overrides old information.

- The discount factor (γ) which determines the importance of future rewards.

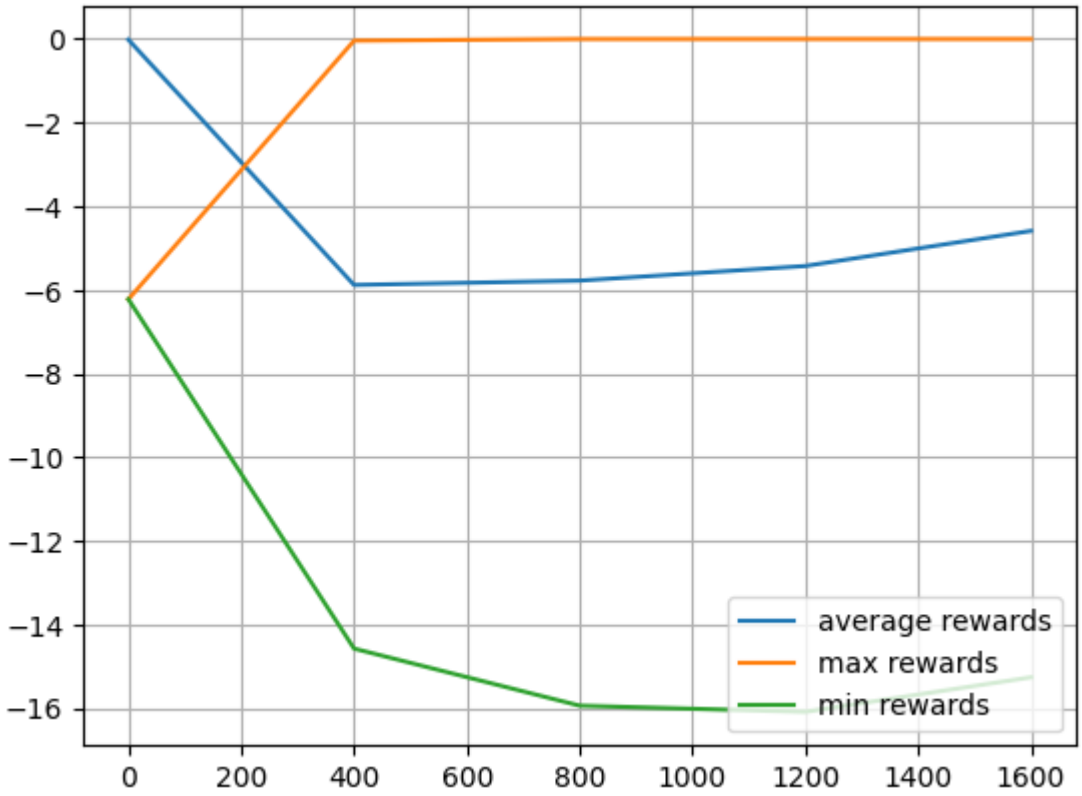
- The epsilon (ε) for the ε-greedy policy and its decay rate that manage the exploration-exploitation trade-off.

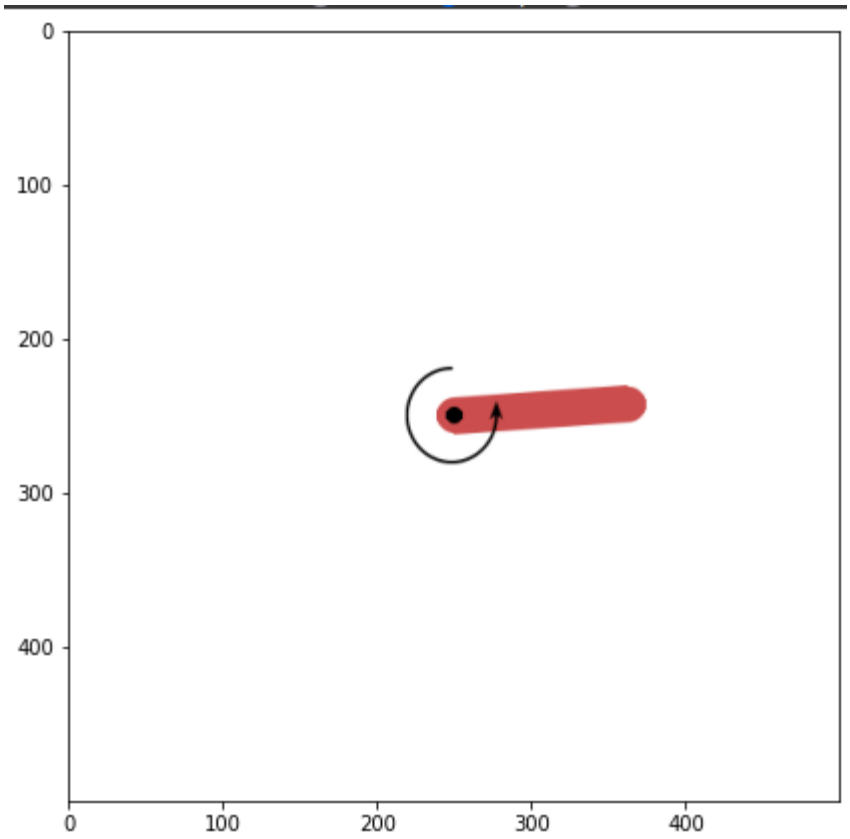
2.3 Q-Learning Update on Weights:

The Q-value update is conducted according to the Bellman equation. The learning step calculates the temporal difference error (TD error), which is the discrepancy between the current Q-value and the estimated optimal future value (reward plus discounted maximum Q-value of the next state). The Q-value is then updated to reduce this error.

Evaluation of Results:

The attached graph displays average, maximum, and minimum rewards per episode over a span of episodes. An initial period of learning can be observed, where the agent accumulates knowledge about the environment, reflected in the increasing average reward. The graph suggests that over time the policy improves, evident from the upward trend in average reward and a convergence towards less variability between the maximum and minimum rewards.





Impact of Hyperparameters:

In Q-Learning, the learning rate and discount factor are pivotal for successful learning. A higher learning rate might lead to rapid learning but could also cause the agent to overshoot optimal values, while a lower learning rate might result in slow convergence. The discount factor affects the agent's foresight, with a higher value making it more far-sighted.

The decaying ε plays a critical role in balancing exploration and exploitation. Initially high, it allows the agent to explore various actions and gather diverse experiences. As ε decays, the agent increasingly relies on its accumulated knowledge, exploiting the strategy it has learned to be most effective.

The Q-Learning approach in this case is limited by the discretization of the continuous space, which can lead to a loss of information and suboptimal policies. Moreover, for complex environments like Pendulum-v1, Q-Learning can struggle to find the optimal policy within a reasonable number of episodes due to the sparse and delayed rewards

structure, as the pendulum must first be swung with enough momentum to achieve the upright position.

The Q-Learning approach, as seen in the graph, suggests the agent does learn to increase the reward over time. However, due to the discretization of the action space, the performance may not reach the optimal policy that a continuous control algorithm could achieve.

Fixed Q-Learning for Pendulum-v1

Implementation:

The Fixed Q-Learning approach takes the classic Q-Learning algorithm and enhances its stability by using a secondary Q-table, known as the target Q-table, to calculate the future Q-value used in the update equation. This modification is intended to address the inherent instability of Q-Learning, which can arise when the Q-values are frequently updated during training, sometimes referred to as the "moving target" problem.

2.1 Capture and Sampling of the Data:

In the Fixed Q-Learning code for Pendulum-v1, data capture is similar to that in Q-Learning, where the state, action, reward, and next state are recorded at each step of the episode. The actions are selected using an ε-greedy policy, which balances exploration and exploitation by choosing either the best-known action according to the Q-table or a random action with the probability of random action selection decaying over time as the agent learns more about the environment.

2.2 Network Structure and Hyperparameters:

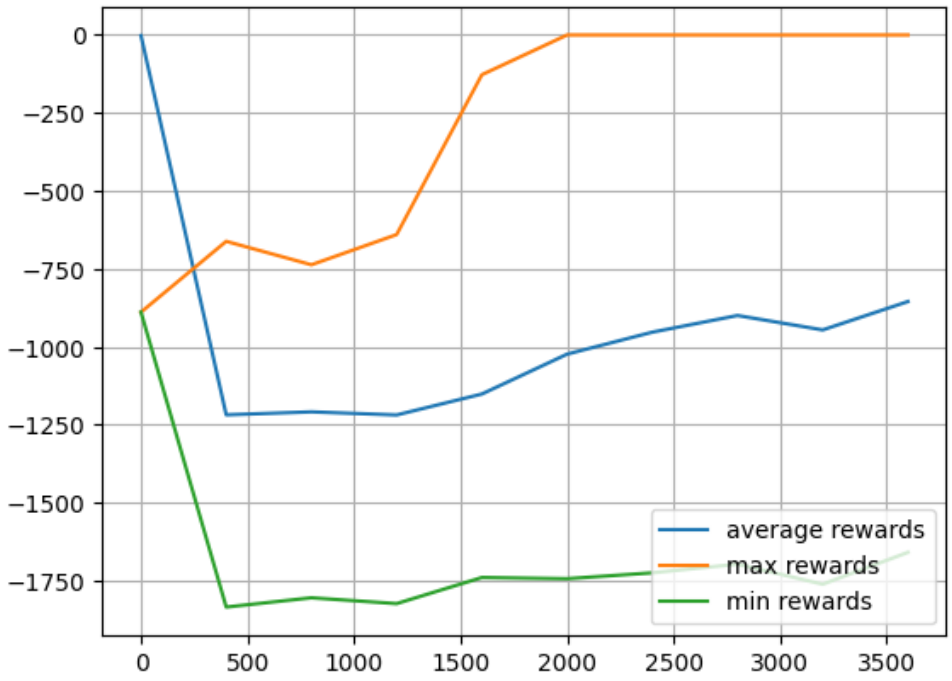
Although Fixed Q-Learning still does not use a neural network, the presence of the target Q-table introduces an additional hyperparameter: the frequency at which the target Q-table is updated from the live Q-table, controlled by `UPDATE_TARGET_EVERY`. Other critical hyperparameters include the learning rate, discount factor, and the strategy for decaying epsilon.

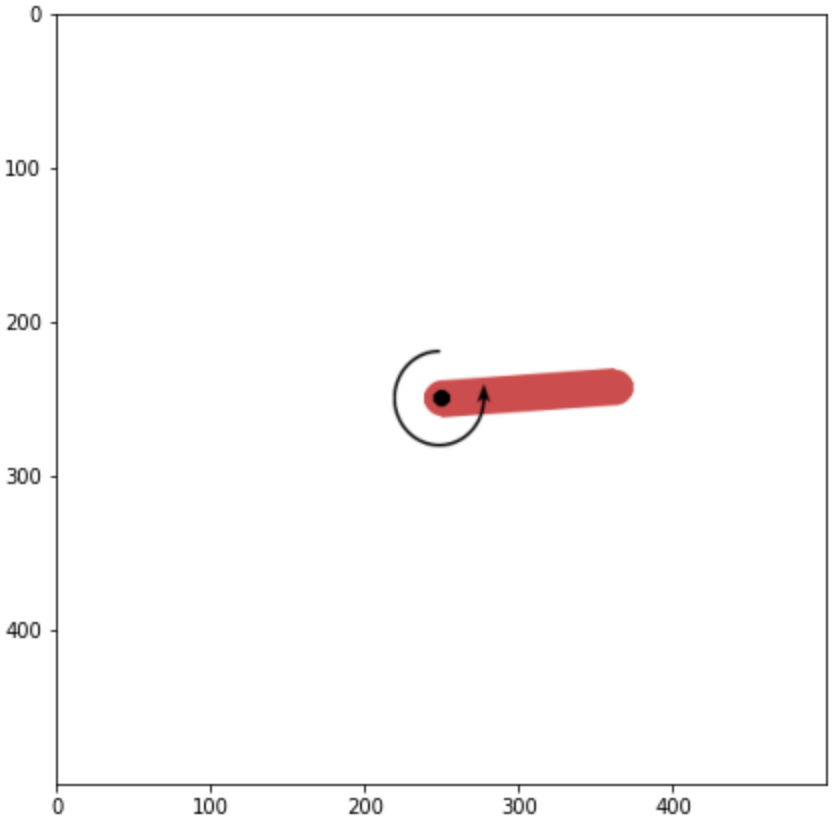
2.3 Q-Learning Update on Weights:

The Q-table update mechanism in Fixed Q-Learning uses the Bellman equation, like in standard Q-Learning, but with the critical difference that the future Q-value used in the update is taken from the target Q-table. The target Q-table itself is only updated to the live Q-table values every `UPDATE_TARGET_EVERY` episodes, which is supposed to provide a stable set of Q-values that do not fluctuate as drastically with each individual update.

Evaluation of Results:

The graph reflects the agent's learning progress across episodes. The average reward, which represents the average score over all episodes, appears to improve over time, indicating that the agent is learning to swing the pendulum closer to the upright position with less effort. The presence of maximum and minimum rewards offers insights into the consistency of the agent's performance.





Impact of Hyperparameters:

The introduction of the target Q-table and its update frequency is pivotal in stabilizing the learning process. The learning rate influences the magnitude of the Q-value updates, while the discount factor affects how future rewards are valued. The decaying epsilon ensures that as the agent becomes more experienced, it relies increasingly on exploitation rather than exploration to make decisions.

The graph associated with Fixed Q-Learning is expected to show a less volatile progression compared to standard Q-Learning, as the agent benefits from the more stable target Q-values during training. This could be evidenced by a more consistent increase in average rewards and a reduction in the spread between the maximum and minimum rewards over time.

The Fixed Q-Learning approach tackles the continuous control task of the Pendulum-v1 environment by discretizing the space and applying a more stable update mechanism. However, despite the improvements over standard Q-Learning, Fixed Q-Learning might still fall short of capturing the full complexity of the environment, which could potentially be better addressed with a function approximation method like Deep Q-Networks (DQN).

DQN for Pendulum-v1

Implementation:

In the provided DQN implementation, a neural network architecture with two hidden layers of 32 neurons each and 'leaky_relu' activation functions is used. This network is responsible for approximating the Q-value function for the continuous state space of the Pendulum-v1 environment. The output layer uses a 'tanh' activation function to scale the actions in the range [-2, 2], matching the action space of the environment.

2.1 Capture and Sampling of the Data:

The replay buffer is a critical component of DQN that stores the agent's experiences, which are then sampled randomly to train the network. This mechanism helps in breaking the temporal correlations in the sequential data and stabilizes the learning process.

2.2 Network Structure and Hyperparameters:

The model's structure with 32 neurons in each hidden layer is relatively simple but sufficient to capture the complexity of the Pendulum-v1 environment. The learning rate is set at 0.01, and the discount factor at 0.95. The optimizer used is Nadam, a variant of Adam with Nesterov momentum, which is more responsive to changing landscapes in the loss function.

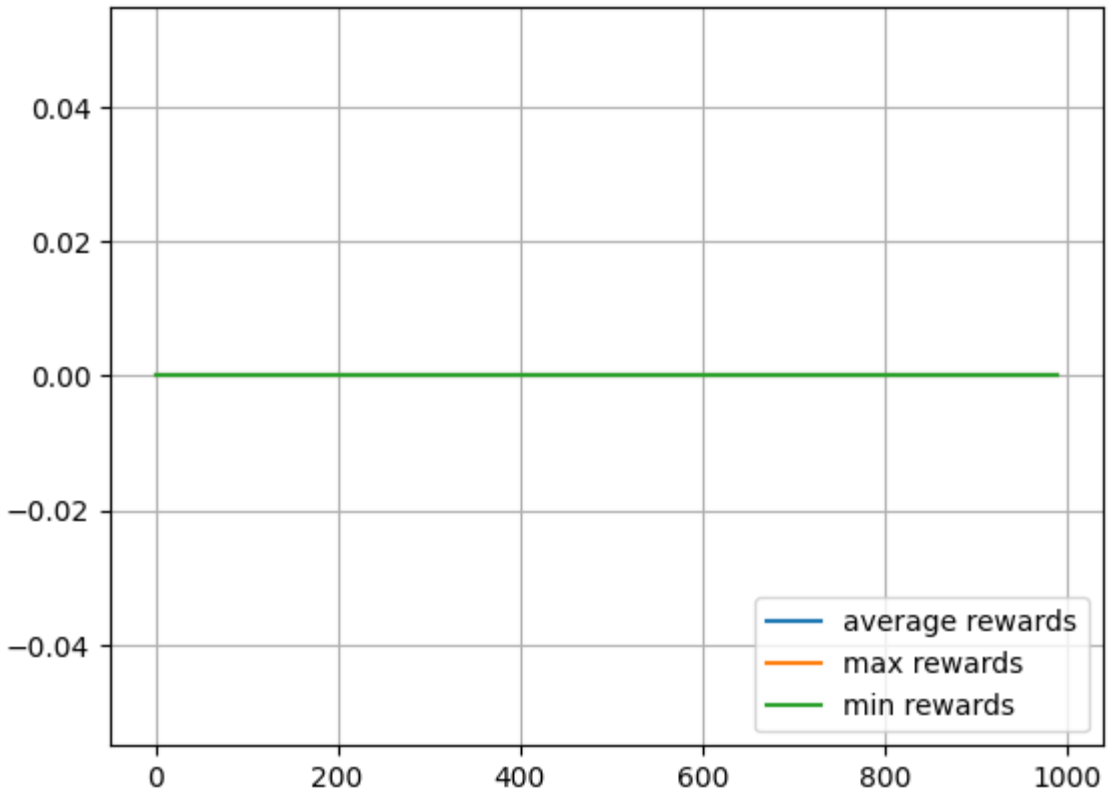
2.3 Q-Learning Update on Weights:

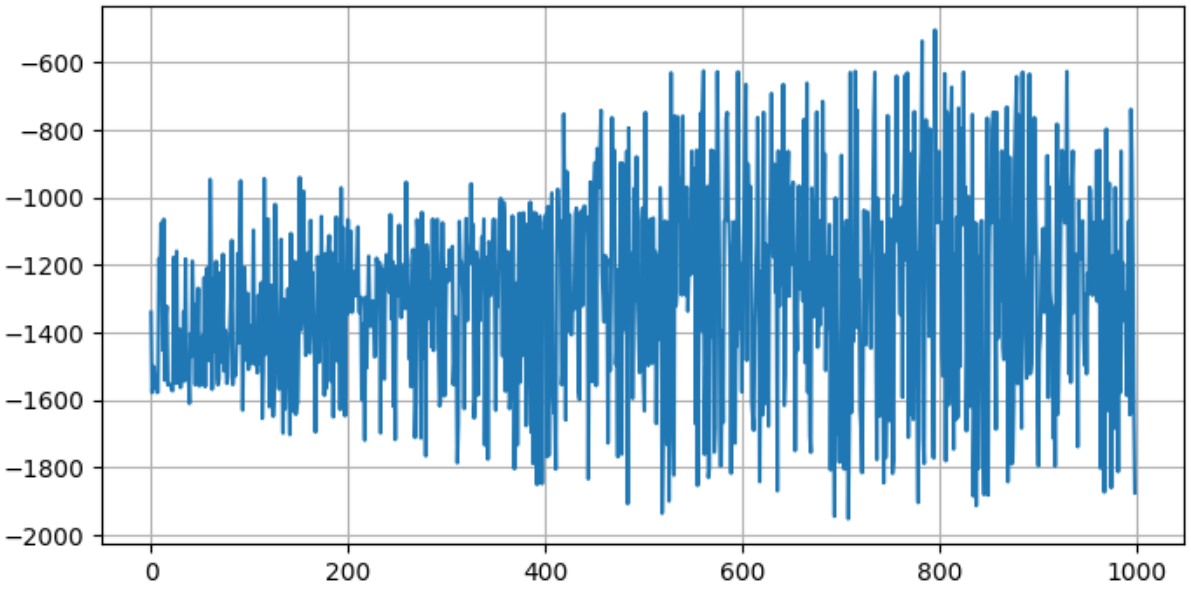
During the training step, the DQN updates the weights by first predicting the next Q-values with the network and then calculating the targets, which consist of the immediate rewards

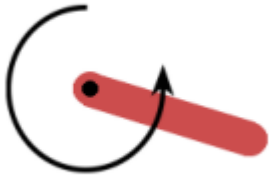
and the discounted max future Q-values from the model's predictions. The loss is the mean squared error between the target Q-values and the predicted Q-values from the actions taken. Gradients are then computed and applied to update the model weights.

Evaluation of Results:

The attached graphs display the agent's learning progress across episodes. The first graph plots average, maximum, and minimum rewards per episode, while the second graph shows the total reward per episode. A steady learning curve or an increase in the average reward over episodes would indicate that the agent is effectively learning to balance the pendulum.







Impact of Hyperparameters:

The learning rate determines the step size in updating the weights, which can affect the convergence speed and stability. The number of neurons affects the model's capacity to approximate the complex relationship between states and actions. A model with too few neurons may not capture the complexity, while too many neurons could lead to overfitting. The decaying epsilon impacts exploration, encouraging the agent to explore various actions initially and gradually shift to exploiting the learned strategy as it becomes more confident.

The first graph for DQN typically shows the progression of learning across episodes, and the average reward line should ideally show an upward trend as the network learns the task. However, the second graph reveals high variability in the total rewards, indicating that while the agent might be improving on average, there is still significant fluctuation in its performance from one episode to the next.

The use of a neural network allows DQN to learn and generalize from a continuous space efficiently, addressing the challenge posed by the Pendulum-v1 environment. However, as seen in the graph, the performance of DQN can be variable, and achieving consistent control of the pendulum may require further tuning of the hyperparameters or the network architecture.

References

- Brockman, G., et al. (2016). OpenAI Gym. *arXiv preprint arXiv:1606.01540*.

- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

- Mnih, V., et al.

(2015). Human-level control through deep reinforcement learning. *Nature*, 518, 529–533.

- Sutton, R. S., and Barto, A. G. (2018). *Reinforcement Learning: An Introduction*. MIT Press.

- Watkins, C. J. C. H. (1989). Learning from delayed rewards. *Ph.D. dissertation*, Cambridge University.

Code References:

https://pythonprogramming.net/q-learning-reinforcement-learning-python-tutorial/

https://ymd_h.gitlab.io/ymd_blog/posts/gym_on_google_colab_with_gnwrapper/

https://github.com/ageron/handson-ml2/blob/master/18_reinforcement_learning.ipynb

https://gist.github.com/korakot/86f152571f15e921a16af6f916081f40

https://stackoverflow.com/questions/61110188/how-to-display-a-gif-in-jupyter-notebook-using-google-colab