Project Report: Dueling DQN Implementation

Overview

This project implements a **Dueling Deep Q-Network (Dueling DQN)** agent for reinforcement learning tasks in a Unity navigation environment. Reinforcement Learning (RL) is suitable for problems where an agent must learn optimal actions by interacting with an environment and receiving feedback in the form of rewards. This project demonstrates how a Dueling DQN agent can efficiently learn navigation policies and maximize cumulative reward.

Reinforcement Learning focuses on **exploration vs. exploitation**, enabling the agent to try different actions and learn from rewards. Deep Reinforcement Learning (Deep RL) combines RL with deep learning, allowing agents to process high-dimensional inputs such as state vectors or images, making it suitable for complex tasks.

The Dueling DQN architecture separates **state-value** and **advantage** estimations, improving learning stability and policy quality compared to standard DQN.

Defining the Problem as an RL Problem

The navigation task is framed as an RL problem: - **State space**: Vector representing the agent's current observation from the environment. - **Action space**: Discrete set of actions the agent can take (e.g., move left, right, forward). - **Reward**: Feedback from the environment based on the agent's action, encouraging optimal behavior.

The agent's objective is to learn a policy π(a|s) that maximizes the **expected cumulative reward** over episodes.

Deep RL is particularly suitable because the environment may contain complex dynamics and high-dimensional input features that traditional RL methods might struggle with.

Algorithm: Dueling DQN

The **Dueling DQN** algorithm extends standard DQN by using two separate streams in the network: - **Value Stream V(s)**: Estimates the value of being in a given state. - **Advantage Stream A(s,a)**: Estimates the relative advantage of each action.

Q-values are calculated as:

```
Q(s,a) = V(s) + (A(s,a) - mean(A(s,·)))
```

This architecture allows better distinction between state values and action advantages, improving learning stability.

Other Deep RL algorithms like DDPG, PPO, and SAC are useful for continuous action spaces and other environments, but Dueling DQN is well-suited for this discrete navigation task.

Code and Implementation

- **Navigation.py**: Initializes the Unity environment, runs training/testing, and manages checkpoints.
- **dqn_agent_banana.py**: Implements the agent with methods for action selection, learning, and experience replay.
- **model.py**: Defines the neural network architecture with separate value and advantage streams.
- **Checkpoints**: Saves trained model weights periodically for resuming or testing.

Training Process: 1. Initialize environment and agent. 2. For each episode, select actions via epsilon-greedy policy, collect rewards, and store experiences. 3. Sample minibatches from the replay buffer to train the network. 4. Gradually decay epsilon to reduce exploration. 5. Save checkpoints periodically.

Loss function:

```
loss = F.mse_loss(Q_expected, Q_target)
```

Results

- The agent learns to maximize cumulative reward efficiently.
- The dueling architecture stabilizes training and speeds up convergence.
- Training logs and saved checkpoints show progressive improvement in navigation performance.

Future Work

1. **Continuous Action Spaces**: Extend the implementation using DDPG or SAC for environments requiring continuous actions.
2. **Visual Inputs**: Incorporate high-dimensional visual inputs like camera images using convolutional networks.
3. **Multi-Agent Scenarios**: Implement multi-agent reinforcement learning for collaborative tasks.
4. **Hyperparameter Optimization**: Experiment with learning rates, batch sizes, and epsilon decay schedules for better performance.
5. **Reward Shaping**: Refine reward structures to encourage more efficient navigation and learning.

References

- Wang, Z., et al. *Dueling Network Architectures for Deep Reinforcement Learning*. https://arxiv.org/abs/1511.06581
- Unity ML-Agents Toolkit: https://github.com/Unity-Technologies/ml-agents
- Sutton, R.S., Barto, A.G. *Reinforcement Learning: An Introduction*, 2nd Edition.