CS230

Cooperative-Competitive Multi-Agent Learning in Soccer Environments

Christopher Covert
cwcovert@stanford.edu

Cameron McMillan
cmac12@stanford.edu

Patipan Pipatpinyopong
pipat001@stanford.edu

Abstract

Inspired by multi-agent cooperation, we are investigating the use of multi-agent deep reinforcement learning networks to play simple cooperative games. This project utilizes a simulated soccer environment developed by Unity to test a suite of deep reinforcement learning algorithms against each other in a simple game benchmark. In the 2 vs 2 player format, where each team is equipped with a striker (offensive agent) and goalie (defensive agent), we explore how agents can learn to be simultaneously collaborative and competitive. To avoid the influence of consecutive examples' strong correlations and interdependence, a randomized experience replay scheme was used to train the networks. We explored the training result between Deep Q-Networks (DQN) and Proximal Policy Optimization (PPO), each with small variations during training.

1 Introduction

Motivated by the use of deep reinforcement learning in multi-agent control, our team is applying deep-RL techniques to train agents to learn collaborative tasks. To do so, we are working in a Unity soccer environment with two agents on each team. A team consists of a goalkeeper and striker were trained with the goal of achieving "human-level" control. This work lends itself to related multi-agent settings such as team pursuit/evasion, formation control, and coordinated object manipulation.

The input to our system uses experience replay which stochastically samples a mini-batch from a buffer of previous state-action-reward outcomes to train the deep network. Taking this sample can stabilize the input dataset by decoupling the influence of the immediately preceding samples from the current test. We then use a series of modified DQN and PPO models to output an action-state pair sequence that simulates the game of soccer between two teams of two agents. The results of this game output a reward value that is used to train further iterations during the training round.

Success is measured by the cumulative number of wins, draws, and loses of a team. The different models will be trained and compared against each other in a series of consecutive matches.

2 Related work

Deep Q-Network (DQN)

Deep Q-learning is a model-free reinforcement learning techniques that learns a Q-function used to find the best action of a given state. This function is approximated through a deep neural network, however, convergence issues may arise. To improve upon DQN, (1) uses experience replay and reward clipping to stabilize multi-agent collaborative formation control from sensory input. In a pursuit/evasion environment, (3) demonstrates the capabilities of a Multi-Agent Deep Q-Network

CS230: Deep Learning, Winter 2019, Stanford University, CA. (LateX template borrowed from NIPS 2017.)

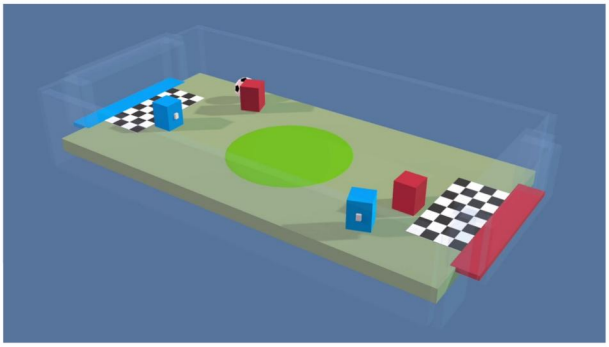

Figure 1: Soccer Simulation Environment in Unity

(MADQN) through centralized training process. The work in (4) applied leniency to overcome outdated stored experience and found that Lenient-DQN (LDQN) models converge to optimal policy quicker than modified Hysteretic-DQN (HDQN) algorithm. The authors in (5) showcases the power of DQN on Atari 2600 games by comparing Deep-RL results with expert human-level performance. (6) and (7) build from the Atari 2600 results by improving the over-estimations of (5) by implementing Double DQN and Dueling DQN models, respectively and demonstrated improved performance. (8) adds Prioritized Experience Replay to the Double DQN algorithm which changes the sampling distribution when there is a large error in prediction.

Proximal Policy Optimization (PPO)

Used to showcase that learning-based algorithms can play actor-critic games, (9) uses PPO against the Atari 2600 benchmarks to show that PPO outperforms standard online policy gradient methods and is a suitable baseline for Deep-RL tasks as a result of its balance between sample complexity and algorithm simplicity. From this work, the authors in (10) apply TRPO to the Atari 2600 benchmark tests and found that it results in monotonic improvements as compared to human and baseline DQN results. This work was further extended by the authors of (2), which used a modified policy-scaled Trust Region Policy Optimization (PS-TRPO) method to simulate cooperative multi-agent control in the pursuit/evasion game.

Unity Deep-Reinforcement Learning

As for the environment, the developers at Unity published (11), which outlines the RL branch of the Unity game engine and their creation of a general platform for intelligent agents. From this work, (12) implements a baseline PPO model in the soccer Deep-RL Environment from which served as the inspiration of our PPO baseline. This previous attempt was limited to running PPO, and was the reason our team pursued a comparison with several modified DQN models.

3 Dataset and Features

We have a Unity-based environment which is a bounded rectangular soccer field as seen in Figure 1. Our baseline is a 2 vs 2 setup, each team consisting of a single goalkeeper and a striker agent. Each role has a defined reward function summarized in Table 1. Each game lasts until the maximum number of steps is reached or one team scores. The penalty on the striker for existing rewards the striker for ending games quickly to motivate the striker to score quickly. The opposite is true for the goalie because longer games mean that neither team has scored.

Table 1: Agent Reward Function Definitions

Agent	Event		
	Ball Enters Opponent's Goal	Ball Enters Own Team's Goal	Existing
Striker	+1	-0.1	-0.001
Goalie	+0.1	-1	+0.001

The action space is slightly different between the striker and keeper. The striker has 6 possible actions: forward, backward, left, right, clockwise rotation, and counterclockwise rotation. The keeper has only four possible actions: forward, backward, left, and right. As for the state space, each agent obtains a 336 element observation vector, that corresponds to 7 rays within the 180 degree view from the front of the agent, with 6 sets of rays stacked in the vertical direction. Each ray consists of 8 values: binary classification of seven possible object types and the object's distance from the agent. The action of each agent in one time step is then communicated directly to the Unity environment based on the policy as a function of the current state.

The dataset for training this model utilizes an experience replay method, meaning that the input set is generated at the time of training.

4 Methods

To provide an intelligent baseline for our DQN tests, we leveraged a repository (12) that has implemented PPO (see Algorithm 1) to train agents in the soccer environment that we're working with (9).

Algorithm 1: Proximal Policy Optimization

Input: initial policy parameters θ_0, initial KL penalty β_0, target KL-divergence δ
for $k = 0, 1, 2, ...$ **do**
  Collect set of partial trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
  Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
  Compute policy update

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}_{\theta_k}(\theta) - \beta_k \bar{D}_{KL}(\theta||\theta_k)$$

  by taking K steps of minibatch SGD (via Adam)
  if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \geq 1.5\delta$ **then**
    $\beta_{k+1} = 2\beta_k$
  else if $\bar{D}_{KL}(\theta_{k+1}||\theta_k) \leq \delta/1.5$ **then**
    $\beta_{k+1} = \beta_k/2$
  end if
end for

While the model is a standard PPO implementation, tuning of the hyperparameters was necessary to achieve desired learning parameters. The following hyperparameters were used to train the PPO baseline: batch size is 32, epsilon is 0.1, gamma is 0.995, 2 hidden layers, 256 neurons in the first layer, 128 neurons in the second, learning rate of 8e-6 for the goalie, and learning rate of 1e-4 for the striker.

As a comparison point to a baseline PPO model, we explored various DQN implementations with different reward structures to see how they performed against each other and the PPO benchmark. The DQN algorithm with experience replay that was implemented as described in Algorithm 2.

As for hyperparameters used to train the baseline DQN model, we used a replay buffer size of 1e5, minibatch size of 64, discount factor (γ) of 0.99, soft update of target parameter (τ) of 1e-3, learning rate of 5e-4, and update iteration count of 4 sessions.

As for the structure of the DQN model, it has three linear fully-connected layers of (the two hidden-layers having 256 and 128 units, respectively). Both layers also have Relu activation functions. This is a standard representation of a DQN.

From the best baseline of DQN that our team was able to train, we varied the reward structure into four versions that are explained in further detail in the following section.

5 Experiments/Results/Discussion

To evaluate the algorithm performance, we first trained a baseline DQN model and a PPO model using the reward structure summarized in section 4. These trained models were evaluated against a team that acts completely at random.

Algorithm 2: Deep Q-Network with Experience Replay

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D
    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Then we altered the reward structure and retrained the DQN models to see how that affected the test performance of the algorithm. First, a reward of +0.3 was added to strikers when they "hit" the ball. Additionally, a penalty of -0.03 was added whenever the striker cannot observe the ball in its observation space. This was intended to help the striker locate and approach the ball, then explore the rewards based on state-action combination near the ball. This change in reward was applied to trained versions 1, 2, and 3. Additionally, we introduced the "modified" versions of each of these strikers by removing the penalty of not observing the ball. Since the goalie was limited to the smaller state space and has a smaller action space, no intermediate reward was added.

On top of introducing intermediate rewards, we also varied the penalty on the striker when the team gets scored on. This was intended to explore behaviors that may emerge. For Version 1, the striker receives the original reward outlined earlier where scoring a goal rewards +1 and -0.1 penalty, corresponding to a "neutral" striker that focuses on scoring and may play some defense. Version 2 corresponds to an "aggressive" striker that receives reward of +1 for scoring and 0 penalty if the team gets scored on. In this case, we expected the agent to try to do whatever it can to score a goal. Lastly, Version 3 corresponds to a "defensive" striker that gets reward of +1 for scoring and penalty of -0.6 if it gets scored on.

Each trained model was tested against one another and against agents that take random actions. Table 2 summarizes the win and not-lose percentages after testing for 500 games. Overall, we see that the DQN algorithm performed better, specifically Version 1 modified and Version 2. Both models fared well against random players, however, at 75-80% win rate, we believe that there is room for improvement. First, we observed an interesting behavior where all versions of DQN agents would be "stuck" without commanding any actions in certain situations. We suspect this behavior as a result from not learning the entire Q function. An extreme example of this is our baseline DQN model where the striker does not move at all and constantly tries to move backwards. This explains the not lose result of 100% when it was tested against itself.

After testing Version 3 against, random and baseline DQN, we observed poorer performance and did not continue to test the model. It became apparent that a defensive striker is not a good strategy within this environment. Similarly, when we trained PPO models with intermediate ball reward, we actually saw worse performance and did not complete the entire test set in the interest of time. In addition, we saw that the PPO striker consistently scores on its own goal.

Overall, the goalie performed well across all trained models. One interesting observation was that if the ball comes to rest in the goalie box where neither strikers could enter, the goalie would choose to do nothing and let the time run out, resulting in a draw.

Based on the success of Versions 1 and 2 of DQN models, we applied the same learning technique in a 3 vs. 3 agent environment which includes one goalie and two strikers on each team. We expected the intermediate reward to help with training, however, we were not able to get the model to converge.

Table 2: Evaluation Result for 2 vs. 2 Environment

	Win (W) and Not Lose (NL) % Against:						
Model	Random	DQN_{base}	DQN_1	DQN_1^{mod}	DQN_2	DQN_2^{mod}	PPO_{base}
DQN_{base}	W: 13% NL: 81%	W: 0% NL:100%	W:18 % NL: 32%	W: 14% NL: 41%	W: 5% NL: 38%	W: 10% NL: 59%	W: 21% NL: 55%
DQN_1	W: 66% NL: 70%	W: 68% NL: 82%	W: 31% NL: 62%	W: 23 % NL: 49%	W: 28% NL: 48%	W: 37% NL: 57%	W: 63% NL: 68%
DQN_1^{mod}	W: 79% NL: 85%	W: 59% NL: 86%	W: 51% NL: 77%	W: 40% NL: 63%	W: 41% NL: 63%	W: 45% NL: 75%	W: 77% NL: 81%
DQN_2	W: 77% NL: 81%	W: 62% NL: 95%	W: 52% NL: 72%	W: 37% NL: 59%	W: 41% NL: 62%	W: 48% NL: 76%	W: 70% NL: 77%
DQN_2^{mod}	W: 53% NL: 74%	W: 41% NL: 90%	W: 43% NL: 63%	W: 25% NL: 55%	W: 24% NL: 52%	W: 39% NL: 59%	W: 49% NL: 71%
PPO_{base}	W: 53% NL: 70%	W: 45% NL: 79%	W: 32% NL: 37%	W: 19% NL: 23%	W: 23% NL: 30%	W: 29% NL: 51%	W: 46% NL: 58%

6 Conclusion/Future Work

Multiple variants of DQN agents and PPO agents were trained in the 2 vs. 2 soccer environment. Initially, the PPO model was observed to perform much better when tested against random agents and against one another. Intermediate rewards were then introduced to help with training DQN models and it showed significant improvement. One interesting result seen is that it is not clear whether the penalty on the striker when it does not see the ball would help the agent perform better or not. We were able to see that the more aggressive strikers performed better in this game overall. We also found that introducing intermediate reward actually hurt the performance when training PPO models.

With more time and resources, interesting future work includes further fine tuning of the reward structure and general hyperparameters to reach near 100% not lose rate against randomly acting agents. As mentioned in the results, the goalie tends to "hog" the ball when it can, which doesn't quite resemble cooperative behavior. Training a more "aggressive" goalie may overcome this behavior. Furthermore, we would like to expand the problem to include more agents, less environmental boundaries by removing walls, more complex agent roles, and have different models trained against each other. Because all models in this work were trained against agents acting completely at random, we expect that training against skilled agents would help improve learning performance. Lastly, there are other deep reinforcement learning methods to be explored such as double DQN, dueling DQN, and imitation learning.

7 Contributions

The code for this project can be found at https://github.com/ppipat/Multi-Agent-Soccer. The PPO code is in the PPO branch of this repository.

Christopher Covert - Developed Unity models, created Unity framework for 3 vs 3 environments, and drafted the initial poster and report for final submission.

Cameron McMillan - Setup and tuned PPO models, setup PPO and DQN running on Amazon, and ran algorithm performance tests.

Patipan Pipatpinyopong - Setup DQN models for this environement. Applied intermediate reward for training.

References

[1] Conde, Ronny, José Ramón Llata, and Carlos Torre-Ferrero. "Time-varying formation controllers for unmanned aerial vehicles using deep reinforcement learning." arXiv preprint arXiv:1706.01384 (2017). https://arxiv.org/ftp/arxiv/papers/1706/1706.01384.pdf

[2] Gupta, Jayesh K., Maxim Egorov, and Mykel Kochenderfer. "Cooperative multi-agent control using deep reinforcement learning." International Conference on Autonomous Agents and Multiagent Systems. Springer, Cham, 2017. http://ala2017.it.nuigalway.ie/papers/ALA2017_Gupta.pdf

[3] Egorov, Maxim. "Multi-agent deep reinforcement learning." (2016). http://cs231n.stanford.edu/reports/2016/pdfs/122_Report.pdf

[4] Palmer, Gregory, et al. "Lenient multi-agent deep reinforcement learning." Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems. International Foundation for Autonomous Agents and Multiagent Systems, 2018. https://dl.acm.org/citation.cfm?id=3237451

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., and Petersen, S. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529. https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf

[6] Van Hasselt, H., Guez, A., and Silver, D. (2016, March). Deep reinforcement learning with double q-learning. In Thirtieth AAAI Conference on Artificial Intelligence. https://arxiv.org/pdf/1509.06461.pdf

[7] Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581. https://arxiv.org/pdf/1511.06581.pdf

[8] Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. arXiv preprint arXiv:1511.05952. https://arxiv.org/pdf/1511.05952.pdf

[9] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347. https://arxiv.org/abs/1707.06347

[10] Schulman, J., Levine, S., Abbeel, P., Jordan, M. I., and Moritz, P. (2015, July). Trust Region Policy Optimization. In Icml (Vol. 37, pp. 1889-1897). http://proceedings.mlr.press/v37/schulman15.pdf

[11] Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). Unity: A General Platform for Intelligent Agents. arXiv preprint https://arxiv.org/pdf/1809.02627.pdf. https://github.com/Unity-Technologies/ml-agents

[12] Marcello Borges GitHub Repository. https://github.com/marcelloaborges/Soccer-PPO