Navigation Project Report

Marc Poirier

May 6, 2025

1 Introduction

This report presents the results of training an agent to navigate and collect bananas in the Unity ML-Agents "Banana" environment using reinforcement learning. The agent's objective is to maximize its score by collecting yellow bananas (+1 reward) while avoiding blue bananas (-1 reward), with the goal of achieving an average score of at least +13 over 100 consecutive episodes. To accomplish this, we implement a variant of the Deep Q-Network (DQN) algorithm, specifically Dueling DQN, enhanced with Double DQN, multi-step learning, and Prioritized Experience Replay (PER).

2 Implementation Description

The implementation is organized into several key sections, each handling a specific aspect of the training process:

- **Setup and Installation**: Configures environment paths and installs necessary packages to ensure compatibility with the Unity ML-Agents environment. It also verifies the NumPy version.
- **Library Imports**: Imports essential libraries, including PyTorch for neural network implementation, NumPy for numerical operations, and Matplotlib for visualization.
- **Environment Initialization**: Initializes the Unity Banana environment in training mode, extracting the state space size (37 dimensions) and action space size (4 discrete actions).
- **State and Action Space**: The state space consists of 37 dimensions: 35 from ray-based perceptions (7 rays, each returning 5 values) and 2 from the agent's velocity. The action space includes 4 discrete actions: move forward, move backward, turn left, and turn right.
- **Dueling DQN Network**: Defines the neural network architecture, splitting Q-values into value and advantage streams for improved estimation.

- **Prioritized Replay Buffer**: Implements a replay buffer that prioritizes experiences based on their temporal difference (TD) errors, enhancing learning efficiency.
- **Agent Class**: Encapsulates the agent's logic, integrating Dueling DQN, Double DQN, multi-step learning, and PER, with methods for acting, stepping, and learning.
- **Training Loop**: Executes the training process, where the agent interacts with the environment, collects experiences, and updates its policy. It includes periodic plotting and model checkpointing.
- **Environment Closure**: Closes the Unity environment to free resources after training.

3 Learning Algorithm and Hyperparameters

The learning algorithm is a sophisticated variant of DQN, combining several advanced techniques:

- **Dueling DQN**: The Q-network is split into two streams: a value stream estimating the state value $V(s)$ and an advantage stream estimating the advantage $A(s, a)$ for each action. The Q-value is computed as $Q(s, a) = V(s) + (A(s, a) - \text{mean}(A(s)))$, improving generalization across actions.
- **Double DQN**: Reduces overestimation bias by using the local network to select actions and the target network to evaluate them, decoupling action selection and evaluation.
- **Multi-Step Learning**: Uses a 3-step horizon (n=3) to compute returns over multiple steps, providing a more accurate estimate of future rewards.
- **Prioritized Experience Replay (PER)**: Prioritizes experiences with higher TD errors, sampled with probability proportional to $|\text{TD error}|^{\alpha}$, and corrects bias with importance sampling weights proportional to $(N \cdot P(i))^{-\beta}$.

The chosen hyperparameters are:

- Buffer Size: 100,000 (replay buffer capacity)
- Batch Size: 128 (number of experiences sampled per learning step)
- Learning Rate: 0.0001 (step size for Adam optimizer, reduced for stability)
- Gamma: 0.99 (discount factor for future rewards)
- N-Step: 3 (multi-step learning horizon)
- Alpha (PER): 0.6 (priority exponent, balancing prioritization strength)

- Beta (PER): Starts at 0.4, linearly anneals to 1.0 over 1000 episodes (importance sampling correction)
- Epsilon Decay: Linear decay from 1.0 to 0.01 over 500 episodes (exploration rate)
- Tau: 0.001 (soft update factor for target network)
- Gradient Clipping: 1.0 (maximum gradient norm to prevent exploding gradients)

4 Model Architecture

The Dueling Q-Network architecture is designed as follows:

- **Input Layer**: 37 neurons, corresponding to the state size.
- **Hidden Layers**: Two fully connected layers, each with 64 neurons and ReLU activation functions.
- **Value Stream**: A fully connected layer with 1 neuron, outputting $V(s)$.
- **Advantage Stream**: A fully connected layer with 4 neurons (action size), outputting $A(s, a)$.
- **Output**: Q-values computed as $Q(s, a) = V(s) + (A(s, a) - \text{mean}(A(s)))$.

This architecture leverages PyTorch's neural network module for efficient computation and training.

5 Training Results

The agent was trained for a maximum of 2000 episodes. The environment was solved in **607 episodes**, with an average score of **13.09** over the last 100 episodes, exceeding the target of +13. Figure 1 illustrates the training progress, plotting scores per episode (blue), the average score over the last 100 episodes (red), and the target score of 13 (green dashed line).

6 Future Improvements

To enhance the agent's performance, the following strategies could be explored:

- **Hyperparameter Tuning**: Systematically adjust learning rate, batch size, alpha, beta, and epsilon decay to optimize convergence speed and stability.

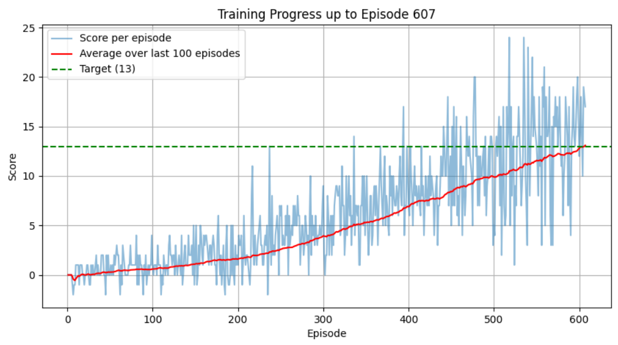

Figure 1: Training progress: scores per episode (blue), average over the last 100 episodes (red), and target score of 13 (green dashed line). The environment was solved in 607 episodes with an average score of 13.09.

- **Network Architecture**: Experiment with deeper networks (additional layers) or wider networks (more neurons) to increase capacity, or test alternative activation functions like Leaky ReLU.
- **Advanced Exploration**: Implement noisy nets or parameter space noise to replace epsilon-greedy exploration, encouraging more robust exploration strategies.
- **Reward Shaping**: Introduce intermediate rewards (e.g., for proximity to yellow bananas) to provide denser feedback and accelerate learning.
- **Transfer Learning**: Pre-train the model on a similar environment and fine-tune it for the Banana task to leverage prior knowledge.
- **Ensemble Methods**: Train multiple agents and combine their predictions to improve robustness and performance.
- **Curriculum Learning**: Gradually increase the environment's complexity (e.g., adding obstacles) to build the agent's skills incrementally.
- **Attention Mechanisms**: Incorporate attention layers to focus on the most relevant parts of the 37-dimensional state space, such as nearby bananas or obstacles.

These enhancements could lead to faster training, higher average scores, or greater generalization to other tasks.