Reinforcement Learning Final Project

1st Long Tri Thai Son
Institute of Artificial Intelligence
VNU University of Engineering and Technology
144 Xuan Thuy, Cau Giay, Ha Noi
22022653@vnu.edu.vn

2nd Nguyen Nhat Minh
Institute of Artificial Intelligence
VNU University of Engineering and Technology
144 Xuan Thuy, Cau Giay, Ha Noi
22022503@vnu.edu.vn

3rd Ngo Viet Anh
Institute of Artificial Intelligence
VNU University of Engineering and Technology
144 Xuan Thuy, Cau Giay, Ha Noi
22022508@vnu.edu.vn

***Abstract*—This report is about the implementation and evaluation of the DQN model optimized with cross Q-learning (CrossQ) for the battle_v4 environment in MAgent2. The goal was to train a team of agents to effectively compete against three types of opponents which are Random Agents, Pretrained Agent that use Deep Q-Network and a stronger hidden Final Agent. The performance of the model was evaluated based on its ability to outperform these opponents in the cooperative and competitive environment of battle_v4. Experimental results show that our CrossQ model achieved 100%-100%-100**

I. Introduction

Deep reinforcement learning (DRL) has been highly successful in solving complex decision-making tasks. Especially in environments that require agents to learn effective strategies through interactions.Traditional DRL methods such as Deep Q-Learning (DQN) rely heavily on experience replay and target networks to maintain the stability of the learning process and improve the teaching efficiency. However, this maybe ineffective when facing opponents that use various strategies or when the agents are required to adapt to unexpected changes.

In this experiment, we use Cross Q-Learning (CrossQ) (Lin et al., 2019), an optimized version of DQN that use SAC algorithm by removing target networks and introducing batch normalization across Q-values. This helps to eliminate the need for target networks by stabilizing Q-value updates through normalization and improve training stability as well as sample efficiency.

To evaluate the performance of CrossQ, we designed an experiment involving three types of adversaries: Random Agent, pre-trained DQN Agent, and a stronger, hidden Final Agent. These adversaries were chosen to test different aspects of CrossQ's performance, including its ability to explore simple, unpredictable behaviors, counter fixed policies as well as learn and adapt to a highly challenging enemy whose strategy remains hidden during training.

Summarize results:

TABLE I
Main results of CrossQ Self-play model

Model	Win	Draw	Lose
Random	100 ± 0	0 ± 0	0 ± 0
Pretrain-0	100 ± 0	0 ± 0	0 ± 0
New Pretrain	100 ± 0	0 ± 0	0 ± 0

II. Method

A. Deep Q-Network (DQN)

Deep Q-Network (DQN) combines Q-learning with deep neural networks to approximate the Q-value function, $Q(s, a)$, which estimates the expected cumulative reward for an agent taking action a in state s and following an optimal policy thereafter. The goal of DQN is to update the Q-values using the Bellman equation:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s_t, a_t) \right]$$

Where:

- $Q(s_t, a_t)$ is the Q-value of state s_t and action a_t.
- r_{t+1} is the immediate reward after taking action a_t.
- γ is the discount factor, determining the importance of future rewards.
- α is the learning rate, controlling how much new information overrides old.
- The term $\max_{a'} Q(s_{t+1}, a')$ represents the estimated maximum Q-value for the next state s_{t+1}.

DQN also incorporates **Experience Replay** and **Target Networks** to stabilize training. In **Experience Replay**, past transitions (s, a, r, s') are stored in a buffer and randomly sampled for updates, mitigating correlation between consecutive training samples. The **Target Network** is used to compute stable Q-value targets by periodically updating the parameters of the target network.

Despite its successes in single-agent environments, DQN struggles in multi-agent settings where the environment is non-stationary due to the concurrent learning of multiple agents.

B. Cross-Q Learning

The **CrossQ** strategy is a unique and innovative approach designed to enhance reinforcement learning efficiency and reduce computational costs. It is a modern off-policy actor-critic architecture based on the Soft Actor-Critic (SAC) structure, with significant architectural simplifications that result in faster training and reduced computational workload.

What sets **CrossQ** apart from other RL algorithms is the elimination of target networks, a crucial element in the same category of algorithms, including SAC. Target networks help stabilize value-based off-policy methods by setting up a separate, slowly-updating network that generates stable targets for value function learning. However, such complex methods can result in slower learning rates, and even SAC requires Polyak averaging related to target networks. **CrossQ** bypasses this requirement and offers a direct approach through Batch Normalization (BN). This technique can stabilize the agent's training without the need for target networks.

The key points that make up **CrossQ** are as follows:

- **Eliminating Target Networks:** Standard SAC involves using target networks for critics to prevent unstable training. Recently, some studies have shown that certain architectural changes can eliminate the need for these networks. In **CrossQ**, we extend this idea by using bounded activation functions or feature normalizers, which prevent critic divergence even without target networks. This simplification reduces algorithmic complexity while maintaining training stability.
- **Using Batch Normalization: CrossQ** aims to generalize the concept of Batch Normalization (BN) in critic networks, which is primarily used in supervised learning rather than reinforcement learning off-policy value-based algorithms, where BN performs best. Normalization is performed by BN, where the normalization line before each layer is set, thereby speeding up convergence. In **CrossQ**, disabling target networks through BN is feasible. By linking current and future state-action pairs, BN can also standardize the distributions of both inputs, thus eliminating out-of-distribution (OoD) cases caused by target networks. This strategy enhances learning and reduces the number of examples.
- **Wider Critic Networks: CrossQ** also uses wider critic networks to enhance the system's learning ability. While the biggest performance gains come from the first two design choices, the wider networks are still beneficial as the model can match or outperform other methods like REDQ and DroQ in terms of the number of state-of-the-art samples used.

III. Implementation

The implementation was carried out using a NVIDIA V100 GPU. The training process for the model took approximately 12 minutes. This efficiency demonstrates the capability of modern hardware in accelerating deep learning workflows.

- **Device:** NVIDIA V100 GPU
- **Training Time:** 12 minutes

A. Environment Setup

We utilized the `battle_v4` environment from `magent2`. The environment was configured with the following parameters:

- **Map Size:** 45×45
- **Maximum Steps per Episode:** 300
- **Reward Settings:**
  - `step_reward`: -0.005
  - `dead_penalty`: -0.1
  - `attack_penalty`: -0.05
  - `attack_opponent_reward`: 0.5
- **Extra Features:** Disabled

B. Neural Network Architecture

Each Q-network employs a hybrid architecture of convolutional and recurrent layers to process spatial and temporal features, followed by dense layers for action-value prediction. Key components are as follows:

- **Convolutional Layers:** Two convolutional layers with kernel size 3×3, followed by ReLU activations.
- **Recurrent Layer:** A GRU layer with 256 hidden units and 2 layers.
- **Dense Layers:** Two fully connected layers with 120 and 84 units, using ReLU and Tanh activations, respectively.
- **Output Layer:** A final linear layer mapping to the action space of size 21.

C. Training Procedure

The training process uses a combination of the agent's Q-network and a Cross Q-network (opponent's network) to guide action selection. The main steps are:

- **Replay Buffer:** A deque storing (s, a, r, s', d) tuples.
- **Batch Size:** 1024 samples per gradient step.
- **Optimizer:** Adam optimizer with a learning rate of 1×10^{-4}.
- **Discount Factor:** $\gamma = 0.99$.
- **Cross Ratio:** 0.5, determining the probability of using the Cross Q-network for action selection.
- **Gradient Clipping:** Gradient norm clipping is set to 1.0 to stabilize training.

The loss function is defined as the Mean Squared Error (MSE) between predicted Q-values and target values. The target values are computed as:

$$\text{target\_values} = r + \gamma \cdot \text{next\_q\_values} \cdot (1 - d),$$

where r represents rewards, γ is the discount factor, and d is a binary indicator for terminal states.

The training loop processes data from a replay buffer using a DataLoader and follows these steps:

1) Transfer states, actions, rewards, next states, and terminal flags to the GPU.
2) Compute the current Q-values by gathering the Q-values of the selected actions.

3) With probability `cross_ratio`, select the next actions using the Cross Q-network (opponent's network); otherwise, use the agent's own Q-network.
4) Compute the next Q-values using the target network, based on the selected next actions.
5) Compute the target values for training the Q-network.
6) Calculate the loss using the MSE between predicted Q-values and target values.
7) Update the Q-network using backpropagation with the Adam optimizer.

The total loss for each epoch is accumulated and returned to monitor training progress.

D. Training Loop

The agents were trained for 10 episodes with the following workflow:

1) At the beginning of each episode, the environment was reset, and all agents' observations were initialized.
2) At each step, actions were selected using the epsilon-greedy policy for both blue and red agents.
3) The selected actions were executed in the environment, and the resulting transitions were stored in the respective replay buffers.
4) After collecting sufficient experiences, the networks were trained using mini-batches sampled from the buffers. Loss was calculated using Mean Squared Error (MSE) between the predicted Q-values and the target Q-values.
5) Target networks were updated using a soft update mechanism with $\tau = 0.01$.
6) The episode ended when all agents terminated or the maximum steps were reached.

EXPERIMENT RESULTS

For the **DQN with random-training** model, it can be observed that, although the reward value increases steadily, there is a significant gap between the training model and the opponent model (as the opponent uses random actions). This gap limits the learning process of the training model.

For the **DQN with self-play** model, the two models achieve similar reward values, eliminating the disparity observed in random-training. This setup promotes better learning capabilities as the models strive to optimize more effectively compared to random-training.

Regarding the **CrossQ with self-play** model, similar to DQN with self-play, the two models exhibit reward values that are very close to each other. The CrossQ model demonstrates stronger intervention by leveraging the opponent's Q-values to calculate its own Q-values. The achieved results include a 100% win rate against the random and pretrained-1 models and an 100% win rate against the full-pretrained model.

CONCLUSION

We have implemented and evaluated the DQN model optimized with the CrossQ method in the battle_v4 environment of MAgent2. Experimental results show that our CrossQ model achieved a 100%-100%-100% win rate against the three types of opponents.

The CrossQ method has proven superior in eliminating target networks and applying batch normalization during agent training. This helps improve the stability of the training process and sample efficiency without the complexity of target networks.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv*, Dec. 2013. Link: https://arxiv.org/abs/1312.5602

[2] A. Bhatt, D. Palenicek, B. Belousov, M. Argus, A. Amiranashvili, T. Brox, and J. Peters, "CrossQ: Batch Normalization in Deep Reinforcement Learning for Greater Sample Efficiency and Simplicity," *arXiv*, Feb. 2019. Link: https://arxiv.org/abs/1902.05605

[3] `MAgent2`. Link: https://github.com/Farama-Foundation/MAgent2