Multi-Agent Battlefield Game with Federated Reinforcement Learning

Fardeen Hasib Mozumder[1†]

Department of Electrical and Electronic Engineering, Islamic University of Technology, Gazipur, Bangladesh

Abstract
This paper discusses the efficacy of federated learning in solving the Battlefield game from the petting zoo simulator for unseen and new environments utilizing multi-agent reinforcement learning. The game simulates a 12v12 battle in an open field with walls of different landscapes. The work incorporated federated learning by forming a global model by averaging the parameters of a few models trained locally on different landscapes utilizing a suitable reinforcement learning algorithm. In the simulation, the performance of the global model relative to local models within a new environment has been evaluated to assess the advantages of federated learning in these settings. The experimental results demonstrated that the global model consistently outperformed local models trained in a non-federated manner, thereby validating the effectiveness of federated learning in such environments.

Keywords:
Reinforcement Learning, Federated Learning, Battlefield, Multi-Agent Learning.

1. Introduction

Reinforcement Learning (RL) is a widely researched area within Machine Learning and Artificial Intelligence. In RL, the objective of the agent is to maximize its reward by learning effective ways to engage with its environment. This learning process typically occurs through trial and error, as the agent discovers how to associate each state s with the most advantageous actions a in order to attain long-term rewards. This line of research has found many applications, from robotics and autonomous vehicles to solving classic Atari games. In recent years, researchers have been trying to determine if it is possible to train RL algorithms to play these games and perform better than humans. Unlike RL agents, human players are able to grasp the basics of Atari games and perform reasonably well within just a few minutes [1]. Atari games became a popular benchmark for RL research following the release of the Arcade Learning Environment (ALE) in 2012 [2]. The integration of reinforcement learning techniques with deep neural networks allowed RL algorithms to learn to play Atari games directly from game screen images, utilizing variants of the Deep Q Networks (DQN) algorithm [3], [4], Proximal Policy Optimization (PPO) [5] as well as Actor-Critic methods [6]–[8]. In this project, reinforcement learning has been used to play the game Battlefield. This is a competitive multi-agent 12v12 game in an open field with different landscapes, where each agent tries to maximize its reward by eliminating the players of the other team. However, the agents trained in a certain landscape perform poorly, i.e., unable to eliminate players from the other team efficiently, in unseen landscapes or environments. In this work, the efficacy of federated learning has been studied in such environments. Multiple battlefield games are played in different environments simultaneously by training the local agents with a suitable Deep RL algorithm. Federated Learning [9], [10] has been utilized to combine the models resulting from training the agents in their local environments and create a global model by aggregating the parameters of the local models at the end of each training round. The obtained global model is then shared among all local models to be used as an initialization for the next training round until each of the local models converges. The workflow of the work can be summarized as follows:

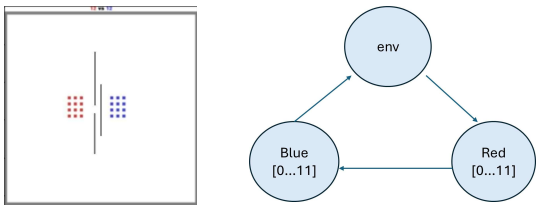
a) Several RL algorithms such as Deep Q-Learning with Neural Networks (DQN), advanced Actor-Critic (A2C), and Proximal Policy Optimization (PPO) have been tried to train the agents and compare their performances to find the best algorithm suitable for the game's setting.

b) Four environments with different wall shapes have been created using environment wrappers of the PettingZoo simulator. Three of these environments were used to train the federated global model and one to test it.

c) Federated learning has been implemented by aggregating the local models resulting from training the agents in the environments allocated for the training purpose using the suitable RL algorithms to form a global model.

Manuscript received November 5, 2025
Manuscript revised November 20, 2025
https://doi.org/10.22937/IJCSNS.2025.25.11.4

d) The performance of the global model and the local models has been compared in the unseen test environment to demonstrate the efficacy of federated learning in such environments.



(a) Battlefield environment (b) Agent environment cycle

Fig. 1. Battlefield environment with 12 red agents vs 12 blue agents, and the agent-environment cycle.

The following sections introduce the key concepts used in the work. For example, Section II introduces the agents, environments, and the simulator used in the project. Section III explains how to create and split new environments for training and testing. Section IV presents three different algorithms used to solve the game and compares their implementations. The usage of federated learning is explained in Section V. The results and findings of our simulations are presented in Sections VI and VII, where different approaches are investigated to improve the performance of each implementation. Finally, Section VIII concludes the paper and provides potential future research based on the findings of the project.

2. Agent and Environment

A. Environment

For this project, each environment is created and simulated using PettingZoo. PettingZoo is a Python library for conducting research in multi-agent reinforcement learning, akin to a multi-agent version of Gym [11]. PettingZoo includes several families of environments to test reinforcement learning algorithms and allows us to use third-party environments through wrappers. Magent is one of the families of environments included in PettingZoo, created by Zheng et al. [12]. Magent includes Battlefield, which is what was used for this project. Battlefield is a two-team battle game where each agent has to find out the best way to maximize their own reward while maximizing their team's success. Meanwhile, the agents have to maneuver around pre-defined obstacles in a medium-sized map, as seen in Fig. 1.

B. Agent

Agents are rewarded based solely on their individual performance, without consideration for the performance of their teammates or opponents, which complicates the implementation of team cooperation. Since agents regenerate health slowly, a strategy that prioritizes rapid elimination of enemies is favored over a slower, tactical approach. The environment includes two teams, each consisting of 12 units. Units take turns sequentially, starting with unit 0 to 11 from the red team, followed by unit 0 to 11 from the blue team, as illustrated in Fig. 1(b). Every unit has 10 HP and recovers 0.1 HP per turn. Each attack inflicts 2 HP of damage on enemies, and attacks on teammates are ignored. As in all Magent environments, agents can either move or attack during their turn, but not both. Positive rewards include 5 points for eliminating an enemy and 0.2 points for attacking an opponent, while penalties consist of -0.005 points per turn, -1 for attacking, and -0.1 for dying.

C. Action and State Space

In terms of the action space available for each agent, the space is discrete and has 12 possible move actions plus eight attack actions. The action space is shown in Fig. 2. The state space is an 80x80 matrix, the same size as the map, containing 0/1 for blue and red team presence and HP and the obstacle present, thus giving the location and health of each agent and obstacles' locations. HP values are normalized (0-1 range). The observation state is an array of shape (n agents, view width, view height, n channel) for all agents. HP is the normalized health point (range 0-1). The channels are team blue HP and presence, team red HP and presence, and Minimap is used to give a fuzzy global observation to the agents.

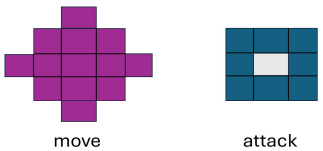


Fig. 2. Battlefield environment parameters, its observation space and action space.

3. Simulation Setup

Each local model was trained on environments with slightly different wall setups, and the final testing was done on an unseen environment. In Fig. 3, the different wall setups used are shown. The wall setups used for training and testing was randomized to reduce the potential bias of intentionally choosing a particular wall setup for testing. In particular, for the environments used in this project that did have walls, pnorm unit circles have been drawn from the centers of the displays. The diamond wall setup is the L1-norm unit circle, the circle wall setup is the L2-norm unit circle, and the L3 wall (rounded, but squarish with softer corners) setup is the L3-norm unit circle. These were chosen because each wall setup offers slightly different properties and takes up different amounts of area. Additionally, the code for each environment only had to be minimally changed.

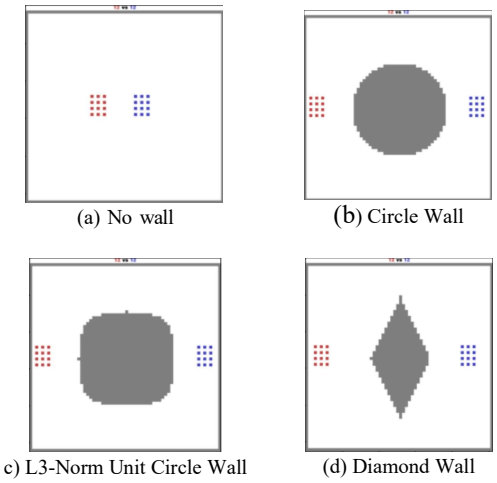


Fig. 3. Different wall setups used in training and testing. Wall setups (a) - (c) were used in training and wall setup (d) was used for testing the final federated model.

4. Implemented Approaches for Solving the Game

A. Deep Q-Learning with Neural Networks

The Deep Q-Network (DQN) algorithm integrates a nonlinear function approximation method called Deep Neural Network (DNN) with the Q-learning algorithm [3]. This approach employs a multi-layered neural network that generates a vector of action values for each input state. Key components of this algorithm include the use of a target network and an experience replay buffer. DQNs have demonstrated the ability to reach human-level performance across various Atari games [13]. These algorithms are remarkably flexible and stable, showing state-of-the-art performance, and have seen many extensions. To address the overestimation bias of Q-learning, Double DQN (DDQN) was proposed, which decouples selection and evaluation of the bootstrap action [14]. The authors in [3] used the last four frames as input of their DQN, which allowed the model to access more information than just the current observation. To help the agent remember older information, the authors in [15] modified the DQN architecture by adding a recurrent layer between and introducing Deep Q-Learning with Recurrent Neural Networks(DQRN). In this project, DQN based on Neural Fitted Q Iteration was used to solve the multiagent battlefield game. The DQN in the project also includes a replay buffer, target network, and gradient clipping technique for stability and avoiding large updates during the training phase.

B. Actor – Critic

Actor-Critic is a popular RL algorithm that combines policy-based and value-based approaches. It consists of two key components: the actor, which is responsible for selecting actions based on a policy, and the critic, which evaluates the actions by estimating the value function. In this architecture, the actor updates the policy parameters to achieve maximum rewards. the critic provides feedback on the actions the actor took by estimating the TD error, where TD is. By doing so, the critic reduces the policy gradients' variances and maintains the training process's stability. As this method learns both the value and policy functions concurrently, it converges faster and thus proves to be highly efficient. Variants of the actor-critic algorithm, such as Synchronous Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C) [6], have been widely used in deep RL applications. In this project, the synchronous advantage actor-critic (A2C) approach was utilized to play the Magent Battlefield game with the trained agents.

C. Proximal Policy Optimization

Proximal Policy Optimization (PPO) is a reinforcement learning algorithm designed to balance efficiency and stability in policy updates. Unlike traditional policy gradient methods, PPO restricts the margin of the change in policy by utilizing a clipped objective function. This function puts a limit on the gradient of policy, i.e., the difference between the new and old policy, to a specified range and so prevents large fluctuations in action probabilities, The process helps to stabilize the training process. PPO is an on-policy algorithm as it updates the policy using the trajectories of the current policy, thereby enhancing sampling efficiency. PPO simplifies the TRPO [16] while achieving similar performance but with less computational overhead. Due to its robustness and ease of implementation, PPO has become a preferred algorithm for Deep RL tasks. PPO has demonstrated state-of-the-art results in various domains, from games to robotic control.

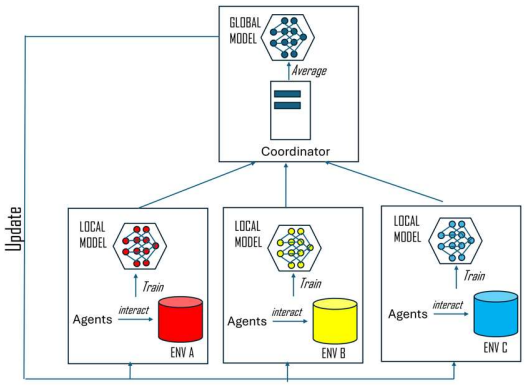


Fig. 4. Federated learning architecture and the client-server model [17]. Each client is solving a Battlefield game with a different environment and their model is being shared to update the global model

5. Federated Learning

Federated learning is a distributed machine learning approach where models are trained across multiple devices, servers or environments without transferring the data to a central server/environment [18]. Separate models are trained on their own server/environment and the model updates, like gradients, are sent to a central environment, which is aggregated to update a global model. This method is useful when two or more parties want to collaboratively build a model using their independent datasets. The resulting model's performance is often comparable to that of a centrally trained model with all the data combined. Federated learning can be implemented in various architectures, such as peer-to-peer or client-server models. In this project, a client-server model [17] was considered where multiple Battlefield games were played in different environments. In this project, the agents in three different environments were trained to form their local models, and the FedAvg algorithm [9] was used to create a global model by taking the average of the parameters of the local models with a view to have better generalization and enhanced performance in new environments. Algorithm 1 illustrates the implementation of the FedAvg.

Algorithm 1 Federated Averaging (FedAvg)

Input: Number of clients K, number of communication rounds T, learning rate η, local epochs E, batch size B
Server initializes: global model weights w_0
for each round $t = 1, \ldots, T$ **do**
  Server selects a random subset of clients $S_t \subseteq \{1, 2, \ldots, K\}$
    for each client $k \in S_t$ **in parallel do**
      Client k downloads global model weights w_t
      Client k updates local model by solving:

$$w_t^{\,k} \leftarrow w_t - \eta \nabla \ell_k(w_t; D_k)$$

  using local data D_k for E epochs with batch size B
    end for
    Server aggregates local models:

$$w_{t+1} \leftarrow \frac{1}{|S_t|} \sum_{k \in S_t} w_t^{\,k}$$

end for
Output: Final global model weights w_T

6. Implementing RL Policies for Local Games: Algorithm Selection

First, we aim to find the most suitable RL algorithm for the Magent battlefield gam setting. DQN, A2C, and PPO methods were tried and PPO was found to yield the most rewards among the approaches. Hence, the PPO was chosen to incorporate into the project to train the local models. After each round of training the local models in three different environments, the FedAvg was used to get a global model of which parameters were utilized as the initial

parameters for the local models for the following round. The final global model was obtained after three such rounds of training.

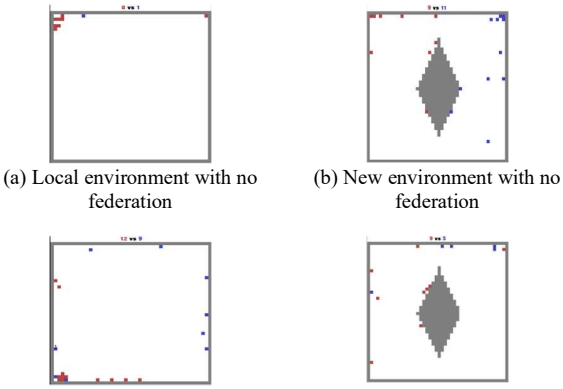
a) Local environments used for training

(b) New diamond wall environment

Fig. 5. Reward of the federated and vanilla (non-federated) PPO algorithms.

7. Experimental Results

To test the performance of the federated models and non-federated models, each of them was evaluated in the local environments and a new environment. Fig. 5 (a) shows that the non-federated models give us higher rewards than the federated models in their own environment and Fig. 5 (b) that the federated model provides higher rewards in a new and unseen environment. As the non-federated models are trained on their local models in opposition to federated models, which are trained by global parameter sharing, non-federated models outperform the federated model in their corresponding local environments. On the other hand, the federated models outperformed the non-federated models in a completely new environment which was expected as they were trained to enhance generalization. Fig. 6(a) also shows that the red agents trained in a non-federated manner were able to kill their oppositions quickly in their local environment, but struggled to do so in an unseen environment as shown in Fig. 6(b). On the other hand, the red agents using the global model were able to eliminate their opposition fast in the unseen environment, as shown in Fig. 6(d), but failed to achieve the same in one of the local environments, as shown in Fig. 6(c). Hence, it was clear from the experimental result that federated learning increases the agents' performance and yields higher rewards in unseen environments, as it enhances generalization.



(a) Local environment with no federation

(b) New environment with no federation

(c) Local environment with federation

(d) New environment with federation

Fig. 6. The result of the Battlefield game. (a) In the setting with no federation agents trained using non-federated model were able to kill opponents faster in local environment. (b) Agents trained using non-federated model were not able to kill opponents quickly in new environment. (c) Agents trained using federated model were not able to kill opponents quickly in local environment. (d) Agents trained using federated model were able to kill opponents faster in new environment.

8. Conclusion and Future Work

The project shows that federated learning can enhance the performance of reinforcement learning algorithms within multi-agent systems when agents interact with entirely new environments. However, training and testing were limited to only a few environments, which will be expanded in future work. Additionally, to further investigate the efficacy of federated learning, the project will be simulated with algorithms beyond FedAvg to determine whether they improve agents' efficiency in gameplay. Future work will also explore the effectiveness of the federated learning and multi-agent learning combination in diverse applications, such as robotics, healthcare, and telecommunications, as any multi-agent problem involving unpredictable environments stands to benefit from the robustness that federated learning offers.

References

[1] P. Tsividis, T. Pouncy, J. L. Xu, J. B. Tenenbaum, and S. J. Gershman, "Human learning in atari. in 2017 aaai spring symposia," 2017.

[2] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," Journal of Artificial Intelligence Research, vol. 47, pp. 253–279, 2013.

[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari

with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 20

[4] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," in Thirtysecond AAAI conference on artificial intelligence, 2018.

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in International conference on machine learning. PMLR, 2016, pp. 1928–1937.

[7] M. Babaeizadeh, I. Frosio, S. Tyree, J. Clemons, and J. Kautz, "Reinforcement learning through asynchronous advantage actor-critic on a gpu," arXiv preprint arXiv:1611.06256, 2016.

[8] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning et al., "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in International Conference on Machine Learning. PMLR, 2018, pp. 1407–1416.

[9] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in Artificial intelligence and statistics. PMLR, 2017, pp. 1273– 1282.

[10] J. Konecnˇ y, H. B. McMahan, D. Ramage, and P. Richt ` arik, "Federated ´ optimization: Distributed machine learning for on-device intelligence," arXiv preprint arXiv:1610.02527, 2016.

[11] J. K. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sulivan, L. Santos, R. Perez, C. Horsch, C. Dieffendahl, N. L. Williams, Y. Lokesh, R. Sullivan, and P. Ravi, "Pettingzoo: Gym for multi-agent reinforcement learning," arXiv preprint arXiv:2009.14471, 2020.

[12] L. Zheng, J. Yang, H. Cai, M. Zhou, W. Zhang, J. Wang, and Y. Yu, "Magent: A many-agent reinforcement learning platform for artificial collective intelligence," in Proceedings of the AAAI Conference on Artificial Intelligence, vol. 32, no. 1, 2018.

[13] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al., "Human-level control through deep reinforcement learning," nature, vol. 518, no. 7540, pp. 529–533, 2015.

[14] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in Proceedings of the AAAI conference on artificial intelligence, vol. 30, no. 1, 2016.

[15] M. Hausknecht and P. Stone, "Deep recurrent q-learning for partially observable mdps," in 2015 aaai fall symposium series, 2015.

[16] J. Schulman, "Trust region policy optimization," arXiv preprint arXiv:1502.05477, 2015.

[17] J. Qi, Q. Zhou, L. Lei, and K. Zheng, "Federated reinforcement learning: Techniques, applications, and open challenges," arXiv preprint arXiv:2108.11887, 2021.

[18] Q. Yang, Y. Liu, Y. Cheng, Y. Kang, T. Chen, and H. Yu, "Federated learning," Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 13, no. 3, pp. 1–207, 2019.