Multi-Agent Reinforcement Learning

IQL and PS-DQN Implementation Report

---

1. Problem Statement and Goal

Problem

The task is to solve a cooperative multi-agent meeting problem in a grid world environment. Two agents must coordinate to simultaneously reach a shared goal location. This requires:

- **Coordination**: Both agents must arrive at the goal at the same time
- **Efficient navigation**: Agents must learn optimal paths while avoiding obstacles
- **Temporal coordination**: Agents must synchronize their movements to meet at the goal

Environment: Meeting Grid world

- **Grid size**: 5×5 discrete grid
- **Agents**: 2 agents that move simultaneously
- **Actions**: 5 actions per agent (up, down, left, right, stay)
- **Observations**: Each agent observes its own position, other agent's position, and goal position (6-dimensional vector)
- **Reward**:
  - +10.0 if both agents reach the goal simultaneously
  - -0.01 per step (encourages efficiency)
- **Episode termination**: Success when both agents are on the goal, or timeout after 50 steps

Goal

The goal is to implement different multi-agent reinforcement learning algorithms:

1. **Independent Q-Learning (IQL):** Baseline where agents learn independently
2. **Parameter-Shared DQN (PS-DQN):** Agents share a single Q-network

---

2. Independent Q-Learning (IQL)

Algorithm Overview

Independent Q-Learning (IQL) is a baseline MARL algorithm where each agent learns independently, treating other agents as part of the environment. This approach:

- Treats the multi-agent problem as multiple single-agent problems
- Each agent maintains its own Q-function: $Q_i(o_i, a_i)$
- Agents learn from their own experiences independently
- Simple but effective, though may struggle with non-stationarity

What “Non-Stationarity” Means

1. Stationarity in single-agent RL

In single-agent DQN, the environment is assumed to be **stationary**:

$P(s_{t+1}, r_t \mid s_t, a_t)$; does not change over time

This means:

If the agent takes the same action in the same state,
the reward distribution and next-state distribution are the same,
regardless of when in training this happens.

2. IQL breaks stationarity

In this project we implemented Independent Q-Learning (IQL) setup:

- We have two learning agents
- Each agent treats the other agent as part of the environment
- The other agent is also learning

* This means the environment changes during training.

3. Transition dynamics change over time

Each agent observes: $o_i = [self_{pos}, other_{pos}, goal_{pos}]$

$Agent_1$ learns a Q-function: $Q_1(o_1, a_1)$

But the next observation depends on $Agent_2$'s action: $o_1' = f(o_1, a_1, a_2)$

Now the problem:

- At training step t_1, $Agent_2$ behaves almost randomly (high ε)
- At training step t_2, $Agent_2$ behaves more greedily (low ε)
- At training step t_3, $Agent_2$ has learned to wait or approach strategically

So even if $Agent_1$ sees the same observation and takes the same action, the resulting next state and reward can differ depending on $Agent_2$'s current policy. ➡️ This violates stationarity.

4. Reward non-stationarity

reward in this project: +10 if BOTH agents reach the goal simultaneously

From agent 1’s perspective Early in training:

Going to the goal almost never gives +10
Agent 2 arrives late or not at all

Later in training:

The same action suddenly produces +10
Because agent 2 learned to coordinate

So the expected reward of an action changes over training, even in identical states.
➡️Direct violation of the MDP assumption**

5. IQL still works well in this project

1. Small environment: 5×5 grid, Only 2 agents, Low combinatorial explosion
2. Rich observations: Each agent observes
   - its own position
   - the other agent’s position
   - the goal position
3. Strong shared reward signal: +10 is large compared to step penalty

Key Characteristics

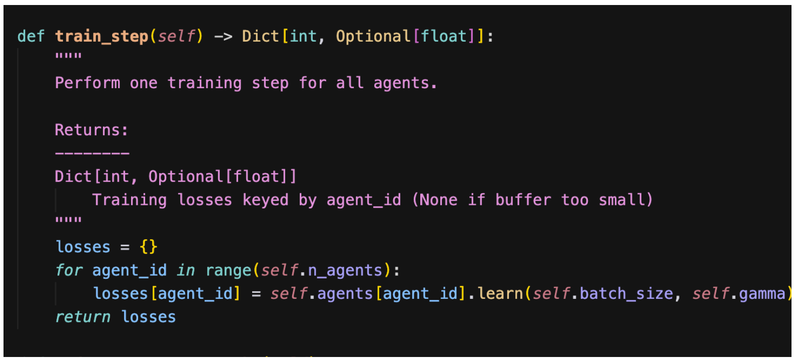
- Decentralized learning: Each agent has its own Q-network, optimizer, and replay buffer
- Independent policies: Agents don't explicitly coordinate during training
- Non-stationarity issue: As agents learn, the environment changes (other agents' policies change), making learning non-stationary

Implementation Structure

Core Files

✻ src / algos / **iql.py** (Main Algorithm Coordinator)

- **IQL class**: Manages multiple independent agents
  Manages training loop, evaluation,
  Coordinates agent actions and collects statistics
  Handles epsilon decay schedule across all agents
- **select_actions()**: Collects actions from all agents
- **store_transitions()**: Stores experiences in each agent's buffer
- **train()**: Main training loop with evaluation and logging



```python
def train_step(self) -> Dict[int, Optional[float]]:
    """
    Perform one training step for all agents.

    Returns:
    --------
    Dict[int, Optional[float]]
        Training losses keyed by agent_id (None if buffer too small)
    """
    losses = {}
    for agent_id in range(self.n_agents):
        losses[agent_id] = self.agents[agent_id].learn(self.batch_size, self.gamma)
    return losses
```

Image 1: iql train_step method-all agents train independently

✻ src / algos / **iql_agent.py** (Individual Agent Component)

- **IQLAgen class**: Represents a single independent agent Each agent has:
  Main Q-network(q_network)
  Target Q-network (target_network)
  Optimizer (Adam)
  Replay buffer (replay_memory)
- **select_actions**: Epsilon-greedy action selection
- **learn()**: Performs DQN update using agent's own buffer
- **update_target_network()**: Copies main network to target

✻ src / utils / **replay_memory.py** (Shared Replay Buffer)

- ReplayMemory class: Stores dict-based transitions
- Stores: (state_dict, action_dict, next_state_dict, reward, done)
- Each agent extracts its own data from the dicts during sampling

✻ src / utils / **qvalue_network.py** (Q-Network Architecture)

- QValueNetwor class: Feedforward neural network
- Architecture: input_dim (6) → hidden_dim (64) → hidden_dim (64) → num_actions (5)
- Used by all IQL agents

✻ scripts / **train_iql.py**

- Multi-seed training (seeds: 2025-2029)
- Aggregates results across seeds
- Saves results to JSON files

Learning Process

1. Each agent selects action independently using epsilon-greedy
2. Environment returns joint reward and next observations

3. Each agent stores transition in its own replay buffer
4. Each agent learns independently from its own buffer
5. Target networks updated periodically for stability

---

3. Parameter-Shared DQN (PS-DQN)

Algorithm Overview

Parameter-Shared DQN (PS-DQN) extends IQL by sharing a single Q-network across all agents. This approach:

- Uses one shared Q-network for all agents
- Enables parameter sharing, improving sample efficiency
- Agents learn from all agent experiences through the shared network

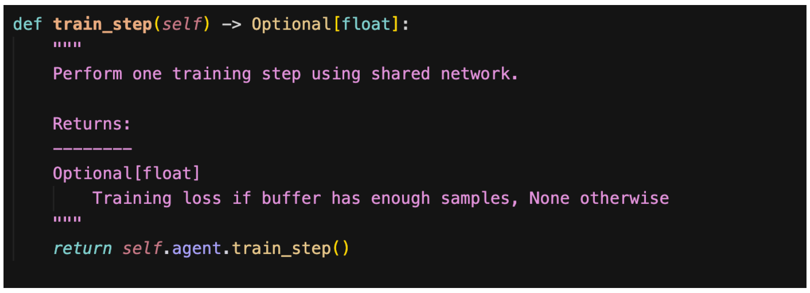
Key Characteristics

- Shared parameters: Single Q-network used by all agents
- Shared optimizer: One optimizer updates the shared network
- Shared replay buffer: Stores joint transitions (dict-based)
- Sample efficiency: Each timestep contributes n_agents samples to learning

Implementation Structure

Core Files

✻ src / algos / **ps_dqn.py** (Main Algorithm Coordinator)
- **PS_DQN class**: Manages shared learning components
  Coordinates training, evaluation, and logging
  Handles epsilon schedule and action selection
- **select_actions()**: Uses shared network for all agents
- **store_transitions()**: Stores joint transitions
- **train()**: Training loop with shared network updates



```python
def train_step(self) -> Optional[float]:
    """
    Perform one training step using shared network.

    Returns:
    --------
    Optional[float]
        Training loss if buffer has enough samples, None otherwise
    """
    return self.agent.train_step()
```

Image 2: ps_dqn train_step method-all agents use same q_network

✻ src / algos / **ps_dqn_agent.py** (Shared Learning Component)
- **PS_DQNAgent class**: Contains shared learning components
  Single shared Q-network (main and target)
  Single shared optimizer
  Single shared replay buffer
- **select_actions()**: Action selection using shared network
- **store_transitions()**: Stores dict-based transitions
- **train_step()**: Extracts individual agent data from dicts and combines them
- **update_target_network()**: Updates shared target network

✻ src / utils / **replay_memory.py** (Shared Replay Buffer)
- Same ReplayMemory class as IQL
- Stores joint transitions as dicts
- During sampling, individual agent transitions are extracted and combined

✻ src / utils / **qvalue_network.py** (Shared Q-Network)
- Same QValueNetwork architecture

  - Shared across all agents

✻ scripts / **train_ps_dqn.py**
  - Same structure as train_iql.py
  - Multi-seed training with aggregation
  - Results saved for comparison with IQL

Learning Process

1. All agents select actions using the shared Q-network
2. Environment returns joint reward and next observations
3. Joint transition stored in shared replay buffer (one per timestep)
4. During training:
    - Sample batch of joint transitions
    - Extract individual agent data from dicts
    - Combine all agent transitions (batch_size × n_agents samples)
    - Update shared network using combined batch
5. Target network updated periodically

Key Difference from IQL

- **IQL**: Each agent learns from its own experiences only
- **PS-DQN**: All agents learn from all agent experiences through parameter sharing
- Benefit: PS-DQN can be more sample-efficient, especially when agents are homogeneous

---

4. Experimental Results

Experimental Setup

- **Environment**: 5×5 Meeting Grid world with 2 agents
- **Training**: 1000 episodes per seed
- **Seeds**: 5 seeds (2025, 2026, 2027, 2028, 2029) for statistical significance
- **Hyperparameters**: Same for both algorithms (learning rate: 1e-3, batch size: 32, gamma: 0.99)

<u>**IQL Results**</u>

❖ Aggregated Results (across all seeds):

    Number of seeds: 5
    Seeds used: [2025, 2026, 2027, 2028, 2029]
    Final Metrics (last 100 episodes, mean ± std):
        Success Rate: 62.80% ± 10.40%
        Episode Length: 29.8 ± 4.9
        Return: 5.99 ± 1.09

<u>**PS-DQN Results**</u>

❖ Aggregated Results (across all seeds):

    Number of seeds: 5
    Seeds used: [2025, 2026, 2027, 2028, 2029]
    Final Metrics (last 100 episodes, mean ± std):
        Success Rate: 6.20% ± 2.04%
        Episode Length: 48.4 ± 0.7
        Return: 0.14 ± 0.21

Analysis of IQL Results

- **Success Rate**: 62.80% ± 10.40%
- **Performance**: Strong performance with 62.80% success rate
- **Consistency**: Moderate variance (10.40% std) indicates some variability across seeds, but generally stable
- **Interpretation**: IQL successfully learns to coordinate in most cases, with agents independently learning to reach the goal together
- **Episode Length**: 29.8 ± 4.9
    - **Efficiency**: Episodes complete in ~30 steps on average, well below the 50-step limit
    - **Variance**: Moderate variance (4.9 steps) suggests some episodes are easier/harder
    - **Interpretation**: Agents learn efficient paths, with successful episodes ending quickly

- **Return**: 5.99 ± 1.09
  - **Reward**: Positive average return indicates successful episodes outweigh step penalties
  - **Calculation**: With 62.80% success rate, successful episodes give +10 reward, failed episodes give ~-0.5 (50 steps × -0.01)
  - **Interpretation**: The algorithm is learning a viable policy that achieves the goal frequently

Overall Assessment: IQL demonstrates strong performance, successfully learning coordination despite treating other agents as part of the environment. The moderate variance suggests the algorithm is robust but performance could improve further with additional training or hyperparameter tuning.

Analysis of PS-DQN Results

- **Success Rate:** 6.20% ± 2.04%
- **Performance**: Poor performance with only 6.20% success rate
- **Consistency**: Low variance (2.04% std) indicates consistently poor performance across seeds
- **Interpretation**: PS-DQN struggles to learn effective coordination, despite parameter sharing
- **Episode Length**: 48.4 ± 0.7
  - **Efficiency**: Episodes nearly always hit the 50-step timeout (48.4 average)
  - **Variance**: Very low variance (0.7 steps) confirms most episodes time out
  - **Interpretation**: Agents are not learning to reach the goal efficiently, suggesting the shared network may be struggling with the task
- **Return**: 0.14 ± 0.21
  - **Reward**: Near-zero return indicates minimal learning progress
  - **Calculation**: With only 6.20% success, most episodes time out with ~-0.5 return (50 steps × -0.01)
  - **Interpretation**: The algorithm is barely learning, with returns close to random policy performance

Overall Assessment: PS-DQN performs significantly worse than IQL. This surprising result suggests that parameter sharing may not be beneficial for this specific task, or there may be implementation issues preventing effective learning.

---

5. Summary

Both algorithms use the same environment and evaluation protocol, enabling fair comparison. IQL demonstrates strong performance (62.80% success rate), successfully learning coordination despite treating other agents as part of the environment. PS-DQN performs poorly (6.20% success rate), suggesting that parameter sharing may not be beneficial for this specific coordination task, or requires different implementation strategies.

The results highlight that algorithm choice matters significantly for multi-agent tasks, and that seemingly more advanced approaches (parameter sharing) do not always outperform simpler baselines (independent learning). The modular implementation allows easy comparison and extension to other MARL algorithms like QMIX for further investigation.