Rainbow DQN

Udacity Nanodegree – Project 1 Report | Value Based Methods

Aljoscha Grunwald
aljoscha.grunwald@gmail.com
GitHub: https://github.com/ARgruny

This project presents a unique implementation of the Rainbow Deep Q-Network (DQN) within the framework of the Udacity Deep Reinforcement Learning Nanodegree. In contrast to the standard Rainbow DQN, this adaptation omits the Distributional/Categorical DQN component due to the simplistic nature of the chosen environment. This modification acknowledges that the complexity of the Distributional/Categorical DQN may not yield significant benefits in simpler settings. Instead, the focus is on a streamlined version of the Rainbow DQN, utilizing key components such as Double-Q-Learning, Dueling Network architectures, N-step Learning, Prioritized Experience Replay (PER), and Noisy Networks. This tailored approach aims to leverage the strengths of each of these elements to efficiently address the challenges posed by the environment. The project demonstrates how this adapted Rainbow DQN model achieves effective learning and decision-making, while also discussing the implications of excluding the Distributional/Categorical DQN component in simpler reinforcement learning scenarios.

I. Introduction

Project 1 of the Udacity Deep Reinforcement Learning Nanodegree introduces a unique challenge set within a 3D environment where an agent is tasked with the collection of bananas. The objective is straightforward yet engaging navigate the environment to collect as many yellow bananas as possible while avoiding blue bananas. This task is set in a space with 37 dimensions representing the state space, which includes the agent's perception of its surroundings, and an action space of four possible movements.

The complexity of this project lies in the agent's ability to discern and learn the optimal strategy for maximizing its score, which is increased with each yellow banana collected and decreased with each blue banana. This environment provides an excellent platform for understanding and applying reinforcement learning concepts. The agent must develop a policy to navigate effectively through the state space, making decisions at each step that consider both immediate rewards and long-term strategy. The challenge encapsulates key aspects of reinforcement learning, including the balance between exploration and exploitation and the ability to learn from interactions within a dynamic environment. This project serves as a foundational exercise in understanding how agents perceive, interact with, and learn within a defined space, essential skills for any aspiring practitioner in the field of artificial intelligence.

The environment can be considered solved if an agent reaches an average score of greater or equal to 13 over the last 100 episodes.

II. Learning Algorithm

The learning algorithm used in the Udacity project is a variant of the Rainbow DQN, which combines several key enhancements to the basic Deep Q-Network (DQN) algorithm to improve its performance and robustness. The basic DQN algorithm uses a neural network to approximate the Q-value function, employing experience replay and fixed Q-targets to stabilize learning. The Rainbow DQN extends this with additional components: Double-Q Learning, Dueling Networks, N-step Learning, Prioritized Experience Replay, and Noisy Networks for exploration.

A. *Double-Q-Learning*

This component addresses the overestimation bias in Q-learning by maintaining two separate Q-value estimators. One estimator is used for action selection, and the other for action evaluation, thereby reducing overestimation and leading to more accurate Q-value estimates.

B. *Dueling Networks*

The Dueling Network architecture splits the neural network into two streams: one for estimating the state value function and the other for estimating the advantage function for each action. This allows for more efficient learning by separately assessing the value of each state and the importance of each action within those states.

C. *N-Step Learning*

N-step Learning incorporates the idea of looking ahead n steps to update the value estimates, instead of relying on a single step or the entire episode. This approach balances between the efficiency of TD learning and the lower variance of Monte Carlo methods.

D. *Prioritized Experience Replay*

Unlike standard experience replay where transitions are sampled uniformly, Prioritized Experience Replay samples experiences based on their importance, measured by the TD error. Important experiences are replayed more frequently, leading to more efficient learning.

E. *Noisy Networks for Exploration*

Noisy Networks introduce parameterized noise in the network weights to drive exploration. This method replaces the traditional epsilon-greedy strategy, enabling more sophisticated, state-dependent exploration policies. Each of these components contributes to the overall effectiveness of the Rainbow DQN, making it a powerful algorithm capable of handling complex and high-dimensional environments. The combination of these enhancements leads to faster learning, improved stability, and better overall performance compared to the basic DQN and its individual extensions.

III. Used Hyperparameters

BUFFER_SIZE = int(1e5): This parameter sets the size of the replay buffer to 100,000. A large buffer size allows the storage of a wide range of experiences, enabling the agent to learn from a diverse set of past experiences. This size is a common choice balancing memory usage and diversity of experiences.

BATCH_SIZE = 64: The minibatch size of 64 is used for sampling from the replay buffer. This size is large enough to provide a good approximation of the gradient when updating the network, but small enough to keep the computational load manageable.

GAMMA = 0.99: The discount factor of 0.99 determines the importance of future rewards. A value close to 1, like 0.99, places significant weight on future rewards, encouraging the agent to consider long-term outcomes.

TAU = 1e-3: This parameter is used for the soft update of target network parameters. A small value like 0.001 ensures that the target network parameters are updated slowly, providing stability to the learning process.

LR = 5e-4: The learning rate of 0.0005 is chosen as a moderate value that allows the network to learn effectively without making excessively large updates that could destabilize training.

UPDATE_EVERY = 4: This value specifies that the network is updated every 4-time steps. This frequency strikes a balance between learning from new experiences regularly and not updating too frequently, which can be computationally expensive.

NSTEP = 3: The number of steps for N-step learning is set to 3. This provides a compromise between the bias of one-step TD learning and the variance of Monte Carlo methods, allowing the agent to benefit from multi-step bootstrapping.

ALPHA = 0.6: In Prioritized Experience Replay (PER), alpha determines how much prioritization is used, with 1 being full prioritization. A value of 0.6 is a compromise that allows important experiences to be replayed more often while still maintaining some randomness.

BETA = 0.4: Beta is another hyperparameter for PER, which controls the importance-sampling weight. A value of 0.4 helps in compensating for the bias introduced by prioritized replay. This value is often annealed towards 1 over time to reduce the bias.

IV. Network Architcture

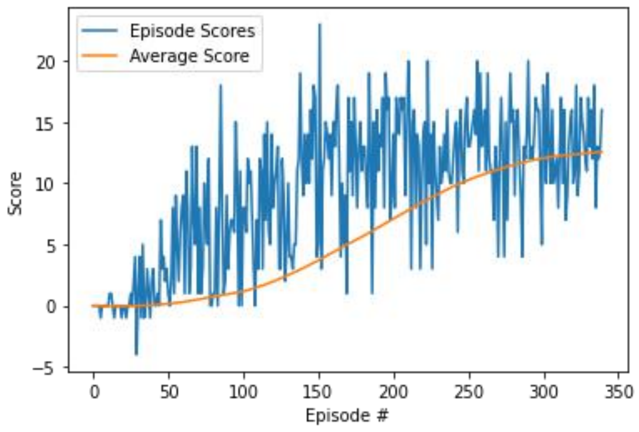
The NoisyDuelingQNetwork class, as described in the code, is a neural network model specifically designed for reinforcement learning tasks, combining the Dueling Network architecture with Noisy Networks for exploration. This network architecture is designed to work with environments characterized by a given state and action space.

Initialization Parameters:

- state_size (int): Dimension of each state.
- action_size (int): Dimension of each action.
- seed (int): Random seed for reproducibility.
- fc1_units, fc2_units, fc3_units (int): Number of nodes in the first, second, and third hidden layers, respectively, with default values of 256, 128, and 64.

The network uses Noisy Linear layers (NoisyLinear) to replace standard linear layers, introducing parameterized noise for exploration.
The first layer (fc1) takes the state as input and maps it to a hidden space with fc1_unit's nodes. The second layer (fc2) further maps the representation to fc2_unit's nodes. For Dueling DQN, two separate streams are created after the second layer. First the state value stream which contains a third layer (fc3) maps the representation to fc3_unit's nodes. The state value layer outputs a single value representing the state value. Second the action advantage stream with a fourth layer (fc4), parallel to fc3, also maps to fc3_unit's nodes. The action advantage layer outputs a vector with action size elements, each representing the advantage of a corresponding action. The final Q-value for each action is computed by combining the state value and the action advantages, with the mean advantage subtracted to maintain identifiability.

V. Results



The graph depicts the performance of the described agent over a series of episodes until the solve conditions are met. The agent was able to solve the environment in 340 episodes. The blue line represents the score obtained by the agent in each individual episode, while the orange line represents the average score over the last 100 episodes up to that point. The episode scores are quite volatile, with a significant variation in scores from one episode to the next. This is common in reinforcement learning environments where the agent's performance can be influenced by the stochastic nature of the environment or exploration strategies. Despite the volatility in individual episode scores, the orange line shows a clear upward trend over time. This indicates that the agent is learning and improving its policy as it experiences more episodes. The average score increases, suggesting that the agent is accumulating more reward on average as it learns. There are no evident plateaus or performance dips in the average score, which would suggest that the agent is consistently learning without getting stuck at a suboptimal policy. The results suggest that the agent, likely employing the previously described agent is effectively learning and improving its performance over time. The increasing average score is a positive sign of the agent's capability to optimize its policy in response to the received rewards.

VI. Future Work

Further improvements to the DQN algorithm could be made by hyperparameter tuning, testing out different network sizes and architectures, different exploration strategies like the traditional epsilon-greedy action, using reward shaping techniques or implementing recent algorithms like Soft Actor-Critic methods or PPO.