Reinforcement Learning Report

Dokeun Lee

Problem 1 - DQN Implementation

(a) Complete the DQN skeleton code.

- QNetwork

```python
class QNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_dim, 256)
        self.fc_out = nn.Linear(256, action_dim)
        self.relu = nn.ReLU()

    def forward(self, s):
        s = self.relu(self.fc1(s))
        q = self.fc_out(s)

        return q
```

This is a neural network that takes a state as input and outputs the Q-value for each action. It uses one hidden layer with 256 nodes and a ReLU activation function.

- DQNAgent

```python
class DQNAgent:
    def __init__(self, state_size, action_size, device):
        self.state_size          = state_size
        self.action_size         = action_size
        self.device              = device

        # Do not modify these hyper-parameters
        self.Epochs              = 1000
        self.discount_factor     = 0.98
        self.learning_rate       = 0.001  # learning rate for q function
        self.epsilon             = 1.0    # initial epsilon value
        self.epsilon_min         = 0.001  # minimum epsilon value
        self.batch_size          = 256
        self.train_start         = self.batch_size * 5
        self.memory              = deque(maxlen=100000)  # replay memory

        # You can modify this depending on environments.
        self.epsilon_decay_rate = 0.995    # decay rate

        # Define and initialize your networks and optimizer
        self.q_net = QNetwork(state_size, action_size).to(device)
        self.q_target = QNetwork(state_size, action_size).to(device)
        self.optimizer = optim.Adam(self.q_net.parameters(), lr=self.learning_rate)
        self.loss_function = nn.MSELoss()

    def update_target_network(self):
        # implement target Q network update function
        self.q_target.load_state_dict(self.q_net.state_dict())

    def get_action(self, state, use_epsilon_greedy=True):
        if use_epsilon_greedy and random.random() < self.epsilon:
            # implement epsilon greedy policy given state
            action = random.randrange(self.action_size)
        else:
            # implement greedy policy given state
            # this greedy policy is used for evaluation
            state_tensor = torch.FloatTensor(state).unsqueeze(0).to(self.device)
            q_values = self.q_net(state_tensor)
            action = q_values.argmax().item()

        return action
```

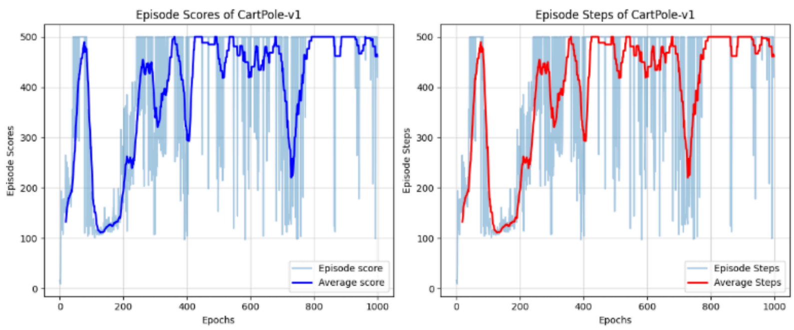
```python
    def append_sample(self, state, action, reward, next_state, done):
        # implement storing function given (s,a,r,s',done) into the replay memory.
        self.memory.append((state, action, reward, next_state, done))
    def get_samples(self, n):
        # implement transition random sampling function from the replay memory,
        # and make the transiton to batch.
        samples = random.sample(self.memory, n)
        s_batch = torch.tensor([s[0] for s in samples]).float().to(self.device)
        a_batch = torch.tensor([[s[1]] for s in samples]).long().to(self.device)
        r_batch = torch.tensor([[s[2]] for s in samples]).float().to(self.device)
        s_next_batch = torch.tensor([s[3] for s in samples]).float().to(self.device)
        done_batch = torch.tensor([[s[4]] for s in samples]).float().to(self.device)
        # i.e.) s_batch : (batch_size, state_dim)
        return s_batch, a_batch, r_batch, s_next_batch, done_batch

    def epsilon_decay(self):
        # implement epsilon decaying function
        self.epsilon = max(self.epsilon * self.epsilon_decay_rate, self.epsilon_min)

    def train(self):
        if len(self.memory) < self.train_start:
            return
        s_batch, a_batch, r_batch, s_next_batch, done_batch = self.get_samples(self.batch_size)

        # implement DQN training function.
        # You can return any statistics you want to check and analize the training.  (i.e. loss, q values, target q values, ... etc)
        target_values = r_batch + self.discount_factor * self.q_target(s_next_batch).detach().max(1)[0].unsqueeze(1) * (1 - done_batch)
        current_values = self.q_net(s_batch).gather(1, a_batch)
        loss = self.loss_function(current_values, target_values)

        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
```

Replay Memory: It uses a replay buffer (`memory`) to store transition tuples of (`state, action, reward, next_state, done`).
Target Network: To ensure training stability, a separate target network (`q_target`) is used in addition to the main network (`q_net`).
get_action: It balances exploration and exploitation using an epsilon-greedy policy.
train: The agent learns by sampling a minibatch from the replay buffer. The loss is calculated as the Mean Squared Error (MSE) between the TD target computed by the target network and the current Q-value predicted by the main network.

(b) Train the DQN Agent on the three environments.

(c) Plot evaluation episode score and steps graphs on each environment.

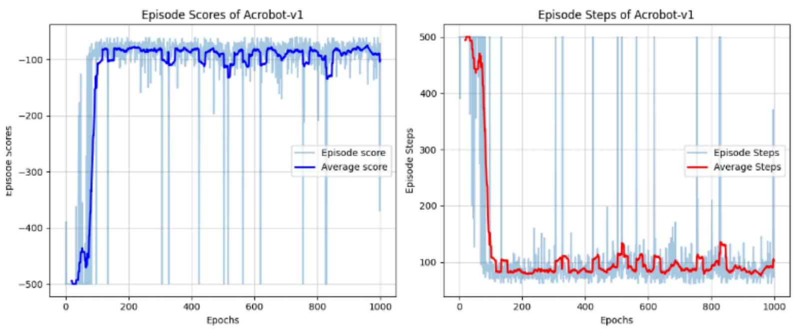
- Cartpole-v1, `self.epsilon_decay_rate = 0.995`




```
state_size:4, action_size:2, max_episode_steps:500, device:cuda

Epoch:0, Episode_score:16.0, Episode_steps:16, epsilon:0.995
Epoch:10, Episode_score:107.0, Episode_steps:107, epsilon:0.946354579813443
Epoch:20, Episode_score:147.0, Episode_steps:147, epsilon:0.9000874278732445
Epoch:30, Episode_score:178.0, Episode_steps:178, epsilon:0.8560822709551227
Epoch:40, Episode_score:321.0, Episode_steps:321, epsilon:0.8142285204175609
Epoch:50, Episode_score:500.0, Episode_steps:500, epsilon:0.7744209942832988
Epoch:60, Episode_score:293.0, Episode_steps:293, epsilon:0.736559652908221
Epoch:70, Episode_score:500.0, Episode_steps:500, epsilon:0.7005493475733617
Epoch:80, Episode_score:500.0, Episode_steps:500, epsilon:0.6662995813682115
Epoch:90, Episode_score:112.0, Episode_steps:112, epsilon:0.6337242817644086
Epoch:100, Episode_score:192.0, Episode_steps:192, epsilon:0.6027415843082742
Epoch:110, Episode_score:101.0, Episode_steps:101, epsilon:0.5732736268885887
Epoch:120, Episode_score:114.0, Episode_steps:114, epsilon:0.5452463540625918
Epoch:130, Episode_score:116.0, Episode_steps:116, epsilon:0.5185893309484582
Epoch:140, Episode_score:121.0, Episode_steps:121, epsilon:0.4932355662165453
Epoch:150, Episode_score:136.0, Episode_steps:136, epsilon:0.46912134373457726
Epoch:160, Episode_score:122.0, Episode_steps:122, epsilon:0.446186062443672
Epoch:170, Episode_score:198.0, Episode_steps:198, epsilon:0.42437208406280985
Epoch:180, Episode_score:123.0, Episode_steps:123, epsilon:0.4036245882390106
Epoch:190, Episode_score:140.0, Episode_steps:140, epsilon:0.38389143477919885
Epoch:200, Episode_score:171.0, Episode_steps:171, epsilon:0.36512303261753626
Epoch:210, Episode_score:252.0, Episode_steps:252, epsilon:0.3472722151889232
Epoch:220, Episode_score:220.0, Episode_steps:220, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:500.0, Episode_steps:500, epsilon:0.008090597512943647
Epoch:970, Episode_score:500.0, Episode_steps:500, epsilon:0.007695049256018795
Epoch:980, Episode_score:500.0, Episode_steps:500, epsilon:0.007318839301892576
Epoch:990, Episode_score:500.0, Episode_steps:500, epsilon:0.006961022203281065
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

The agent learned quickly and succeeded in keeping the episode score close to 500. The training is not perfectly stable, but it attempts to maintain its performance.

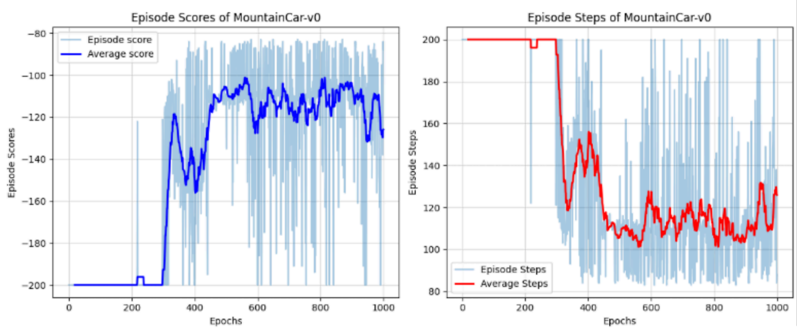
- Acrobot-v1, `self.epsilon_decay_rate = 0.995`




```
state_size:6, action_size:3, max_episode_steps:500, device:cuda

Epoch:0, Episode_score:-500.0, Episode_steps:500, epsilon:0.995
Epoch:10, Episode_score:-500.0, Episode_steps:500, epsilon:0.946354579813443
Epoch:20, Episode_score:-500.0, Episode_steps:500, epsilon:0.9000874278732445
Epoch:30, Episode_score:-500.0, Episode_steps:500, epsilon:0.8560822709551227
Epoch:40, Episode_score:-500.0, Episode_steps:500, epsilon:0.8142285204175609
Epoch:50, Episode_score:-500.0, Episode_steps:500, epsilon:0.7744209942832988
Epoch:60, Episode_score:-500.0, Episode_steps:500, epsilon:0.736559652908221
Epoch:70, Episode_score:-500.0, Episode_steps:500, epsilon:0.7005493475733617
Epoch:80, Episode_score:-106.0, Episode_steps:107, epsilon:0.6662995813682115
Epoch:90, Episode_score:-92.0, Episode_steps:93, epsilon:0.6337242817644086
Epoch:100, Episode_score:-89.0, Episode_steps:90, epsilon:0.6027415843082742
Epoch:110, Episode_score:-76.0, Episode_steps:77, epsilon:0.5732736268885887
Epoch:120, Episode_score:-70.0, Episode_steps:71, epsilon:0.5452463540625918
Epoch:130, Episode_score:-78.0, Episode_steps:79, epsilon:0.5185893309484582
Epoch:140, Episode_score:-91.0, Episode_steps:92, epsilon:0.4932355662165453
Epoch:150, Episode_score:-99.0, Episode_steps:100, epsilon:0.46912134373457726
Epoch:160, Episode_score:-71.0, Episode_steps:72, epsilon:0.446186062443672
Epoch:170, Episode_score:-83.0, Episode_steps:84, epsilon:0.42437208406280985
Epoch:180, Episode_score:-99.0, Episode_steps:100, epsilon:0.4036245882390106
Epoch:190, Episode_score:-108.0, Episode_steps:109, epsilon:0.38389143477919885
Epoch:200, Episode_score:-69.0, Episode_steps:70, epsilon:0.36512303261753626
Epoch:210, Episode_score:-82.0, Episode_steps:83, epsilon:0.3472722151889232
Epoch:220, Episode_score:-105.0, Episode_steps:106, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:-100.0, Episode_steps:101, epsilon:0.008090597512943647
Epoch:970, Episode_score:-98.0, Episode_steps:99, epsilon:0.007695049256018795
Epoch:980, Episode_score:-61.0, Episode_steps:62, epsilon:0.007318839301892576
Epoch:990, Episode_score:-94.0, Episode_steps:95, epsilon:0.006961022203281065
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

While there is some variance, the agent tries to maintain a score of -100, and the average reward converged to around this level as training progressed.

- MountainCar-v0, `self.epsilon_decay_rate = 0.995`




```
state_size:2, action_size:3, max_episode_steps:200, device:cuda

Epoch:0, Episode_score:-200.0, Episode_steps:200, epsilon:0.995
Epoch:10, Episode_score:-200.0, Episode_steps:200, epsilon:0.946354579813443
Epoch:20, Episode_score:-200.0, Episode_steps:200, epsilon:0.9000874278732445
Epoch:30, Episode_score:-200.0, Episode_steps:200, epsilon:0.8560822709551227
Epoch:40, Episode_score:-200.0, Episode_steps:200, epsilon:0.8142285204175609
Epoch:50, Episode_score:-200.0, Episode_steps:200, epsilon:0.7744209942832988
Epoch:60, Episode_score:-200.0, Episode_steps:200, epsilon:0.736559652908221
Epoch:70, Episode_score:-200.0, Episode_steps:200, epsilon:0.7005493475733617
Epoch:80, Episode_score:-200.0, Episode_steps:200, epsilon:0.6662995813682115
Epoch:90, Episode_score:-200.0, Episode_steps:200, epsilon:0.6337242817644086
Epoch:100, Episode_score:-200.0, Episode_steps:200, epsilon:0.6027415843082742
Epoch:110, Episode_score:-200.0, Episode_steps:200, epsilon:0.5732736268885887
Epoch:120, Episode_score:-200.0, Episode_steps:200, epsilon:0.5452463540625918
Epoch:130, Episode_score:-200.0, Episode_steps:200, epsilon:0.5185893309484582
Epoch:140, Episode_score:-200.0, Episode_steps:200, epsilon:0.4932355662165453
Epoch:150, Episode_score:-200.0, Episode_steps:200, epsilon:0.46912134373457726
Epoch:160, Episode_score:-200.0, Episode_steps:200, epsilon:0.446186062443672
Epoch:170, Episode_score:-200.0, Episode_steps:200, epsilon:0.42437208406280985
Epoch:180, Episode_score:-200.0, Episode_steps:200, epsilon:0.4036245882390106
Epoch:190, Episode_score:-200.0, Episode_steps:200, epsilon:0.38389143477919885
Epoch:200, Episode_score:-200.0, Episode_steps:200, epsilon:0.36512303261753626
Epoch:210, Episode_score:-200.0, Episode_steps:200, epsilon:0.3472722151889232
Epoch:220, Episode_score:-200.0, Episode_steps:200, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:-107.0, Episode_steps:107, epsilon:0.008090597512943647
Epoch:970, Episode_score:-85.0, Episode_steps:85, epsilon:0.007695049256018795
Epoch:980, Episode_score:-122.0, Episode_steps:122, epsilon:0.007318839301892576
Epoch:990, Episode_score:-200.0, Episode_steps:200, epsilon:0.006961022203281065
```

Starting from around 300 epochs, the performance gradually started to improve, converging to an average reward of around -120 with a maximum reward of -100.

(d) Are the DQN agents trained well? Report and analyze the experiment's results.

The DQN agent was trained successfully on the CartPole-v1 environment. Furthermore, it demonstrated meaningful learning on both Acrobot-v1 and MountainCar-v0, achieving convergence to scores around -100 and -120, respectively. However, the variance during training and the failure to reach a perfectly optimal score can be attributed to DQN's inherent problem of Overestimation Bias. This tendency to optimistically overestimate action values hinders training stability and limits the ability to find the optimal policy in complex or sparse-reward environments.

Problem 2 - Double DQN Implementation

(a) Complete the Double DQN skeleton code.

- QNetwork

```python
class QNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(QNetwork, self).__init__()
        self.fc1 = nn.Linear(state_dim, 256)
        self.fc_out = nn.Linear(256, action_dim)
        self.relu = nn.ReLU()

    def forward(self, s):
        s = self.relu(self.fc1(s))
        q = self.fc_out(s)

        return q
```

- DoubleDQNAgent

```python
class DoubleDQNAgent:
    def __init__(self, state_size, action_size, device):
        self.state_size          = state_size
        self.action_size         = action_size
        self.device              = device

        # Do not modify these hyper-parameters
        self.Epochs              = 1000
        self.discount_factor     = 0.98
        self.learning_rate       = 0.001  # learning rate for q function
        self.epsilon             = 1.0    # initial epsilon value
        self.epsilon_min         = 0.001  # minimum epsilon value
        self.batch_size          = 256
        self.train_start         = self.batch_size * 5
        self.memory              = deque(maxlen=100000)  # replay memory

        # You can modify this depending on environments.
        self.epsilon_decay_rate = 0.995    # decay rate

        # Define and initialize your networks and optimizer
        self.q_net = QNetwork(state_size, action_size).to(device)
        self.q_target = QNetwork(state_size, action_size).to(device)
        self.optimizer = optim.Adam(self.q_net.parameters(), lr=self.learning_rate)
        self.loss_function = nn.MSELoss()

    def update_target_network(self):
        # implement target Q network update function
        self.q_target.load_state_dict(self.q_net.state_dict())

    def get_action(self, state, use_epsilon_greedy=True):
        if use_epsilon_greedy and random.random() < self.epsilon:
            # implement epsilon greedy policy given state
            action = random.randrange(self.action_size)
        else:
            # implement greedy policy given state
            # this greedy policy is used for evaluation
            state_tensor = torch.FloatTensor(state).unsqueeze(0).to(self.device)
            q_values = self.q_net(state_tensor)
            action = q_values.argmax().item()

        return action
```

```python
def append_sample(self, state, action, reward, next_state, done):
    # implement storing function given (s,a,r,s',done) into the replay memory.
    self.memory.append((state, action, reward, next_state, done))
def get_samples(self, n):
    # implement transition random sampling function from the replay memory,
    # and make the transiton to batch.
    samples = random.sample(self.memory, n)
    s_batch = torch.tensor([s[0] for s in samples]).float().to(self.device)
    a_batch = torch.tensor([[s[1]] for s in samples]).long().to(self.device)
    r_batch = torch.tensor([[s[2]] for s in samples]).float().to(self.device)
    s_next_batch = torch.tensor([s[3] for s in samples]).float().to(self.device)
    done_batch = torch.tensor([[s[4]] for s in samples]).float().to(self.device)
    # i.e.) s_batch : (batch_size, state_dim)
    return s_batch, a_batch, r_batch, s_next_batch, done_batch

def epsilon_decay(self):
    # implement epsilon decaying function
    self.epsilon = max(self.epsilon * self.epsilon_decay_rate, self.epsilon_min)

def train(self):
    if len(self.memory) < self.train_start:
        return
    s_batch, a_batch, r_batch, s_next_batch, done_batch = self.get_samples(self.batch_size)

    best_actions = self.q_net(s_next_batch).argmax(dim=1).unsqueeze(1)
    next_q_values = self.q_target(s_next_batch).gather(1, best_actions).detach()
    target_values = r_batch + self.discount_factor * next_q_values * (1 - done_batch)

    current_values = self.q_net(s_batch).gather(1, a_batch)
    loss = self.loss_function(current_values, target_values)

    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

To address DQN's overestimation bias, a Double DQN agent was implemented.

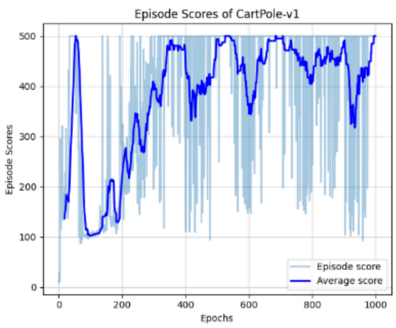
- The network architecture is identical to DQN, using two networks: a main network (`q_net`) and a target network (`q_target`).
- The key difference lies in the TD target calculation within the `train` method. Unlike DQN, the role of selecting the best next action is handled by the main network, while the role of evaluating the value of that selected action is handled by the target network.

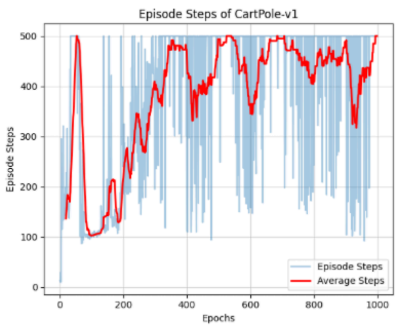
$$
\begin{aligned}
Q_1(s,a) &\leftarrow Q_1(s,a) + \alpha\left[r + \gamma Q_2(s', \mathrm{argmax}_{a'} Q_1(s',a')) - Q_1(s,a)\right] \\
Q_2(s,a) &\leftarrow Q_2(s,a) + \alpha\left[r + \gamma Q_1(s', \mathrm{argmax}_{a'} Q_2(s',a')) - Q_2(s,a)\right]
\end{aligned}
$$

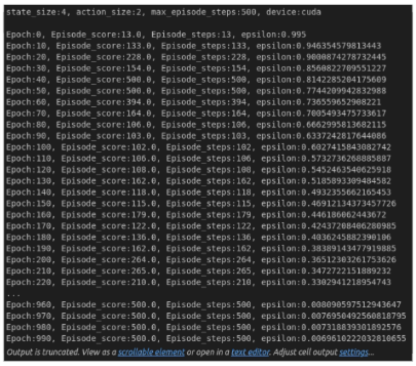
(b) Train the Double DQN Agent on the three environments.

(c) Plot evaluation episode score and steps graphs on each environment.

- Cartpole-v1, `self.epsilon_decay_rate = 0.995`



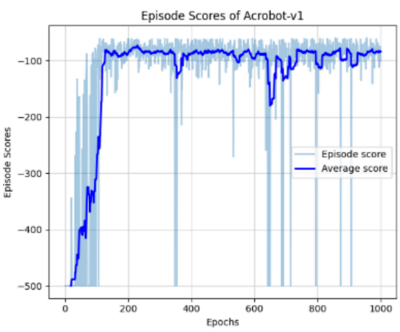


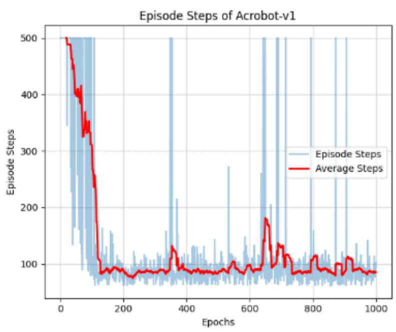


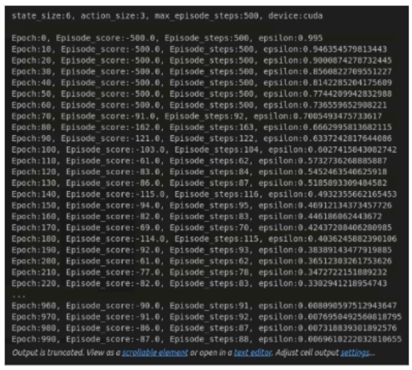
```
state_size:4, action_size:2, max_episode_steps:500, device:cuda

Epoch:0, Episode_score:13.0, Episode_steps:13, epsilon:0.995
Epoch:10, Episode_score:133.0, Episode_steps:133, epsilon:0.946354579813443
Epoch:20, Episode_score:228.0, Episode_steps:228, epsilon:0.9000874278732445
Epoch:30, Episode_score:154.0, Episode_steps:154, epsilon:0.8560822709551227
Epoch:40, Episode_score:500.0, Episode_steps:500, epsilon:0.8142285204175609
Epoch:50, Episode_score:500.0, Episode_steps:500, epsilon:0.7744209942832988
Epoch:60, Episode_score:394.0, Episode_steps:394, epsilon:0.736559652908221
Epoch:70, Episode_score:164.0, Episode_steps:164, epsilon:0.7005493475733617
Epoch:80, Episode_score:106.0, Episode_steps:106, epsilon:0.6662995813682115
Epoch:90, Episode_score:103.0, Episode_steps:103, epsilon:0.6337242817644086
Epoch:100, Episode_score:102.0, Episode_steps:102, epsilon:0.6027415843082742
Epoch:110, Episode_score:106.0, Episode_steps:106, epsilon:0.5732736268885887
Epoch:120, Episode_score:108.0, Episode_steps:108, epsilon:0.5452463540625918
Epoch:130, Episode_score:162.0, Episode_steps:162, epsilon:0.5185893309484582
Epoch:140, Episode_score:118.0, Episode_steps:118, epsilon:0.4932355662165453
Epoch:150, Episode_score:115.0, Episode_steps:115, epsilon:0.46912134373457726
Epoch:160, Episode_score:179.0, Episode_steps:179, epsilon:0.446186062443672
Epoch:170, Episode_score:122.0, Episode_steps:122, epsilon:0.42437208406280985
Epoch:180, Episode_score:136.0, Episode_steps:136, epsilon:0.4036245882390106
Epoch:190, Episode_score:162.0, Episode_steps:162, epsilon:0.38389143477919885
Epoch:200, Episode_score:264.0, Episode_steps:264, epsilon:0.36512303261753626
Epoch:210, Episode_score:265.0, Episode_steps:265, epsilon:0.347272215188923
Epoch:220, Episode_score:210.0, Episode_steps:210, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:500.0, Episode_steps:500, epsilon:0.008090597512943647
Epoch:970, Episode_score:500.0, Episode_steps:500, epsilon:0.007695049256081879
Epoch:980, Episode_score:500.0, Episode_steps:500, epsilon:0.007318839301892576
Epoch:990, Episode_score:500.0, Episode_steps:500, epsilon:0.006961022203281065
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Similar to DQN, the agent learned quickly and succeeded in keeping the episode score close to 500. The training is not perfectly stable, but it attempts to maintain its performance.

- Acrobot-v1, `self.epsilon_decay_rate = 0.995`



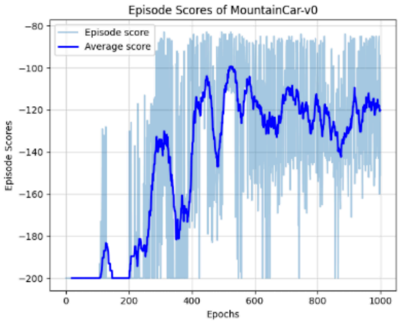


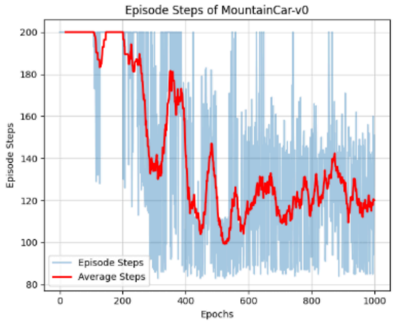


```
state_size:6, action_size:3, max_episode_steps:500, device:cuda

Epoch:0, Episode_score:-500.0, Episode_steps:500, epsilon:0.995
Epoch:10, Episode_score:-500.0, Episode_steps:500, epsilon:0.946354579813443
Epoch:20, Episode_score:-500.0, Episode_steps:500, epsilon:0.9000874278732445
Epoch:30, Episode_score:-500.0, Episode_steps:500, epsilon:0.8560822709551227
Epoch:40, Episode_score:-500.0, Episode_steps:500, epsilon:0.8142285204175609
Epoch:50, Episode_score:-500.0, Episode_steps:500, epsilon:0.7744209942832988
Epoch:60, Episode_score:-500.0, Episode_steps:500, epsilon:0.736559652908221
Epoch:70, Episode_score:-91.0, Episode_steps:92, epsilon:0.7005493475733617
Epoch:80, Episode_score:-162.0, Episode_steps:163, epsilon:0.6662995813682115
Epoch:90, Episode_score:-121.0, Episode_steps:122, epsilon:0.6337242817644086
Epoch:100, Episode_score:-103.0, Episode_steps:104, epsilon:0.6027415843082742
Epoch:110, Episode_score:-61.0, Episode_steps:62, epsilon:0.5732736268885887
Epoch:120, Episode_score:-83.0, Episode_steps:84, epsilon:0.5452463540625918
Epoch:130, Episode_score:-86.0, Episode_steps:87, epsilon:0.5185893309484582
Epoch:140, Episode_score:-115.0, Episode_steps:116, epsilon:0.4932355662165453
Epoch:150, Episode_score:-94.0, Episode_steps:95, epsilon:0.46912134373457726
Epoch:160, Episode_score:-82.0, Episode_steps:83, epsilon:0.446186062443672
Epoch:170, Episode_score:-69.0, Episode_steps:70, epsilon:0.42437208406280985
Epoch:180, Episode_score:-114.0, Episode_steps:115, epsilon:0.4036245882390106
Epoch:190, Episode_score:-92.0, Episode_steps:93, epsilon:0.38389143477919885
Epoch:200, Episode_score:-61.0, Episode_steps:62, epsilon:0.36512303261753626
Epoch:210, Episode_score:-77.0, Episode_steps:78, epsilon:0.347272215188923
Epoch:220, Episode_score:-82.0, Episode_steps:83, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:-90.0, Episode_steps:91, epsilon:0.008090597512943647
Epoch:970, Episode_score:-91.0, Episode_steps:92, epsilon:0.007695049256081879
Epoch:980, Episode_score:-86.0, Episode_steps:87, epsilon:0.007318839301892576
Epoch:990, Episode_score:-87.0, Episode_steps:88, epsilon:0.006961022203281065
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

A performance improvement over DQN is visible. The variance is smaller, indicating more stable training, although the maximum reward level is similar.

- MountainCar-v0, `self.epsilon_decay_rate = 0.995`






```
state_size:2, action_size:3, max_episode_steps:200, device:cuda

Epoch:0, Episode_score:-200.0, Episode_steps:200, epsilon:0.995
Epoch:10, Episode_score:-200.0, Episode_steps:200, epsilon:0.946354579813443
Epoch:20, Episode_score:-200.0, Episode_steps:200, epsilon:0.9000874278732445
Epoch:30, Episode_score:-200.0, Episode_steps:200, epsilon:0.8560822709551227
Epoch:40, Episode_score:-200.0, Episode_steps:200, epsilon:0.8142285204175609
Epoch:50, Episode_score:-200.0, Episode_steps:200, epsilon:0.7744209942832988
Epoch:60, Episode_score:-200.0, Episode_steps:200, epsilon:0.736559652908221
Epoch:70, Episode_score:-200.0, Episode_steps:200, epsilon:0.7005493475733617
Epoch:80, Episode_score:-200.0, Episode_steps:200, epsilon:0.6662995813682115
Epoch:90, Episode_score:-200.0, Episode_steps:200, epsilon:0.6337242817644086
Epoch:100, Episode_score:-200.0, Episode_steps:200, epsilon:0.6027415843082742
Epoch:110, Episode_score:-200.0, Episode_steps:200, epsilon:0.5732736268885887
Epoch:120, Episode_score:-200.0, Episode_steps:200, epsilon:0.5452463540625918
Epoch:130, Episode_score:-200.0, Episode_steps:200, epsilon:0.5185893309484582
Epoch:140, Episode_score:-200.0, Episode_steps:200, epsilon:0.4932355662165453
Epoch:150, Episode_score:-200.0, Episode_steps:200, epsilon:0.46912134373457726
Epoch:160, Episode_score:-200.0, Episode_steps:200, epsilon:0.446186062443672
Epoch:170, Episode_score:-200.0, Episode_steps:200, epsilon:0.42437208406280985
Epoch:180, Episode_score:-200.0, Episode_steps:200, epsilon:0.4036245882390106
Epoch:190, Episode_score:-200.0, Episode_steps:200, epsilon:0.38389143477919885
Epoch:200, Episode_score:-200.0, Episode_steps:200, epsilon:0.36512303261753626
Epoch:210, Episode_score:-200.0, Episode_steps:200, epsilon:0.3472722151889232
Epoch:220, Episode_score:-200.0, Episode_steps:200, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:-89.0, Episode_steps:89, epsilon:0.008090597512943647
Epoch:970, Episode_score:-149.0, Episode_steps:149, epsilon:0.007695049256081879 5
Epoch:980, Episode_score:-140.0, Episode_steps:140, epsilon:0.007318839301892576
Epoch:990, Episode_score:-109.0, Episode_steps:109, epsilon:0.006961022203281065 5
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Starting from around 200 epochs, the performance gradually increased, which is faster than DQN. The maximum reward is also better, reaching around -95.

(d) Are the Double DQN agents trained well? Report and analyze the experiment's results.

The Double DQN agent demonstrated improved performance compared to DQN overall. In the Acrobot environment, training stability was improved with less variance, and in the MountainCar environment, it achieved a faster learning speed and a higher maximum reward. This is because Double DQN effectively mitigates the overestimation bias of DQN by decoupling action selection and value evaluation. As a result, more accurate and reliable Q-value learning became possible, leading to enhanced overall performance and stability.

Problem 3 - Dueling Double DQN Implementation

(a) You can implement any RL algorithm that could increase performance, but experiment on the three environments.

- DuelingQNetwork

```python
class DuelingQNetwork(nn.Module):
    def __init__(self, state_dim, action_dim):
        super(DuelingQNetwork, self).__init__()

        self.feature_layer = nn.Sequential(
            nn.Linear(state_dim, 256),
            nn.ReLU(),
            nn.Linear(256, 128),
            nn.ReLU()
        )

        self.value_stream = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1)
        )

        self.advantage_stream = nn.Sequential(
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, action_dim)
        )

    def forward(self, state):
        features = self.feature_layer(state)
        v = self.value_stream(features)
        a = self.advantage_stream(features)
        q_values = v + (a - a.mean(dim=-1, keepdim=True))

        return q_values
```

- DuelingDoubleDQNAgent

```python
class DuelingDoubleDQNAgent:
    def __init__(self, state_size, action_size, device):
        self.state_size = state_size
        self.action_size = action_size
        self.device = device

        # Do not modify these hyper-parameters
        self.Epochs                = 1000
        self.discount_factor       = 0.98
        self.learning_rate         = 0.001  # learning rate for q function
        self.epsilon               = 1.0    # initial epsilon value
        self.epsilon_min           = 0.001  # minimum epsilon value
        self.batch_size            = 256
        self.train_start           = self.batch_size * 5
        self.memory                = deque(maxlen=100000)  # replay memory

        # You can modify this depending on environments.
        self.epsilon_decay_rate = 0.995    # decay rate

        self.q_net = DuelingQNetwork(state_size, action_size).to(device)
        self.q_target = DuelingQNetwork(state_size, action_size).to(device)
        self.optimizer = optim.Adam(self.q_net.parameters(), lr=self.learning_rate)

        self.update_target_network()

    def update_target_network(self):
        self.q_target.load_state_dict(self.q_net.state_dict())

    def get_action(self, state, use_epsilon_greedy=True):
        if use_epsilon_greedy and random.random() < self.epsilon:
            return random.randrange(self.action_size)
        else:
            with torch.no_grad():
                state_tensor = torch.FloatTensor(state).unsqueeze(0).to(self.device)
                q_values = self.q_net(state_tensor)
                return q_values.argmax().item()
```

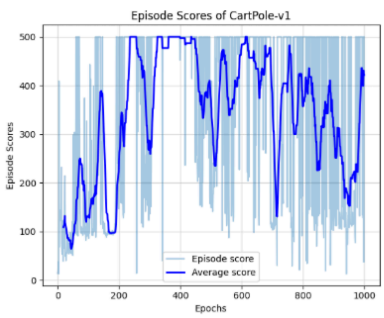
```python
def append_sample(self, state, action, reward, next_state, done):
    self.memory.append((state, action, reward, next_state, done))

def epsilon_decay(self):
    self.epsilon = max(self.epsilon * self.epsilon_decay_rate, self.epsilon_min)

def train(self):
    if len(self.memory) < self.train_start:
        return

    batch = random.sample(self.memory, self.batch_size)
    states, actions, rewards, next_states, dones = zip(*batch)

    states = torch.FloatTensor(np.array(states)).to(self.device)
    actions = torch.LongTensor(actions).unsqueeze(1).to(self.device)
    rewards = torch.FloatTensor(rewards).unsqueeze(1).to(self.device)
    next_states = torch.FloatTensor(np.array(next_states)).to(self.device)
    dones = torch.FloatTensor(dones).unsqueeze(1).to(self.device)

    with torch.no_grad():
        best_actions = self.q_net(next_states).argmax(dim=1).unsqueeze(1)
        next_q_values = self.q_target(next_states).gather(1, best_actions)
        target_q_values = rewards + (1 - dones) * self.discount_factor * next_q_values

    current_q_values = self.q_net(states).gather(1, actions)
    loss = nn.MSELoss()(current_q_values, target_q_values)

    self.optimizer.zero_grad()
    loss.backward()
    self.optimizer.step()
```

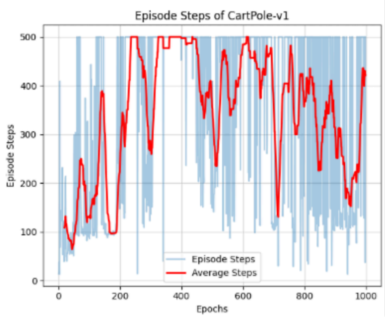
To further improve performance, a Dueling Double DQN agent was implemented by combining the Dueling Network architecture with Double DQN.

- `DuelingQNetwork`: This network is internally divided into two streams.
    - Value Stream: Calculates the value of the state itself, `V(s)`.
    - Advantage Stream: Calculates the advantage for each action over the average, `A(s, a)`.
- The final Q-value is calculated by combining these two streams using the formula: `Q(s, a) = V(s) + (A(s, a) - mean(A(s, a)))`. This allows for more efficient learning by decoupling the state's value from the actions.
- The rest of the agent follows the learning method of Double DQN.

(b) Train the Dueling Double DQN Agent on the three environments.
(c) Plot evaluation episode score and steps graphs on each environment.

- Cartpole-v1, `self.epsilon_decay_rate = 0.995`

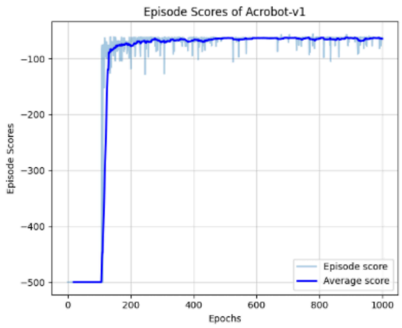


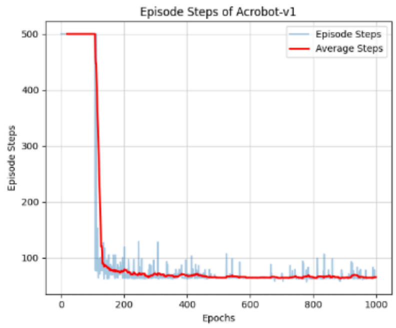



```
state_size:4, action_size:2, max_episode_steps:500, device:cuda

Epoch:0, Episode_score:14.0, Episode_steps:14, epsilon:0.995
Epoch:10, Episode_score:138.0, Episode_steps:138, epsilon:0.946354579813443
Epoch:20, Episode_score:66.0, Episode_steps:66, epsilon:0.9000874278732445
Epoch:30, Episode_score:56.0, Episode_steps:56, epsilon:0.8560822709551227
Epoch:40, Episode_score:98.0, Episode_steps:98, epsilon:0.8142285204175609
Epoch:50, Episode_score:257.0, Episode_steps:257, epsilon:0.7744209942832988
Epoch:60, Episode_score:500.0, Episode_steps:500, epsilon:0.736559652908221
Epoch:70, Episode_score:500.0, Episode_steps:500, epsilon:0.7005493475733617
Epoch:80, Episode_score:106.0, Episode_steps:106, epsilon:0.6662995813682115
Epoch:90, Episode_score:109.0, Episode_steps:109, epsilon:0.6337242817644086
Epoch:100, Episode_score:89.0, Episode_steps:89, epsilon:0.6027415843082742
Epoch:110, Episode_score:287.0, Episode_steps:287, epsilon:0.5732736268885887
Epoch:120, Episode_score:113.0, Episode_steps:113, epsilon:0.5452463540625918
Epoch:130, Episode_score:500.0, Episode_steps:500, epsilon:0.5185893309484582
Epoch:140, Episode_score:117.0, Episode_steps:117, epsilon:0.4932355662165453
Epoch:150, Episode_score:107.0, Episode_steps:107, epsilon:0.46912134373457726
Epoch:160, Episode_score:96.0, Episode_steps:96, epsilon:0.446186062443672
Epoch:170, Episode_score:97.0, Episode_steps:97, epsilon:0.42437208406280985
Epoch:180, Episode_score:93.0, Episode_steps:93, epsilon:0.4036245882390106
Epoch:190, Episode_score:276.0, Episode_steps:276, epsilon:0.38389143477919885
Epoch:200, Episode_score:500.0, Episode_steps:500, epsilon:0.36512303261753626
Epoch:210, Episode_score:222.0, Episode_steps:222, epsilon:0.3472722151889232
Epoch:220, Episode_score:500.0, Episode_steps:500, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:120.0, Episode_steps:120, epsilon:0.008090597512943647
Epoch:970, Episode_score:77.0, Episode_steps:77, epsilon:0.007695049256818795
Epoch:980, Episode_score:500.0, Episode_steps:500, epsilon:0.007318839301892576
Epoch:990, Episode_score:500.0, Episode_steps:500, epsilon:0.006961022032810655
```

Similar to DQN and Double DQN, it converges quickly and attempts to maintain performance, but its performance is slightly worse than the other two algorithms.

- Acrobot-v1, `self.epsilon_decay_rate = 0.995`

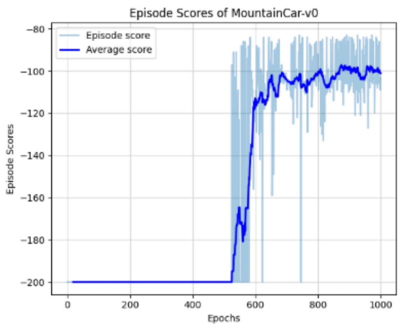


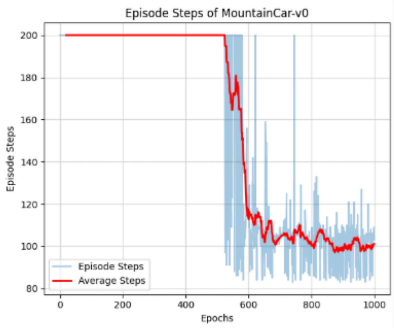



```
state_size:6, action_size:3, max_episode_steps:500, device:cuda

Epoch:0, Episode_score:-500.0, Episode_steps:500, epsilon:0.995
Epoch:10, Episode_score:-500.0, Episode_steps:500, epsilon:0.946354579813443
Epoch:20, Episode_score:-500.0, Episode_steps:500, epsilon:0.9000874278732445
Epoch:30, Episode_score:-500.0, Episode_steps:500, epsilon:0.8560822709551227
Epoch:40, Episode_score:-500.0, Episode_steps:500, epsilon:0.8142285204175609
Epoch:50, Episode_score:-500.0, Episode_steps:500, epsilon:0.7744209942832988
Epoch:60, Episode_score:-500.0, Episode_steps:500, epsilon:0.736559652908221
Epoch:70, Episode_score:-500.0, Episode_steps:500, epsilon:0.7005493475733617
Epoch:80, Episode_score:-500.0, Episode_steps:500, epsilon:0.6662995813682115
Epoch:90, Episode_score:-500.0, Episode_steps:500, epsilon:0.6337242817644086
Epoch:100, Episode_score:-500.0, Episode_steps:500, epsilon:0.6027415843082742
Epoch:110, Episode_score:-283.0, Episode_steps:284, epsilon:0.5732736268885887
Epoch:120, Episode_score:-71.0, Episode_steps:72, epsilon:0.5452463540625918
Epoch:130, Episode_score:-81.0, Episode_steps:82, epsilon:0.5185893309484582
Epoch:140, Episode_score:-70.0, Episode_steps:71, epsilon:0.4932355662165453
Epoch:150, Episode_score:-76.0, Episode_steps:77, epsilon:0.46912134373457726
Epoch:160, Episode_score:-69.0, Episode_steps:70, epsilon:0.446186062443672
Epoch:170, Episode_score:-69.0, Episode_steps:70, epsilon:0.42437208406280985
Epoch:180, Episode_score:-63.0, Episode_steps:64, epsilon:0.4036245882390106
Epoch:190, Episode_score:-83.0, Episode_steps:84, epsilon:0.38389143477919885
Epoch:200, Episode_score:-83.0, Episode_steps:84, epsilon:0.36512303261753626
Epoch:210, Episode_score:-62.0, Episode_steps:63, epsilon:0.3472722151889232
Epoch:220, Episode_score:-73.0, Episode_steps:74, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:-62.0, Episode_steps:63, epsilon:0.008090597512943647
Epoch:970, Episode_score:-62.0, Episode_steps:63, epsilon:0.007695049256081879
Epoch:980, Episode_score:-63.0, Episode_steps:64, epsilon:0.007318839301892576
Epoch:990, Episode_score:-82.0, Episode_steps:83, epsilon:0.006961022203281065
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Compared to the other two algorithms, the variance is definitively reduced. It consistently attempts to maintain a reward of -60, which seems to perfectly solve a certain drawback of DQN and Double DQN.

- MountainCar-v0, `self.epsilon_decay_rate = 0.995`






```
state_size:2, action_size:3, max_episode_steps:200, device:cuda

Epoch:0, Episode_score:-200.0, Episode_steps:200, epsilon:0.995
Epoch:10, Episode_score:-200.0, Episode_steps:200, epsilon:0.946354579813443
Epoch:20, Episode_score:-200.0, Episode_steps:200, epsilon:0.9000874278732445
Epoch:30, Episode_score:-200.0, Episode_steps:200, epsilon:0.8560822709551227
Epoch:40, Episode_score:-200.0, Episode_steps:200, epsilon:0.8142285204175609
Epoch:50, Episode_score:-200.0, Episode_steps:200, epsilon:0.7744209942832988
Epoch:60, Episode_score:-200.0, Episode_steps:200, epsilon:0.736559652908221
Epoch:70, Episode_score:-200.0, Episode_steps:200, epsilon:0.7005493475733617
Epoch:80, Episode_score:-200.0, Episode_steps:200, epsilon:0.6662995813682115
Epoch:90, Episode_score:-200.0, Episode_steps:200, epsilon:0.6337242817644086
Epoch:100, Episode_score:-200.0, Episode_steps:200, epsilon:0.6027415843082742
Epoch:110, Episode_score:-200.0, Episode_steps:200, epsilon:0.5732736268885887
Epoch:120, Episode_score:-200.0, Episode_steps:200, epsilon:0.5452463540625918
Epoch:130, Episode_score:-200.0, Episode_steps:200, epsilon:0.5185893309484582
Epoch:140, Episode_score:-200.0, Episode_steps:200, epsilon:0.4932355662165453
Epoch:150, Episode_score:-200.0, Episode_steps:200, epsilon:0.46912134373457726
Epoch:160, Episode_score:-200.0, Episode_steps:200, epsilon:0.446186062443672
Epoch:170, Episode_score:-200.0, Episode_steps:200, epsilon:0.42437208406280985
Epoch:180, Episode_score:-200.0, Episode_steps:200, epsilon:0.4036245882390106
Epoch:190, Episode_score:-200.0, Episode_steps:200, epsilon:0.38389143477919885
Epoch:200, Episode_score:-200.0, Episode_steps:200, epsilon:0.36512303261753626
Epoch:210, Episode_score:-200.0, Episode_steps:200, epsilon:0.3472722151889232
Epoch:220, Episode_score:-200.0, Episode_steps:200, epsilon:0.3302941218954743
...
Epoch:960, Episode_score:-104.0, Episode_steps:104, epsilon:0.008090597512943647
Epoch:970, Episode_score:-104.0, Episode_steps:104, epsilon:0.007695049256081879
Epoch:980, Episode_score:-120.0, Episode_steps:120, epsilon:0.007318839301892576
Epoch:990, Episode_score:-88.0, Episode_steps:88, epsilon:0.006961022203281065
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

Its convergence speed is slower than DQN and Double DQN. Performance gradually stabilizes from 500 epochs, and it was confirmed to have excellent performance, attempting to maintain a superior average reward of -100 compared to the other two algorithms.

(d) Are the Dueling Double DQN agents trained well? Report and analyze the experiment's results.

The Dueling Double DQN agent showed mixed but insightful results depending on the environment. In the Acrobot environment, it significantly reduced variance to stably maintain a high score (-60), and in the MountainCar environment, it ultimately achieved the best average score (-100) despite a slower start. This demonstrates the strength of the Dueling Network in finding more stable and qualitatively higher policies in complex problems by efficiently learning state values. Conversely, for the relatively simple CartPole environment, the added network complexity may have led to a slight degradation in performance. Nevertheless, the overwhelming stability and superior performance in the more difficult tasks can be attributed to the synergistic effect of Double DQN's bias reduction and the Dueling Network's efficient value learning.