Multi-Agent Battlefield Game with Federated Reinforcement Learning

Fardeen Hasib Mozumder, Soroush Famili, Ruben Lopez Villalobos, Ali Menati

Abstract

In this project, we are solving the game Battlefield using federated multi-agent reinforcement learning. This game is a competitive 12v12 game in an open field with walls of different landscapes. Our project consists of 2 phases. In the first phase, we explore three different deep reinforcement learning methods for multi-agent domains: deep Q learning (DQN), advance actor-critic (A2C), and proximal policy optimization (PPO). In the second phase, we introduce federated learning by averaging the models trained using a single algorithm on three different landscapes. The algorithm used in the second phase is the one that offered the best trade-off between performance, speed, and ease of use in the first phase. In our simulation, we evaluate the federated learning model against a single model in a new environment to quantify the advantage of federated learning when evaluating new and unknown environments compared to using a single multi-agent model.

Index Terms

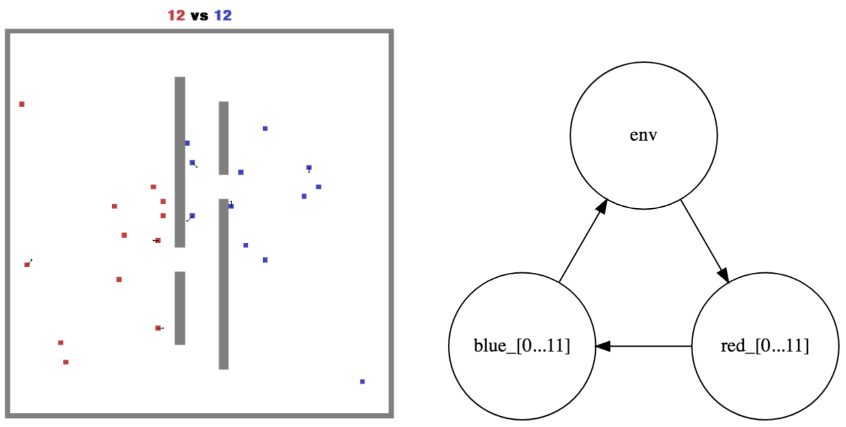
Reinforcement Learning, Federated Learning, Battlefield, Multi-Agent Learning.

I. Introduction

Reinforcement Learning (RL) has been one of the most active research areas in Machine Learning and Artificial Intelligence. In RL, the agent's goal is to achieve the maximum reward by learning how to interact with the environment. This usually happens through trial-and-error, where an RL agent learns how to map each state s to optimal actions a to achieve 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 even perform better than humans. In contrast to RL agents, human players can learn how to play Atari games somewhat successfully in a matter of minutes [1]. Atari games gained fame as a benchmark for reinforcement learning with the introduction of the Arcade Learning Environment (ALE) in 2012 [2]. The combination of reinforcement learning and deep neural networks then enabled RL algorithms to learn to play Atari games directly from images of the game screen using variants of the DQN algorithm [3], [4] and Actor-Critic algorithms [5]–[8]. In this project, we use reinforcement learning to play the game Battlefield. This is a competitive multi-agent 12v12 game in an open field with different walls, where each agent tries to maximize its reward by eliminating the players of the other team. We consider a setting where multiple local battlefield games are being played simultaneously in different environments. We use Federated Learning to combine the local models and create a global model without sharing local information. At the end of each training round, the global model is aggregated using the local models. The global model is then shared among all local models to be used as an initialization for the next training round, which speeds up training on new environments. In section II we introduce the environments, the agents, action and state spaces, and the simulator used in our project. We summarize our main contributions as follows:

1) To produce an agent capable of successfully playing Battlefield, we implement a variety of different RL algorithms such as Deep Q-Learning with Neural Networks (DQN), advanced Actor-Critic (A2C), and Proximal Policy Optimization (PPO). We then simulate the environments and train our algorithms to compare their performances.
2) We create various environments using environment wrappers to be used in the PettingZoo simulator. We randomly choose some of these new environments for training the federated learning and one for testing the federated model.
3) We apply Federated Learning (FL) to determine the neural network parameters of a global model in a decentralized fashion without sharing private information. We show that the resulting global model then generalizes better on average than any local model for any unseen environment.

4) We compare the trained global model with the non-federated models by simulating the federated setting. We generate new environments and test our new model on these unseen environments. We show that the federation generates a more robust model, which adapts to the new environment by obtaining higher rewards.



(a) Battlefield environment (b) Agent environment cycle

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

In Section II, the agents, environments, and simulator used in our project are introduced. In Section III, we explain how to create the new environments and split them up for training and testing. Section IV presents four different approaches used to solve the game, and compares their implementations. Our usage of federated learning is explained in Section V. The results and findings of our simulations are presented in Section VI and VII, where different approaches are investigated to improve the performance of each implementation. Finally, Section VIII provides our discussion and Section IX concludes the paper and provides potential future research based on our findings.

II. 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 [9]. 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. [10]. 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 only for their individual performance and not for the performance of their team members or the enemies, making it harder to implement team cooperation. Agents have slow auto regeneration, so a faster

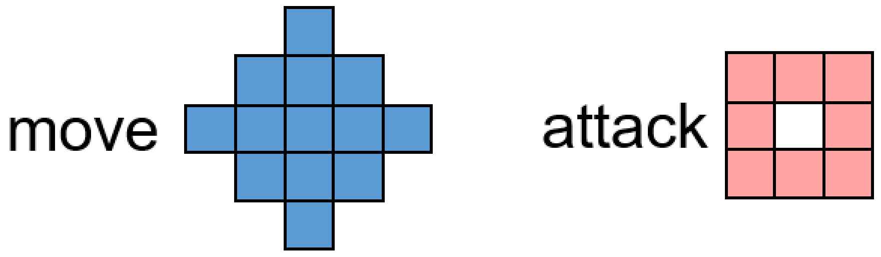


Fig. 2. Battlefield environment parameters, its observation space and action space.

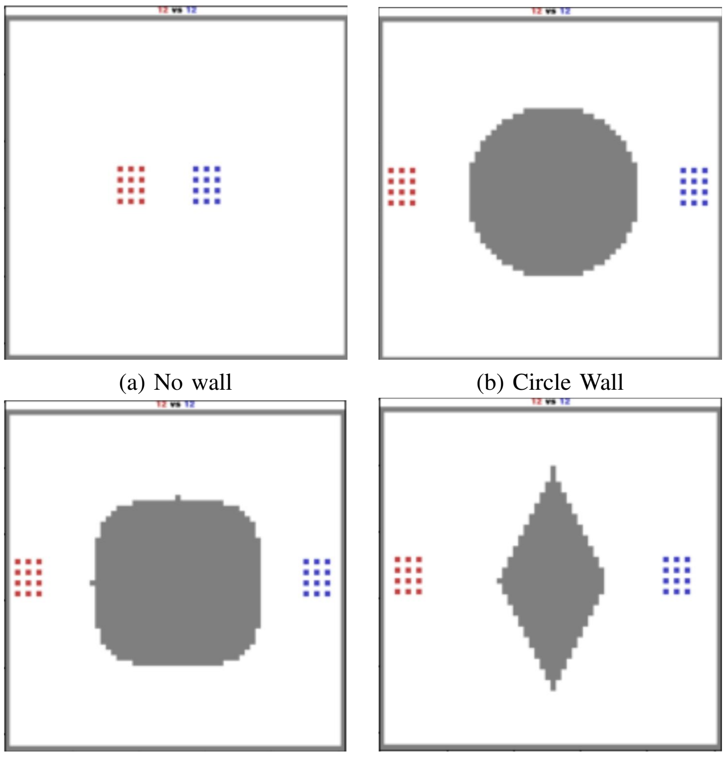
elimination of enemies will be preferred to a slow tactical approach. The environment consists of two teams of 12 units each. The units move in order, starting with unit 0 to 11 of the red team and followed by unit 0 to 11 of the blue team, as shown in Fig. 1(b). Each unit has 10 HP, with a recovery of 0.1 HP per turn. Each attack causes 2 HP of damage against enemies and an attack against another agent on their own team will not be registered. Like all MAgent environments, agents can either move or attack each turn, but not both. The positive reward consists of 5 points for killing an enemy and 0.2 points for attacking an opponent, while the punishment is a constant -0.005 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.

III. Simulation set-up

Each local model was trained on environments with slightly different wall setups, and the final testing was done on an environment the local models had not seen prior. In Fig. 3 below, the different wall setups used are shown. We randomized the wall setups used for training and testing 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, we drew p-norm unit circles 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 setup is, obviously, the L3-norm unit circle. These were chosen because each wall setup offers slightly different properties and take up different amount of areas. Additionally, the code for each environment only had to be minimally changed.



(a) No wall (b) Circle Wall

(c) L3-Norm Unit Circle Wall (d) Diamond Wall

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.

A. *Policy network*

All the implemented approaches used the same neural network architecture to avoid adding more variables in the comparison between approaches. The neural network uses a feature extraction layer, which converts the observation vector/matrix into a 1D tensor followed by two fully connected layers of 64 neurons, each followed by the output layer. The activation function is ReLu, as shown in Fig. 4. The library used is Baseline3 [11]. Baseline3 is a Python library that provides reliable implementations of reinforcement learning algorithms in PyTorch. It provides a clean and simple interface, allowing off-the-shelf state-of-the-art model-free RL algorithms. It is an open-source framework based on the OpenAI interface.

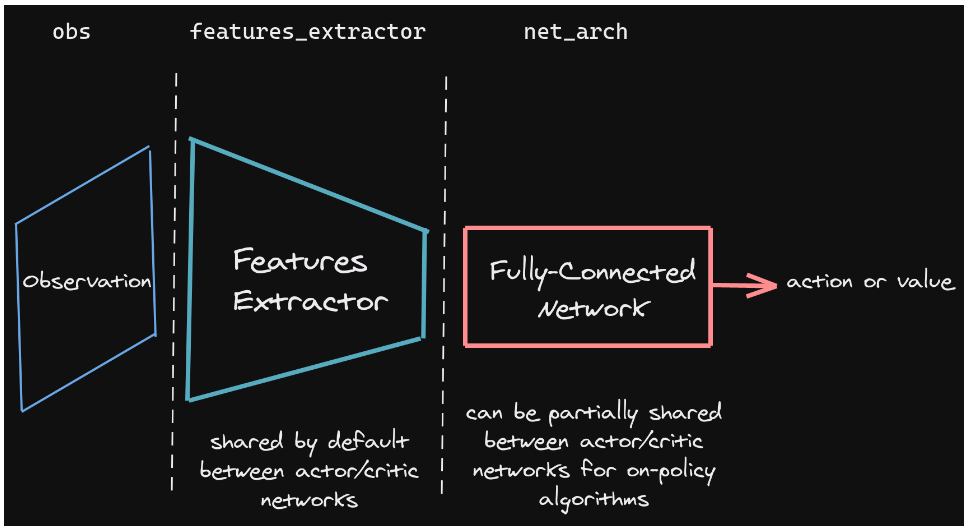

obs
features_extractor
net_arch
Observation
Features Extractor
Fully-Connected Network
action or value
shared by default between actor/critic networks
can be partially shared between actor/critic networks for on-policy algorithms

Fig. 4. Policy Network used for DQN, PPO, A2C.

IV. Implemented approaches for solving the game

A. *Deep Q-Learning with Neural Networks*

The Deep Q-Network (DQN) algorithm combines a powerful non-linear function approximation technique known as Deep Neural Network (DNN) with the Q-learning algorithm [3]. It is a multi-layered neural network that, for each given state, outputs a vector of action values. Two essential features of this algorithm are using a target network and an experience replay buffer. DQNs are capable of achieving human-level performance on a variety of different Atari games [12]. 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 [13]. 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 [14] modified the DQN architecture by adding a recurrent layer between and introducing Deep Q-Learning with Recurrent Neural Networks(DQRN). In our project, we use DQN based on Neural Fitted Q Iteration, which improves the use of past experience. We also use replay buffer, target network, and gradient clipping to avoid large updates. The battlefield game gives a matrix of the observations which are passed through the feature extraction into the neural network.

B. *Actor - Critic*

The Actor-Critic approach is an on-policy algorithm that maintains two function approximation mappings: an actor and a critic. The actor maps a particular state to an action policy, and the critic maps the same state to a predicted value. In our case, we implemented neural networks for the actor and critic. Then, just like other neural network-based policy gradient algorithms such as the DQN, the loss is computed, and the neural network parameters are updated using backpropagation. A2C is a specific actor-critic algorithm that uses the advantage function to determine the loss for the gradient ascent step. A2C is a synchronous, deterministic version of the A3C algorithm proposed by [5], allowing for larger batch sizes and more efficient use of GPU resources. The actor-critic approach used in our project is called synchronous advantage actor-critic, which is a deterministic and synchronous variant of asynchronous advantage actor-critic [5], but offers better performance in GPU. The method uses parallel workers doing the training independently but waiting for all the workers to finish to update the global model.

C. Proximal Policy Optimization

Proximal Policy Optimization is an on-policy reinforcement learning algorithm that is scalable, efficient, robust, and simpler than methods like Q-learning, vanilla policy gradients and trust region policy optimization (TRPO). Proximal Policy optimization algorithm (PPO) offers a more balanced approach between ease of implementation,tuning, and computation time. It outperforms the common online policy gradient methods when tested in Atari games [6]. PPO has been used in the multi-agent setups, such as the video game DOTA2. When tested against humans, it successfully defeated the DOTA2 esport world champion team. [15]. [16] compared PPO against off-policy baselines in the Starcraft and Hanabi challenges and the OpenAI particle-world enviroment, achieving good results while being comparable speed wise. The most common implementation of PPO is based on the A2C algorithm described above. The advantages of PPO is the use of clipping in the surrogate objective with clipped probability ratios, allowing for the estimate to be toward the lower bound. The clipping is done to avoid large updates in the model. Schulman, et al [6] tested PPO in atari games against synchronous advantage actor critic from Mnih et al. [5], trust region policy optimization, and other algorithms. It uses the same policy network as [5] for all the algorithms and uses tuned hyperparameters for each game. It was shown to be as accurate as ACER. We propose a novel objective with clipped probability ratios, forming a lower bound of the performance of the policy. To optimize policies, we alternate between sampling data from the policy and performing several epochs of optimization on the sampled data. Our experiments compared the performance of various different versions of the surrogate objective, and found that the version with the clipped probability ratios performs best. We also compared PPO to several previous algorithms from the literature. On continuous control tasks, it performs better than the algorithms we compare against. On Atari, it performs significantly better (in terms of sample complexity) than A2C and similarly to ACER, although it is much simpler.

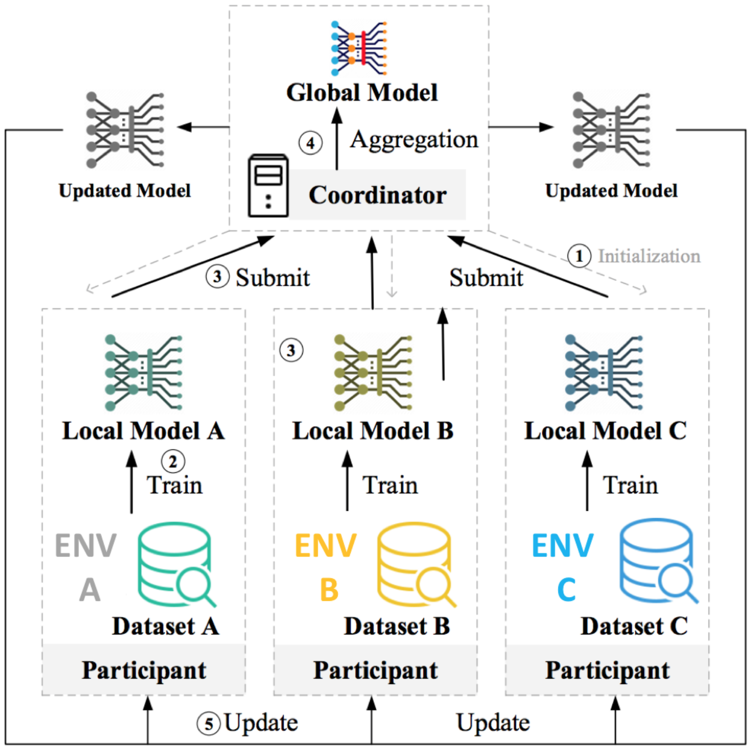


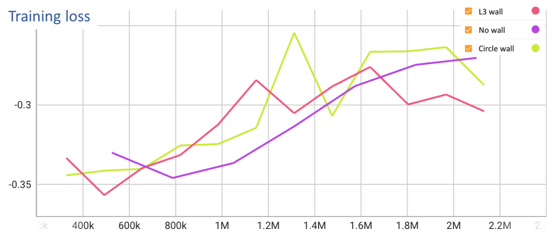
Fig. 5. Federated learning architecture and the client-server model [17]. Each client is solving a Battlefield game with a different environment and their model is being shared to update the global model.

V. Federated Learning

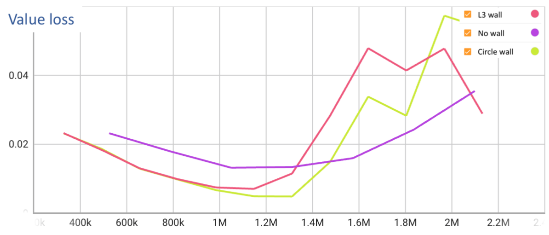
Federated learning is an emerging topic in machine learning, which aims to train models across multiple users without sharing their personal data samples [18]. In this algorithmic framework, it is possible to exchange the local models between parties during training, but not the data itself. This model is used when there are two or more parties that hope to jointly build a model to tackle similar tasks. Each party holds independent data and would like to use it for model training. It has been shown that the performance of the resulting model is very close to that of the ideal model established with all data transferred to one centralized party. Federated learning is implemented for various network architectures, including peer-to-peer and client-server models. In this project, we consider a client-server model where we have multiple Battlefield games being played in different environments. Our goal is to use a central server and build a global model for the optimal policy using local models of different environments. As is shown in Fig. 5, the coordinator acts as a central aggregation server, which initializes the local model and aggregates model updates from participants [19]. The participants train based on their local observations and share their personal model with the model aggregation mechanism to build the global model, which will be sent to all participants. This greatly enhances the training efficiency and the performance of the final models. The most common federated learning algorithm is federated averaging (FedAvg), which creates the global model by simply taking the average of the collected local models [19]. In this project, we design three different environments, and we are using the FedAvg algorithm to obtain the global model to be shared among them.

VI. Implementing RL Policies for Local Games: Algorithm Selection

Our first task is to select the policy gradient algorithm to train the final federated algorithm. We trained DQN, A2C, and PPO models on different environments and found that PPO outperformed the other policy gradient algorithms. Hence, we chose to exclusively use PPO for the final federated algorithm. In the final federated algorithm, we train PPO local models on different environments each round, use FedAvg to get a global model, then use the global model parameters as initial parameters for the local models for the following round. Overall, we did three rounds of local model training followed by FedAvg.

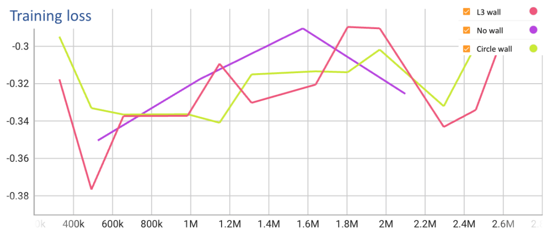


(a) Training loss

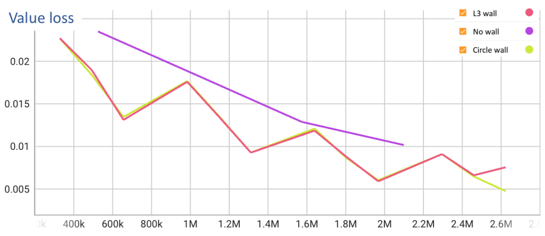


(b) Value loss

Fig. 6. Training loss and value loss for different environment using their local model with no federation.



(a) Training loss



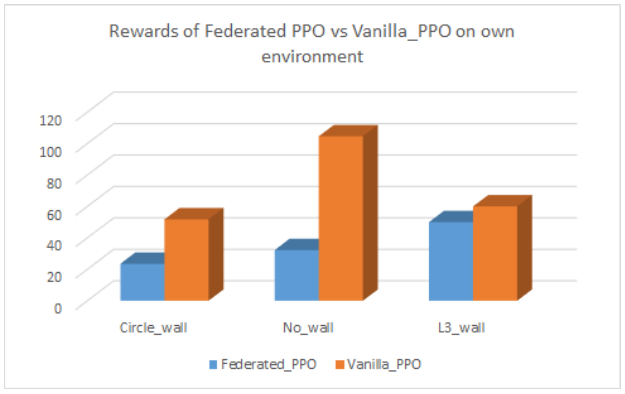
(b) Value loss

Fig. 7. Training loss and value loss for different environment using federated learning to get a global model as an initialization.

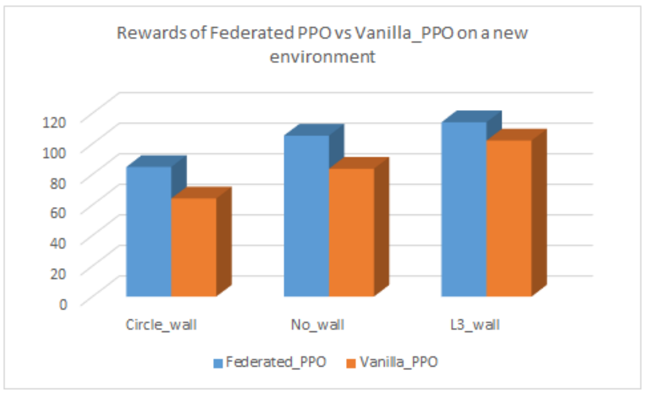
VII. Experimental Results

A. Training

For evaluating the training process, we utilized Tensorboard to find the optimal number of timesteps to train our agents. While training, Fig. 6 and Fig. 7 suggested that the non-federated learning process is faster than the federated learning process. The reason behind this is that, as the non-federated training process is done solely on their local environments, it converges to optimal policy faster (around 500K timesteps) than the federated models (around 1200K timesteps) which are trained using global shared parameters.

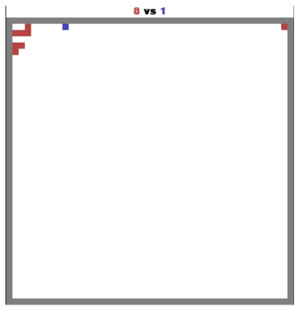


(a) Local environments used for training

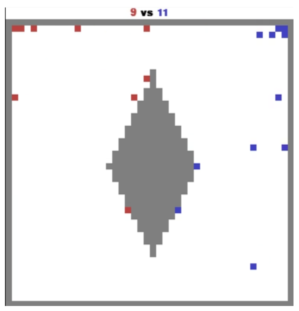


(b) New diamond wall environment

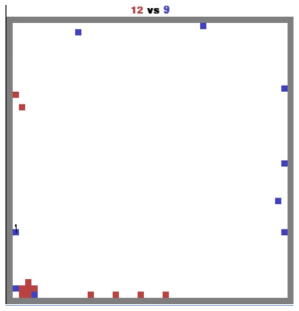
Fig. 8. Reward of the federated and vanilla (non-federated) PPO algorithms.



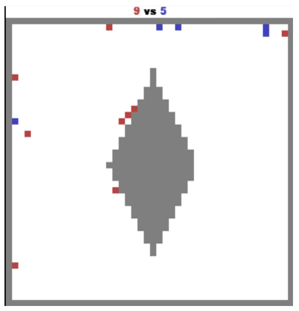
(a) Local environments with no federation



(b) New environment with no federation



(c) Local environments with federation



(d) New environment with federation

Fig. 9. 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.

B. *Testing*

For testing the performance of the federated models and non-federated models, we evaluated each of them on their local environments and a new environment. The results suggested that the non-federated models give us higher rewards than the federated models on their own environment, see Fig. 8 and Fig. 9 below. As the non-federated models are trained on their local models in oppose to federated models, which are trained by global parameter sharing, non-federated outperforms federated models on local environments. On the other hand, the federated models outperformed their respective non-federated models on a completely new environment which was expected as they were trained to perform in more generalized settings. The results are shown in Fig. 8 and Fig. 9 below.

VIII. Discussion

A. *Novelty*

To our knowledge, this project was the first attempt at using Federated Learning to improve the robustness of Multi-Agent algorithms. Although the Battlefield game was very simple, our results show that using these two types of Reinforcement Learning algorithms in combination could have promising results in other problems. Likewise, we have not seen any use of federated learning where the local models were trained on fundamentally different environments, like what we did in this project. We argue that using different environments for local models and using federated learning in combination with Multi-Agent algorithms was an original and unique approach.

B. Hardness

Although the PettingZoo environment is easy to use, doing custom modifications was difficult, adding a layer of complexity to the creation of each of our environments, even though we used similar landscapes. The reinforcement learning algorithms were wrapped using parallel processing, which improves performance but negates the ability to analyze each training episode and record useful information like the rewards-score. Another added difficulty was extracting local parameters from the trained model, then passing them to the global model to implement the federated learning. Lastly, as our project was multi-agent reinforcement learning, the training process was extremely time-consuming. Even using TAMU-HPRC (10 cores, 64 GB RAM) somewhat limited our ability to train our model on more environments. The federated implementation had the added difficulty of Extracting local parameters from the trained model Sharing the global model to be used as initialization

IX. Conclusion and Future Work

We have presented a test to show that federated learning can improve the success of reinforcement learning algorithms in games when the landscape is different than the one used for training the model. Our tests were of limited size, with only three different maps for training and only one in testing. Future work will improve increasing the size of the training and also the testing maps and reduce the similarity of the maps since two of our maps were similar. Testing different methods of merging the model is of consideration, we use a simple average, but we can use the median between each element of the output matrix or use a more complex method of aggregation of models. In the future, we would like to see how successful this federated learning and multi-agent learning combination works in other settings. We can imagine that any multi-agent problem that has unpredictable environments can benefit from the robustness that federated learning provides. For example, if you want to train a flock of drones to collectively achieve a task, the number of potential weather conditions might make it impractical to train the drones in every weather condition possible. Using our approach would minimize the number of weather conditions that the drones would have to be trained in to achieve a final model which can be successfully applied in any environment.

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*, 2013.

[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 *Thirty-second AAAI conference on artificial intelligence*, 2018.

[5] 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.

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

[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] 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.

[10] 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.

[11] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, 2021.

[12] 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.

[13] 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.

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

[15] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[16] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative, multi-agent games," *arXiv preprint arXiv:2103.01955*, 2021.

[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.

[19] 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.