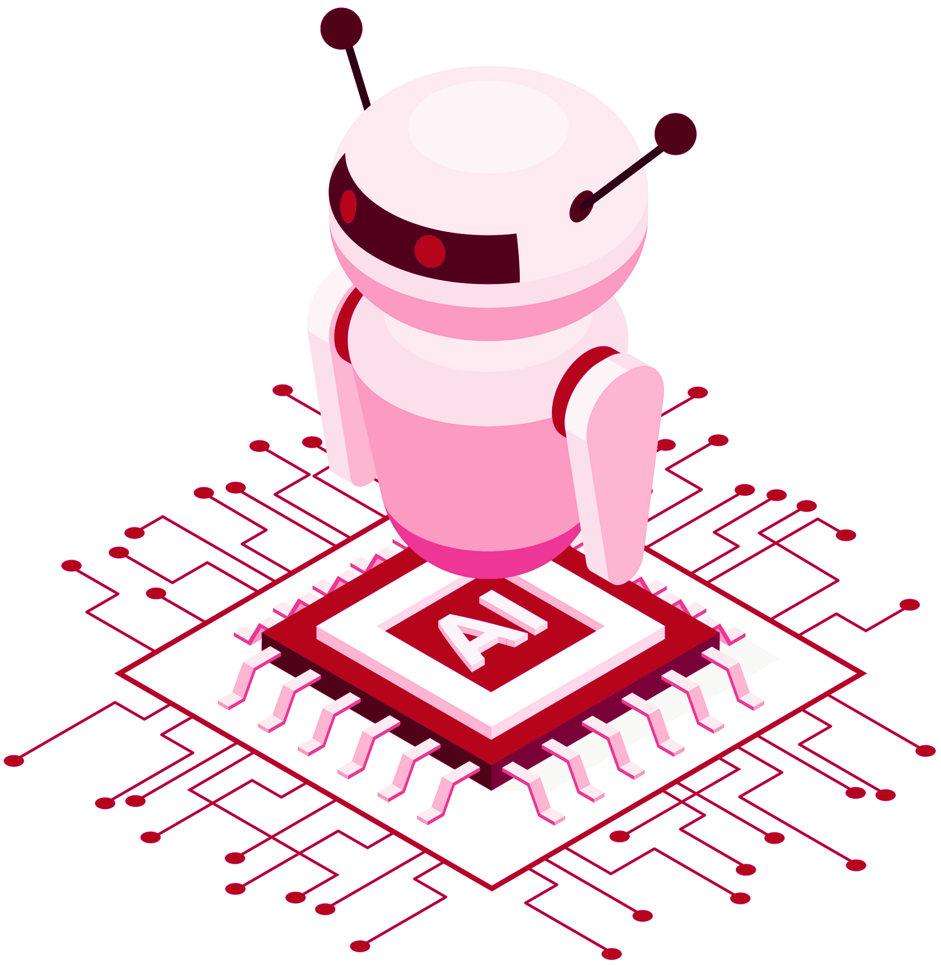


Deep Reinforcement Learning

Professor Mohammad Hossein Rohban

Homework 2:

Value-Based Methods

By:

Arian komaei

403206373

Spring 2025 w

Contents

Grading

The grading will be based on the following criteria, with a total of 100 points:

Task	**Points**
Task 1: Epsilon Greedy & N-step Sarsa/Q-learning	40
Jupyter Notebook	25
Analysis and Deduction	15
Task 2: DQN vs. DDQN	50
Jupyter Notebook	30
Analysis and Deduction	20
Clarity and Quality of Code	5
Clarity and Quality of Report	5
Bonus 1: Writing your report in Latex	10

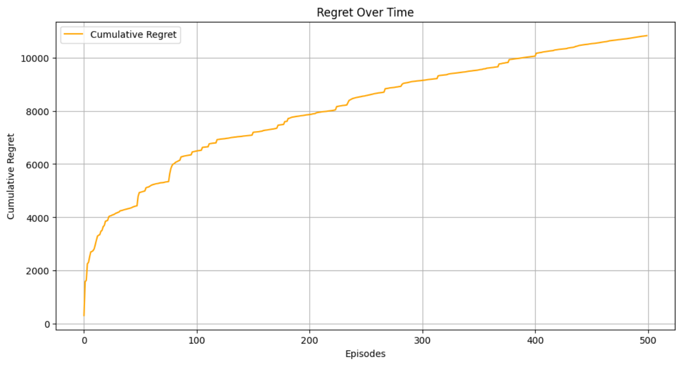
Notes:

- Include well-commented code and relevant plots in your notebook.
- Clearly present all comparisons and analyses in your report.
- Ensure reproducibility by specifying all dependencies and configurations.

1 Epsilon Greedy

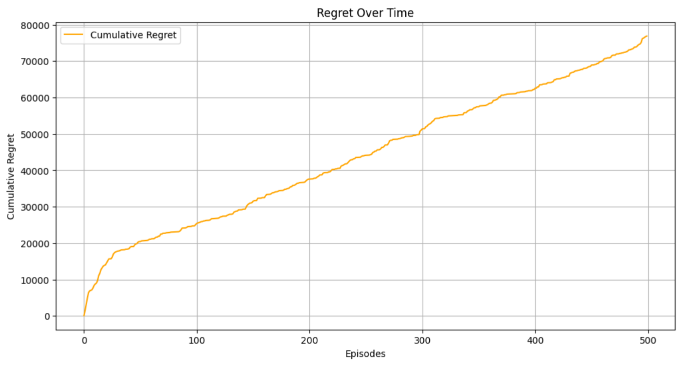
1.1 Epsilon 0.1 initially has a high regret rate but decreases quickly. Why is that? [2.5-points]

Because the agent explores the world only 10% of the time and it quickly converges to the near optimal values which also allows the algorithm to find the near best values early while still refining choices.



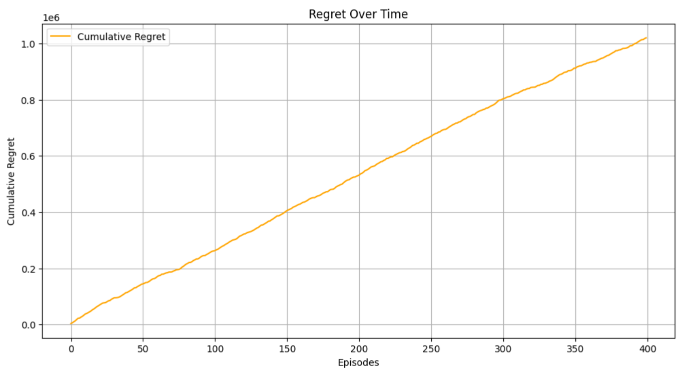
1.2 Both epsilon 0.1 and 0.5 show jumps. What is the reason for this? [2.5-points]

Since eps-greedy explores with probability eps, these occasional suboptimal choices cause sudden increases in regret.



1.3 Epsilon 0.9 changes linearly. Why? [2.5-points]

For eps=0.9, the regret increases linearly because the algorithm explores 90% of the time, frequently choosing random actions, including suboptimal ones. Since it rarely exploits the best action, it accumulates regret at a nearly constant rate, leading to a linear growth pattern.

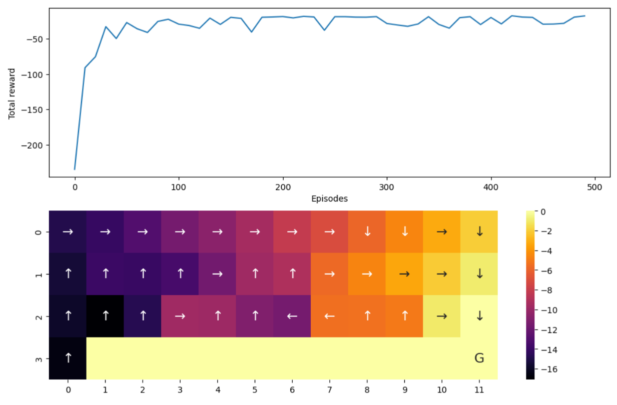


1.4 Compare the policy for epsilon values 0.1 and 0.9. How do they differ, and why do they look different? [2.5-points]

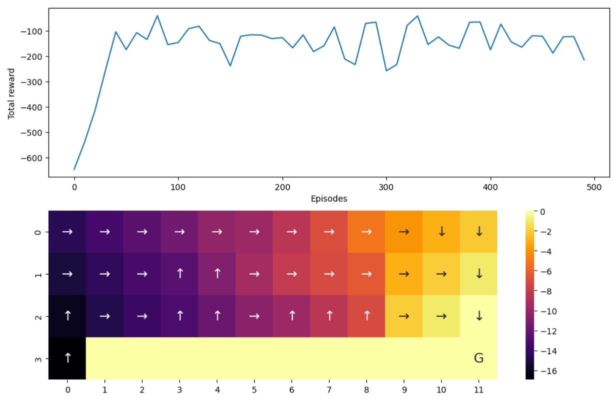
With epsilon = 0.9, the policy is mostly exploratory, choosing random actions 90% of the time and exploiting 10%.

With epsilon = 0.1, the policy is mostly exploitative, meaning the agent chooses the best-known actions 90% of the time and explores 10% of the time.

This leads to faster convergence to the optimal path from the start to the goal. However, the downside is that we don't fully explore all possible states and their associated rewards. As seen in the 0.1 epsilon case, some areas show poor directions toward the goal. In contrast, with an epsilon of 0.9, we explore the environment more thoroughly, which allows for better calculations of action values and, ultimately, a more optimal strategy.



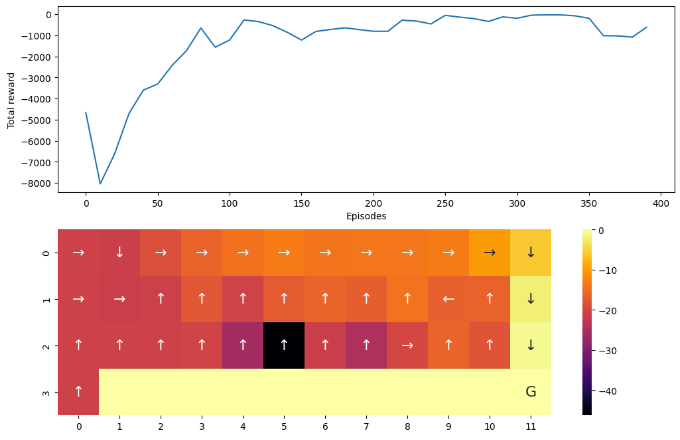
0.1 policy



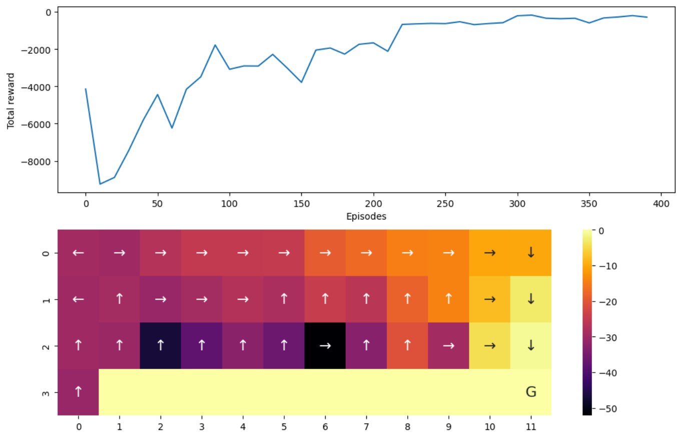
0.9 policy

1.5 In the epsilon decay section, analyze the optimal policy for the row adjacent to the cliff (the lowest row). Then, compare the different learned policies and their corresponding rewards. [2.5-points]

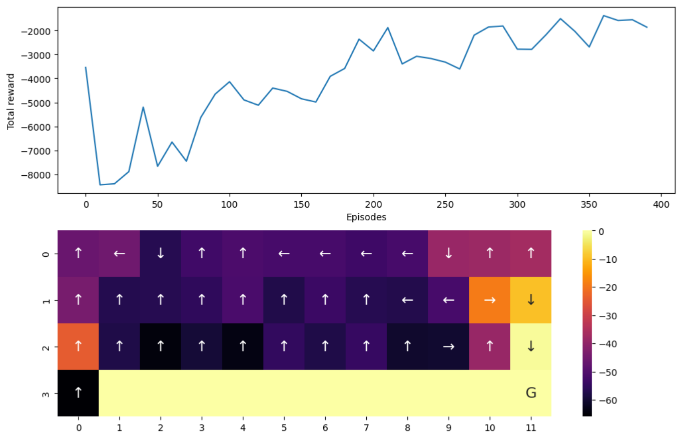
This environment discourages exploration, so reducing epsilon more quickly during the learning process leads to better results and faster convergence. As observed, a rapid decay in epsilon yields superior performance. Interestingly, due to the prolonged high exploration in the slow decay algorithm, the rewards near the cliff are significantly lower compared to the fast decay approach. As we can see, fast decay achieves better convergence due to its lower exploration rate.



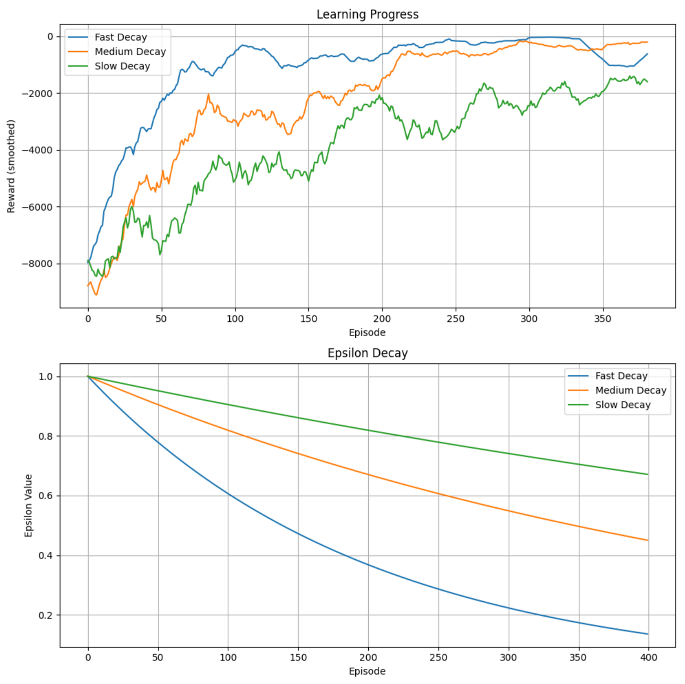
Fast decay



Medium decay



Slow decay



2 N-step Sarsa and N-step Q-learning

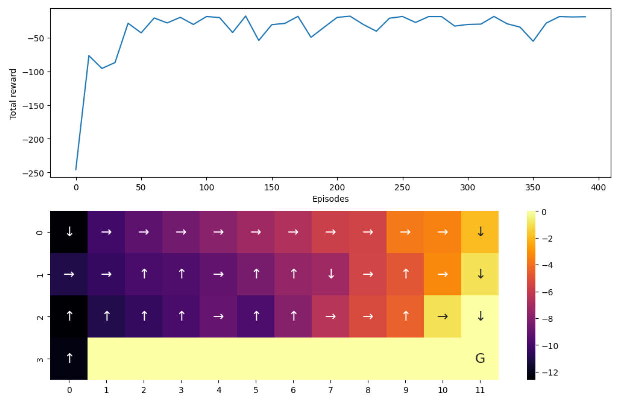
2.1 What is the difference between Q-learning and sarsa? [2.5-points]

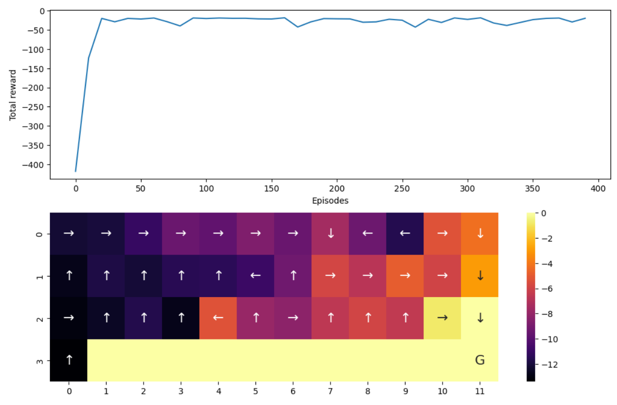
They are both RL algorithm but they differ in the way that they update their Q values. Q learning is an off policy algorithm which update Q-values using the maximum possible future reward. It is a more aggressive and explores more. On the contrary, Sarsa is an on-policy algorithm which updates the Q-value based on the actual action taken by the agent. It is a more conservative and stable algorithm compared to Q-learning.

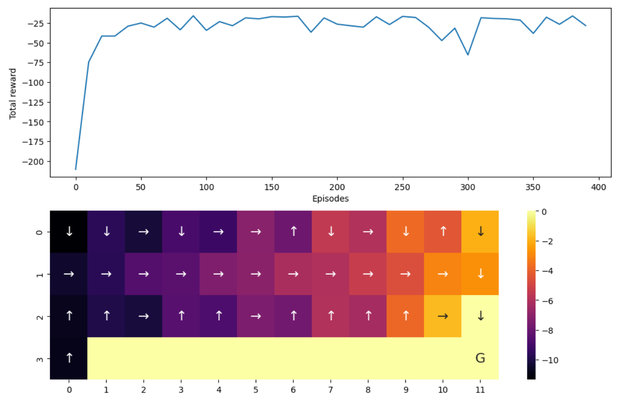
2.2 Compare how different values of n affect each algorithm's performance separately. [2.5-points]

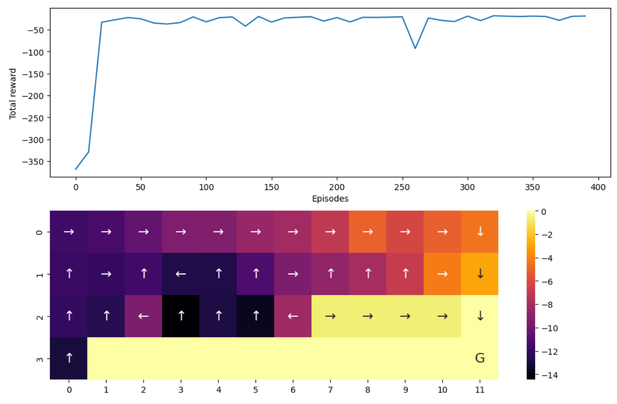
The choice of n in n-step Q-learning and n-step SARSA significantly impacts how future rewards are incorporated into value updates. A small n leads to faster and more stable updates but tends to prioritize short-term rewards, which can result in suboptimal decisions. In contrast, a larger n considers more future rewards, enhancing long-term decision-making. However, this comes at the cost of increased variance and potential instability in learning.

Q-learning (off-policy) tends to learn faster but can become unstable when n is large, whereas SARSA (on-policy) is generally more stable but may take longer to converge to the optimal policy. Increasing n accelerates convergence to optimal values, as illustrated in the figure below. However, a higher n also introduces greater variance, which may impact stability depending on the learning environment.


n=1 sarsa


n=5 sarsa


n=1 Q-learning


n=5 Q-learning

2.3 Is a Higher or Lower n Always Better? Explain the advantages and disadvantages of both low and high n values. [2.5-points]

There is no always better choice between high and low n. It depends on the situation. Advantages of having a small n are that the agent learns faster and updates quickly and is more stable with lower randomness (variance). The disadvantages are that it focuses too much on short-term rewards and may miss better long-term strategies. On the other hand, higher n considers long-term rewards, leading to better decisions and reduces bias in learning but it leads to Slower learning and updates more unstable due to higher randomness

3 DQN vs. DDQN

3.1 Which algorithm performs better and why? [3-points]

DDQN performs better than DQN because it reduces overestimation bias in Q-learning. DQN tends to overestimate Q-values, leading to suboptimal policies. DDQN decouples action selection and evaluation by using the online network to select actions and the target network to evaluate them. This correction leads to more stable learning and better policy performance.DDQN improves accuracy, stability, and policy quality over DQN, making it a better choice for this task

3.2 Which algorithm has a tighter upper and lower bound for rewards. [2-points]

DDQN generally has a tighter upper and lower bound for rewards compared to DQN.DQN is an improvement over traditional Q-learning, using deep neural networks to approximate the Q-value function. However, it suffers from an overestimation bias in the Q-values. This is because DQN uses the same Q-network to both select and evaluate the next action, which can lead to overestimates of the Q-values, especially in noisy environments.DDQN was proposed to address the overestimation bias present in DQN. In DDQN, two networks are used. One to select the action (like in DQN), and another target network to evaluate that action. This reduces the overestimation bias and helps provide more accurate Q-values. By mitigating the overestimation bias, DDQN tends to produce Q-values that are more accurate, which leads to more stable and reliable training. This in turn results in a tighter bound for rewards because the agent's value estimation is more consistent and less prone to extreme values.

3.3 Based on your previous answer, can we conclude that this algorithm exhibits greater stability in learning? Explain your reasoning. [2-points]

Yes we can conclude that DDQN has greater stability compared to DQN.

One of the main challenges with DQN is its overestimation bias. Since DQN uses the same Q-network to both select the next action and evaluate the Q-value for that action, it can overestimate the expected future rewards. Overestimations can lead to suboptimal actions being chosen, causing the agent to make poor decisions and, ultimately, destabilizing the learning process. This is especially problematic in environments with noisy or complex state-action spaces, where overestimates can accumulate over time and drastically affect performance. DDQN was specifically designed to address this issue. By using two separate networks — one for selecting the action (the current Q-network) and another for evaluating the chosen action (the target Q-network) — DDQN reduces the likelihood of overestimating Q-values. This decoupling ensures that the selected action is evaluated more accurately, leading to more stable updates and better overall policy learning.
With reduced overestimation bias, DDQN is better able to accurately approximate the true Q-values for state-action pairs. This leads to more reliable training because the agent's decisions are based on a clearer understanding of the environment. In contrast, the overestimations in DQN can cause erratic learning, with the agent potentially diverging or converging to suboptimal solutions due to incorrect value estimates.

3.4 What are the general issues with DQN? [2-points]

DQN (Deep Q-Network) has several general issues. First is its instability. Training can be very unstable due to correlations in data and high variance in updates. Secondly it is very sample Inefficient. It requires a lot of interactions with the environment to learn effectively. Also it can get stuck in suboptimal policies because it often explores too little. We can mention that it has very limited generalization and it struggles with generalizing across different tasks or environments. These issues have led to improvements like Double DQN.

3.5 How can some of these issues be mitigated? (You may refer to external sources such as research papers and blog posts be sure to cite them properly.) [3-points]

1. Instability

- DQN uses a target network to stabilize training by periodically updating the target Q-values. This reduces the risk of oscillations and divergence in Q-value updates.
- One solution is to use double DQN. This algorithm improves this further by using the main network to select actions and the target network to estimate the Q-values, which reduces the overestimation bias and stabilizes training.

Paper: Hasselt, H. V. (2016). *Double Q-learning*. https://arxiv.org/abs/1509.06461

2. Sample Inefficiency

- Instead of sampling transitions uniformly from the replay buffer, this method prioritizes experiences that have higher temporal-difference (TD) errors, allowing the agent to learn from important experiences more frequently.
- Using techniques like **Noisy Nets** or **Bootstrapped DQN** can encourage better exploration by introducing noise into the network weights or bootstrapping estimates across multiple Q-functions.

Paper: Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). *Prioritized Experience Replay*. https://arxiv.org/abs/1511.05952

Paper: Fortunato, M., Azizzadenesheli, K., & et al. (2017). *Noisy Networks for Exploration*. https://arxiv.org/abs/1706.10295

3. Poor Exploration

- Instead of relying on epsilon-greedy exploration (which decays over time), methods like Noisy Networks or Bootstrapped DQN help maintain exploration by adding noise to the action selection process.
- Adding intrinsic motivation to drive exploration when the agent encounters unfamiliar states (e.g., the Intrinsic Curiosity Module, ICML 2017) can further improve exploration.

Paper: Pathak, D., Agrawal, P., Efros, A. A., & Darrell, T. (2017). *Curiosity-driven Exploration by Self-supervised Prediction*. https://arxiv.org/abs/1705.05363

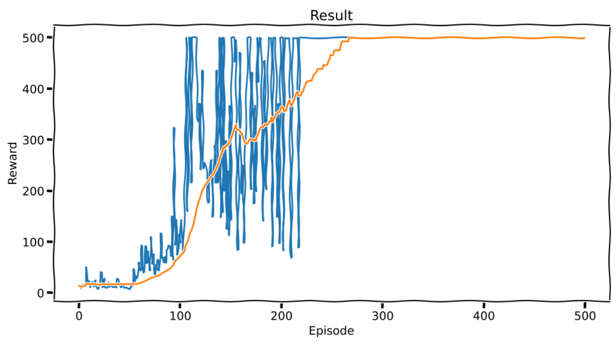
4. Limited Generalization

- To improve generalization, training environments can be randomized (e.g., varying physics parameters or textures), encouraging the agent to learn more robust policies that generalize better to new environments.
- Techniques like MAML (Model-Agnostic Meta-Learning) enable the agent to quickly adapt to new tasks with few interactions, improving its generalization ability.

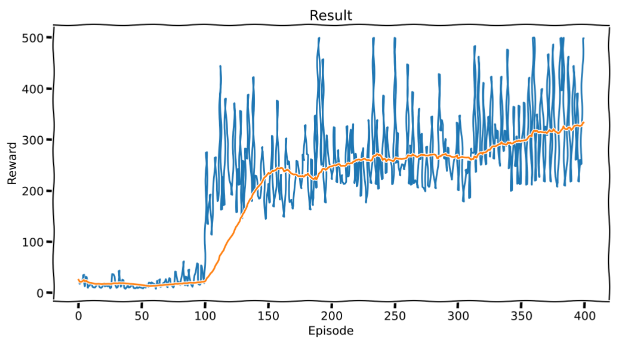
Paper: Finn, C., Abbeel, P., & Levine, S. (2017). *Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks*. https://arxiv.org/abs/1703.03400

3.6 Based on the plotted values in the notebook, can the main purpose of DDQN be observed in the results? [2-points]

we can see that DQN is very unstable in its learning process and there is a good chance that the model does not converges to the optimal prediction. However DDQN has more stable track.



DQN policy



DDQN

3.7 The DDQN paper states that different environments influence the algorithm in various ways. Explain these characteristics (e.g., complexity, dynamics of the environment) and their impact on DDQN's performance. Then, compare them to the CartPole environment. Does CartPole exhibit these characteristics or not? [4-points]

The DDQN paper highlights how different environments affect the algorithm:
Complexity: Environments with large state and action spaces are harder for DDQN due to increased exploration and slower convergence.
Dynamics: Stochastic environments, where outcomes vary, make learning more difficult for DDQN because of the uncertainty in transitions.
Reward Structure: Sparse rewards make it harder for DDQN to learn, while dense rewards speed up learning.
Exploration vs. Exploitation: Balancing exploration and exploitation is crucial, especially in complex or dynamic environments.

Continuous vs. Discrete States: Continuous state spaces require more sophisticated function approximators and can challenge DDQN.

CartPole Comparison:
Complexity: CartPole is low-complexity (small state and action spaces).
Dynamics: CartPole is deterministic (no randomness), making it easier for DDQN.
Reward Structure: CartPole has dense rewards, aiding faster learning.
Exploration vs. Exploitation: Exploration is easier due to simplicity.
Continuous States: CartPole has continuous states, but they're manageable.

3.8 How do you think DQN can be further improved? (This question is for your own analysis, but you may refer to external sources such as research papers and blog posts be sure to cite them properly.) [2-points]

Noisy Nets for Exploration
One limitation of DQN and many reinforcement learning algorithms is the exploration-exploitation tradeoff. DQN typically uses ϵ-greedy exploration, where random actions are chosen with probability ϵ. *Noisy Networks* (Fortunato et al., 2017) provide a more sophisticated approach by injecting noise into the weights of the neural network itself, encouraging the model to explore more effectively without the need for an explicit exploration parameter like ϵ. This can lead to better exploration and improved long-term performance.
Reference: Fortunato, M., Azizzadenesheli, K., Liao, L., et al. (2017). Noisy Networks for Exploration. *ICML*.

Rainbow DQN (Combining Multiple Enhancements)

A notable improvement over basic DQN is *Rainbow DQN*, which combines several of the advancements mentioned above, including Double Q-learning, Prioritized Experience Replay, Dueling DQN, and Noisy Networks, among others. Rainbow DQN offers a state-of-the-art combination of techniques and has demonstrated superior performance across a variety of Atari games (Hessel et al., 2017).

Reference: Hessel, M., Modayil, J., Van Hasselt, H., et al. (2017). Rainbow: Combining Improvements in Deep Reinforcement Learning. *AAAI*.

References

[1] R. Sutton and A. Barto, Reinforcement Learning: An Introduction, 2nd Edition, 2020. Available: http://incompleteideas.net/book/the-book-2nd.html.

[2] Gymnasium Documentation. Available: https://gymnasium.farama.org/

[3] Grokking Deep Reinforcement Learning. Available: https://www.manning.com/books/grokking-deep-reinforcement-learning

[4] Deep Reinforcement Learning with Double Q-learning. Available: https://arxiv.org/abs/1509.06461

[5] Cover image designed by freepik