Report on Dueling DQN Project

Introduction

This project implements a **Dueling Deep Q-Network (Dueling DQN)** agent for reinforcement learning tasks within a Unity-based navigation environment. Reinforcement Learning (RL) is an area of machine learning in which an agent learns to take actions in an environment to maximize cumulative rewards. The Dueling DQN is a variant of the Deep Q-Network (DQN) that introduces separate estimations for the state-value and the advantages of each action, leading to more stable and efficient learning.

Environment Description

The environment used for this project is based on Unity ML-Agents. It presents the agent with states and allows it to take discrete actions. The agent receives a reward signal based on its actions and transitions to new states until an episode ends.

- **State space**: Represented as vectors from the environment.
- **Action space**: Discrete set of actions the agent can take (e.g., move left, right, forward, etc.).
- **Reward signal**: Positive or negative values indicating whether the agent is moving closer to achieving its goal.

The objective for the agent is to learn an optimal policy that maximizes the expected cumulative reward over time.

Algorithm Overview

The algorithm used is **Dueling Deep Q-Network (Dueling DQN)**, an enhancement of the DQN algorithm. It is based on Q-learning but improves learning efficiency by separating value and advantage functions.

Key Concepts:

- **Q-learning**: Updates Q-values using the Bellman equation.
- **Experience Replay**: Stores transitions `(state, action, reward, next_state, done)` in a replay buffer. Random minibatches are sampled to break correlation between consecutive experiences.
- **Target Networks**: Stabilize training by using a separate target network for periodic updates.
- **Epsilon-Greedy Policy**: Balances exploration and exploitation by gradually reducing the probability of random action selection.

Dueling DQN Architecture

Instead of estimating Q-values directly, the network is split into two streams: - **Value stream**: Estimates the overall value of a given state, V(s). - **Advantage stream**: Estimates the relative advantage of each action, A(s,a).

The Q-values are then computed as:

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

This allows the network to better distinguish between the value of a state and the effect of taking different actions.

Code Description

1. Navigation.py

- The main entry point for training and testing the agent.
- Initializes the Unity environment.
- Creates an instance of the Dueling DQN agent.
- Runs training episodes, collects rewards, and manages checkpoints.

2. dqn_agent_banana.py

- Defines the `Agent` class implementing the Dueling DQN logic.
- Core functions:
- `step()`: Stores experience in the replay buffer and triggers learning.
- `act()`: Selects actions using epsilon-greedy policy.
- `learn()`: Samples minibatches and updates the Q-network.
- `soft_update()`: Gradually updates target network parameters.

3. model.py

- Defines the neural network used in the Dueling DQN architecture.
- Input: state vector.
- Hidden layers: fully connected layers with nonlinear activations.
- Output: Q-values for each action computed from separate value and advantage streams.

4. Checkpoints

- Model weights saved periodically (`checkpoint.pth`).
- Enables resuming training or testing a trained model.

Training Process

1. Initialize agent and replay buffer.
2. For each episode:
3. Start from an initial state.
4. Use epsilon-greedy policy to select actions.
5. Collect reward and transition to the next state.
6. Store transition in replay buffer.
7. Periodically sample minibatches to update Q-network.
8. Epsilon value decays over time to shift from exploration to exploitation.
9. Save trained model weights in checkpoint files.

Loss Function:

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

- `Q_expected`: Q-values from local network.
- `Q_target`: Target Q-values from target network.

Results

The Dueling DQN agent shows improved learning stability compared to standard DQN due to its architecture. The value and advantage streams enable better estimation of state importance and action significance, leading to faster convergence and higher rewards.

- The agent learns to maximize cumulative reward in the navigation task.
- Training logs and checkpoints demonstrate progressive improvement.

Conclusion

This project demonstrates the implementation of the **Dueling DQN algorithm** in a Unity navigation environment. By separating the estimation of state-value and action advantages, the agent is able to learn more efficiently and achieve better performance than standard DQN.

The codebase provides a foundation for experimenting with advanced reinforcement learning algorithms and can be extended to other environments.

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