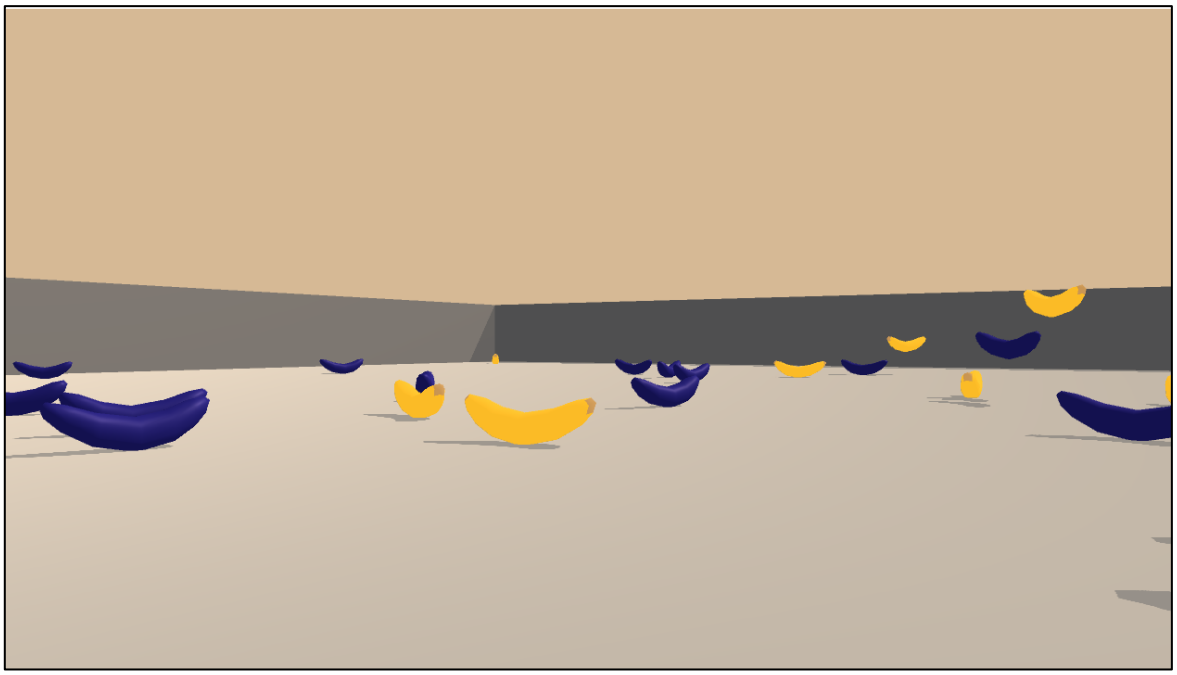
Deep Reinforcement Learning

Project 1: Navigation



Simon FERRAND

School : Udacity

Course name : Deep Reinforcement Learning Nanodegree

13/01/24

UDACITY

Abstract

This report outlines the implementation and results of a Deep Q-Network (DQN) applied to the Navigation problem from the Unity ML-Agents environment. The goal is to train an agent to navigate a space and collect yellow bananas while avoiding blue ones. The DQN agent was successfully trained to achieve an average score of +13 over 100 consecutive episodes, indicating satisfactory learning and problem-solving within the environment.

Introduction

The Navigation project presents a common reinforcement learning problem where an agent is expected to learn an optimal policy for navigating a space with rewards and penalties. The agent must maximize its score by collecting yellow bananas and avoiding blue ones within a Unity-based environment. This task is a stepping stone towards more complex decision-making problems that DRL agents can solve.

1) Environment Description

The simulation consists of a square world where bananas are randomly placed. The agent receives a reward of +1 for collecting a yellow banana and a reward of -1 for collecting a blue banana. Thus, the goal of the agent is to collect as many yellow bananas as possible while avoiding blue bananas.

The state space has 37 dimensions, including the agent's velocity and perception of objects around its forward direction. The action space has four discrete actions:

- 0 - move forward.
- 1 - move backward.
- 2 - turn left.
- 3 - turn right.

The task is episodic, and to solve the environment, the agent must get an average score of +13 over 100 consecutive episodes.

2) Methodology

2.1) Learning Algorithm

The agent employs a DQN, an algorithm that stabilizes training of Q-learning by utilizing a replay buffer and a target network. The replay buffer stores experiences that are randomly sampled, reducing the correlation in the observation sequence. The target network's slower updates provide more stable targets for learning.

2.2) Network Architecture

The neural network acting as the function approximator for our Q-values consists of three fully connected layers:

- First layer: 64 units with ReLU activation.
- Second layer: 128 units with ReLU activation.
- Third layer: 64 units with ReLU activation.
- Output layer: 4 units representing the action values.

2.3) Hyperparameters

The following hyperparameters were used for the DQN:

- Replay buffer size: 1e5
- Batch size: 64
- Gamma (discount factor): 0.99
- Tau (for soft update of target parameters): 5e-4
- Learning rate: 5e-4
- Update frequency: Every 4 steps

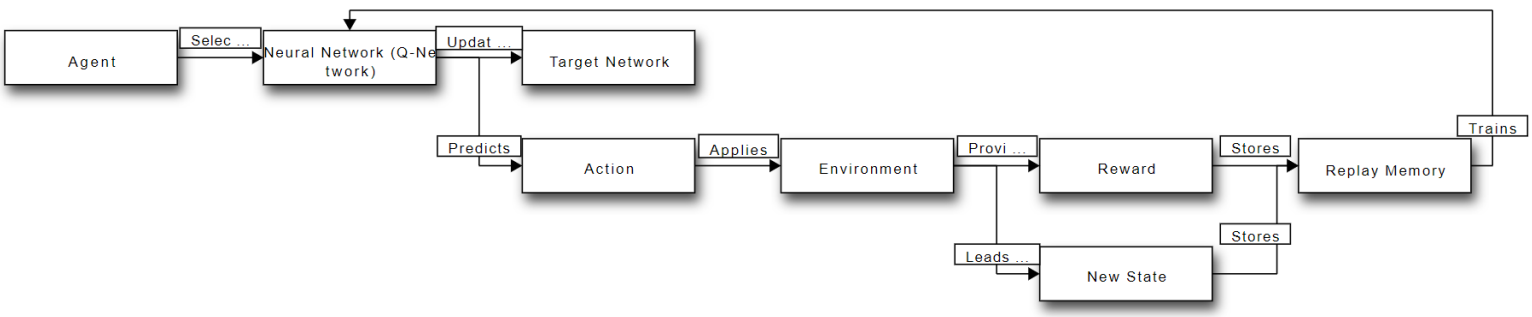
The agent's exploration-exploitation trade-off was managed using epsilon-greedy action selection, with epsilon starting at 1.0 and decaying by a factor of 0.995 each episode, to a minimum of 0.01.

3) Implementation Details

The implementation involved setting up a reinforcement learning pipeline that included initializing the environment, agent, and training loop. The agent was instantiated with the defined neural network architecture and the specified hyperparameters.

During training, the agent interacted with the environment in discrete time steps. At each step, it chose an action based on the current policy, observed the next state and reward, and stored this experience. Periodically, the agent sampled from the stored experiences to update the policy network using stochastic gradient descent.

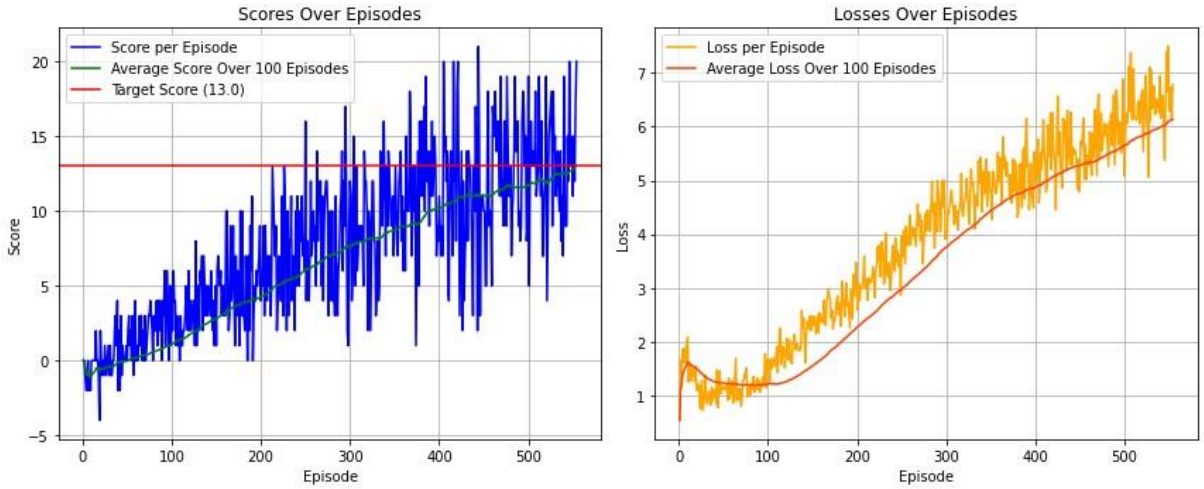
To promote stability, a separate target network, with the same architecture but with slowly updated weights, was used to generate the Q-value targets for the loss computation.



4) Results

4.1) Training Performance

The agent demonstrated learning over time, as shown by the increasing score per episode. The average score plot over 100 episodes illustrates the agent's improvement and its variability in performance due to exploration.



4.2) Losses

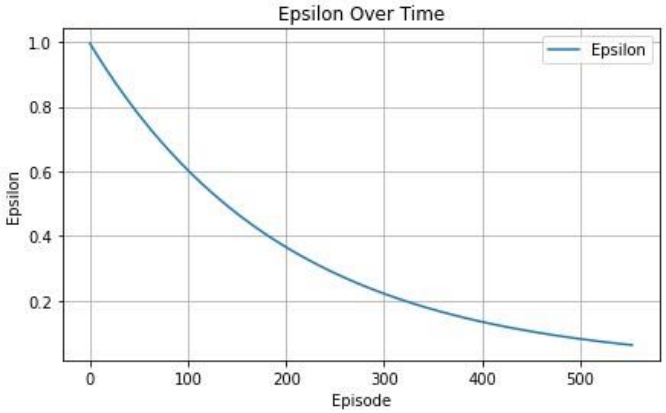
An intriguing observation during training was the incremental increase in the loss per episode. This could raise questions about the stability and convergence of the learning process. While a certain degree of fluctuation in loss is expected due to the exploratory actions of the agent and the evolving policy, a consistent upward trend requires careful analysis. It could suggest that the agent is struggling to assimilate new experiences into its existing knowledge base, or that the complexity of the environment is presenting challenges that are not easily mitigated by the current network architecture and hyperparameters.

The increasing loss also points towards the possibility of the agent encountering more complex scenarios as it navigates the environment, which may not be fully captured by its current state representation or action-value estimation. This warrants a closer look into aspects such as the replay buffer's composition, the sufficiency of the exploration strategy, and the adequacy of the network's capacity to generalize from the observed states.

To address these issues and enhance agent performance, potential steps include hyperparameter optimization, architectural adjustments, or the incorporation of techniques like Double DQN, Dueling DQN, or Prioritized Experience Replay, which are designed to provide more stable and efficient learning.

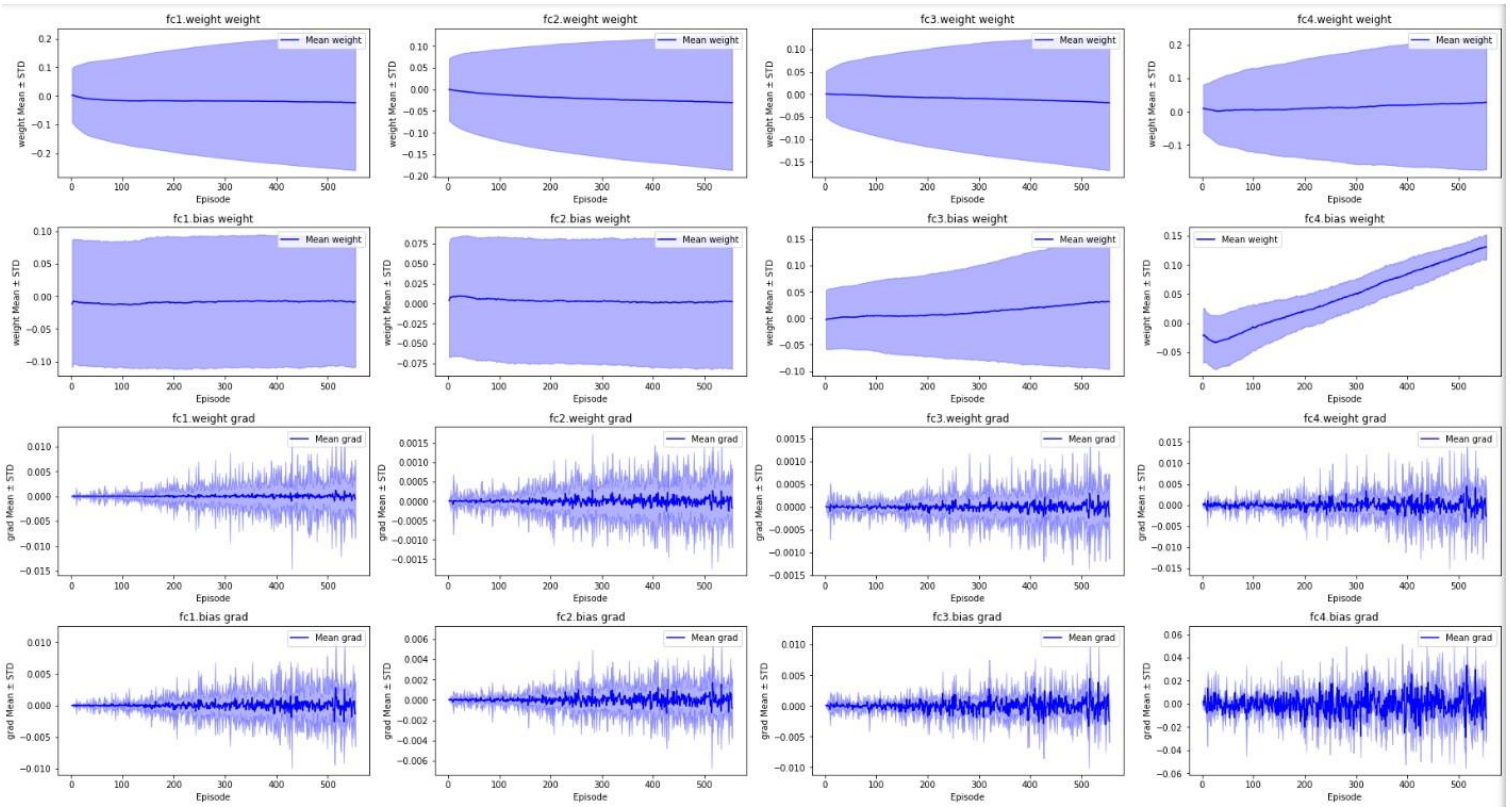
4.3) Epsilon Decay

The epsilon value decayed from 1.0 to 0.01 as per the schedule, balancing the need for exploration versus exploitation throughout the learning process.



4.4) Weights and Gradients

The mean and standard deviation of the weights and gradients of each layer of the neural network were tracked. This monitoring can diagnose issues with learning such as vanishing or exploding gradients.



Throughout the training process, we meticulously monitored the mean and standard deviation (STD) of both the weights and gradients across all layers in the neural network. This kind of scrutiny is crucial as it helps identify potential learning issues, such as vanishing or exploding gradients.

From the observed data, a notable trend was the gradual increase in the standard deviation of the bias weights, particularly visible in the later layers of the network (e.g., fc4.bias weight). This increment in variability could be symptomatic of instability in the learning process, potentially leading to erratic

policy behavior. While an increasing trend in STD might not always be problematic, it warrants a closer examination to ensure that the network's parameters are converging appropriately.

Furthermore, the gradients remained relatively stable without showing signs of diminishing or excessively amplifying, which is a positive indication that the backpropagation is effectively updating the weights throughout the training. The consistency in gradient values across episodes suggests that the learning rate and update frequency were adequately chosen to maintain steady progress without causing instability in the weights' updates.

4.5) Test Performance

Successful loading of the trained model was evidenced by the agent's consistent performance over 10 test runs, achieving an average score of 13.6 with a standard deviation of 2.5. These results confirm the agent's ability to generalize its training to new scenarios, surpassing the required score threshold.

5) Discussion

The results indicate that the DQN agent learned an effective policy for the navigation task. The training process showed the expected decrease in epsilon, signifying a shift from exploration to exploitation. However, the increasing trend in loss suggests that the agent continued to adjust its policy significantly throughout training, which could be due to the complexity of the environment or insufficient training episodes to fully converge.

The variability in the score per episode reflects the inherent exploration component of epsilon-greedy policies and the stochastic nature of the environment.

6) Ideas for Future Work

To further improve the agent's performance and stability, the following enhancements could be considered:

- Double DQN (DDQN): Helps in reducing overestimation of Q-values by decoupling selection and evaluation of the action.
- Dueling DQN: Incorporates two streams to separately estimate state values and advantages for each action, which may lead to better policy evaluation.
- Prioritized Experience Replay: More important experiences can be replayed more frequently, leading to more efficient learning.
- Noisy Networks for Exploration: This method adds parametric noise to the weights to drive exploration, potentially leading to better performance than epsilon-greedy strategies.

Conclusion

The DQN agent was able to solve the specified task by reaching an average score above the required threshold. The project showcased the capabilities of reinforcement learning in a relatively complex and dynamic environment.

The agent's ability to generalize its policy during testing demonstrates the potential of DQN in solving episodic tasks. Future enhancements, as suggested, are expected to yield even more robust and efficient agents.

References

Deep Q-learning

Double DQN

Prioritized experience replay

Dueling DQN

multi-step bootstrap targets

Distributional DQN

Noisy Networks for Exploration

Rainbow