Handling Predator-Prey Task with Multi-Agent Deep Deterministic Policy Gradient(MADDPG)

Joel Schnubel [1] **Senjuti Dutta** [1]

Abstract

This report deals with the implementation of the Multi-Agent Actor-Critic (MADDPG) algorithm in combination with grounded compositional language in multi-agent populations. MADDPG is a reinforcement learning algorithm that has been shown to be effective for training agents in cooperative and competitive environments. In this setting, agents are able to learn to communicate with each other in order to solve complex tasks.

1. Introduction

Multi-agent reinforcement learning (MARL) is a type of reinforcement learning (RL) in which multiple agents learn to interact with an environment and each other to achieve a common goal. In contrast to single-agent RL, where the agent learns to interact with the environment in isolation, MARL agents must learn to coordinate their actions and behaviors in order to maximize their collective reward. Multi-agent self play has recently been applied to solve many reinforcement learning problems and has been proven to be an useful training paradigm(Silver et al., 2016), (Sukhbaatar et al., 2017). Scaling these reinforcement learning scenarios with the natural environment using multiple agents can prove useful to to building artificial intelligent systems to build a better interaction between humans and each other. However, traditional practices like Q-learning and policy gradient can often fail to successfully handle multi-agent cases.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) is a reinforcement learning (RL) algorithm that extends Deep Deterministic Policy Gradient (DDPG) by learning a separate actor-critic model for each agent in a multi-agent environment. The MADDPG algorithm is similar to the DDPG algorithm, but it has several extensions to handle multi-agent interactions. Each agent interacts with the environment and collects a tuple of experience, which includes the agent's action, the action of the other agents, the next state, and the reward. The critic for each agent is updated using the Bellman equation. The critic predicts the value of the current state-action pair for each agent. The Bellman equation is used to update the critic's prediction to match the actual value of the state-action pair. The actor for each agent is updated using the deterministic policy gradient theorem. The actor learns a policy that maps states to actions such that the expected return is maximized. The motivation to consider the topic of implementing the Multi-Agent Actor-Critic (MADDPG) algorithm for Emergence of Grounded Compositional Language was to understand the emergence of language from zero in a Predator Prey task. As grounded compositional language can be a powerful tool in reinforcement learning, although it takes more time to establish useful communication between the agents.

The Goal is to explore the usefulness of Grounded Compositional Language in a Multi-Agent reinforcement learning task.

2. Related works

Recent research in 2016-17(Sukhbaatar et al., 2016),(Foerster et al., 2016),(Mordatch & Abbeel, 2018) in reinforcement learning has focused on developing methods for agents to learn cooperative communication protocols that can be used to solve various tasks. However, these methods often rely on a dedicated communication channel that is specifically designed for this purpose. This means that they may not be applicable in real-world scenarios where agents need to communicate through noisy or unreliable channels. The importance of explicitly modelling the decision making process of other agents has also been recognized by RL studies (Boutilier, 2013),(Chalkiadakis & Boutilier, 2003). In our approach, the main idea for the training Algorithm has been inspired from (Ryan Lowe, 2017), and the setting and the network structure is similar and motivated from a related previous work done in (Igor Mordatch, 2017). There has been previous attempts at handling predator-prey tasks with MADDPG. In 2018, () described a cooperative predator-prey task with multiple robots. The robots are tasked with

[1]Handling Predator-Prey Task with Multi-agent Deep Deterministic Policy Gradient(MADDPG). Correspondence to: Firstname1 Lastname1 <first1.last1@xxx.edu>, Firstname2 Lastname2 <first2.last2@www.uk>.

tracking and capturing a target object. The results show that MADDPG is able to achieve successful coordination between the robots and capture the target object.

An interest in pragmatic view of language understanding has been longstanding (Austin, 1962) and has recently argued for in (Gauthier & Mordatch, 2016),(Lake et al., 2016). Pragmatic language use has been proposed in the context of twoplayer reference games (Golland et al., 2010) and (Andreas & Klein, 2016) focusing on the task of identifying object references through a learned language. (Winograd, 1973) and (Wang et al., 2016) ground language in a physical environment and focusing on language interaction with humans for completion of tasks in the physical environment.

3. Background

3.1. Setting

In our chosen continuous 2 dimensional environment, agents are capable of moving freely within predefined boundaries. Landmarks are randomly placed throughout the environment to serve as obstacles for the agents.

These landmarks are defined by their positions $[x, y]$ and a fixed radius of 30 units. To mitigate overfitting, we designed multiple environments with alternated landmark positions. For our experiments, we utilized three different environments, each containing five landmarks.

The physical state of an agent at time step t comprises its position, velocity, and a compact vector describing physical interaction forces and in our case, color distinguishes predators, prey, and landmarks.

The physical state of an agent can be represented as follows:

$$\begin{bmatrix} x \\ y \\ v \\ f(x,y,e) \\ c \end{bmatrix}^{t} = \begin{bmatrix} x + v\cos(\theta)\Delta t \\ y + v\sin(\theta)\Delta t \\ v \\ f(x,y,e)\Delta t \\ c \end{bmatrix}^{t-1}$$

Here, x and y represent the agent's position in the 2D environment, v is the current velocity, θ is the current angle, c is the color represented by $[r, g, b]$, and $f(x, y, e)$ returns a vector:

$$\begin{bmatrix} \text{hit}(x,y,\text{prey\_agent}) \\ \text{hit}(x,y,\text{predator\_agent}) \\ \text{hit}(x,y,\text{landmark}) \end{bmatrix}^{t}$$

The `hit` function returns 0 or 1, indicating whether the agent is colliding or not with the respective object type.

Agents (prey and predator) also possess additional attributes, including a private rotation angle θ, a communication vector of size s (set to 32), a radius r (set to 15), and a private goal vector.

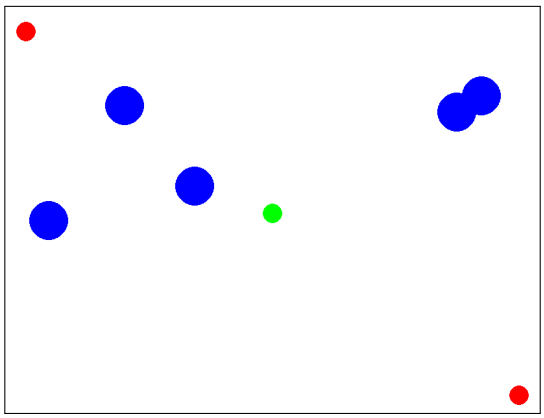

Figure 1. Example of a setting
Blue circles are the landmarks;
red circles are predator agents;
green circles are prey agents

The goal vector for prey at position i is given by:

$$g_i = \begin{bmatrix} d(\text{prey}_i, \text{predator}_0) \\ d(\text{prey}_i, \text{predator}_1) \\ \vdots \\ d(\text{prey}_i, \text{predator}_n) \end{bmatrix}$$

The reward for prey i is calculated as the sum of the goal vector divided by a normalization factor n, with an additional penalty term if the prey hits a predator:

$$r_i = \frac{\sum g_i}{n} - h \cdot \text{hit-predator}(i)$$

The reward for predator i is calculated similarly, with the reward being the negative sum of the goal vector divided by n, and an extra reward if the predator hits a prey:

$$r_i = -\frac{\sum g_i}{n} + h \cdot \text{hit-prey}(i)$$

Here, h is a predefined constant set to 10 in our case, and n is a normalization factor, arbitrarily set to 1000 based on the dimensions of our environment (width: 800, height: 600).

3.2. Networks

In our experiments, we utilize two different types of networks: a simple neural network and one inspired by the design proposed in (Igor Mordatch, 2017). To facilitate these networks, we need to create an agent-specific input vector.

3.2.1. Network Inputs

To construct the input for the network, we generate a physical observation vector $o_i(e)$ for each agent in the environment e. This vector is assembled by appending the agent's

own state vector to the physical states of all other entities (including landmarks) in the environment relative to the agent's position. Additionally, we gather all communication streams c for each agent type, which are constructed from the private communication vectors and the private goal of agent i:

$$\begin{bmatrix} {x_i}^1, \ldots, {x_i}^n \\ c_1, \ldots, c_n \\ g_i \end{bmatrix}$$

where ${x_i}^j$ is the state vector of entity j relative to the position of agent i.

3.2.2. Network Architecture

For the Simple Actor Neural Network, a simple architecture is employed, concatenating the physical observation vector of agent x_i with its private goal vector g_i as the input. No communication between agents or private memory buffer is utilized in this design. The network comprises three layers with Rectified Linear Unit (ReLU) activation functions, and dropout is applied after each layer with $p = 0.1$, except for the output layer. The output of this network is the corresponding action, represented as a vector of size 2 with $\begin{bmatrix} \theta \\ v \end{bmatrix}$ where θ is normalized to the range $[0, 2\pi)$, and v is constrained by the maximum velocity of the respective agent.

The other network, inspired by the Policy Architecture in (Igor Mordatch, 2017), uses the complete physical observations of agent i, denoted as $o_i(e)$. The network is initialized to accommodate any number of communication streams and states, with the size of communication streams, states, and goal vectors being variable.

For each communication stream, a private memory vector of size 32 is initialized, as well as for the output layer. The physical observations and communication streams are processed separately in two different modules, each consisting of two linear layers with ReLU activation functions and dropout of 0.1. For the communication module, the respective memory vector is carried along as input and output of the module with its corresponding stream. The weights between the communication processing and physical observation modules are shared.

The outputs of the processing modules are pooled with a softmax operation into feature vectors for communication and physical observation streams, respectively. The pooled features, agent's private goal vector, and the final layer memory vector are passed to the final processing module that outputs distribution parameters and the memory vector $[\psi_u, \psi_c, \Delta m]$. The final output is obtained as $u = \psi_u + \epsilon$, and $c \sim G(\psi_c)$, where ϵ is a zero-mean Gaussian noise, and G is a Gumbel-Softmax Estimator. The output vector u can be separated into the new θ and v values. After a forward pass, all the memory vectors are updated as follows: $m^t = \tanh(m^{t-1} + \Delta m^{t-1} + \epsilon)$.

The corresponding critic for both network types is the same: a simple 2-layer network with one ReLU activation function between the layers. It takes as input a state and action and returns the value of that state-action pair.

The hidden dimension of all the networks is set to 128.

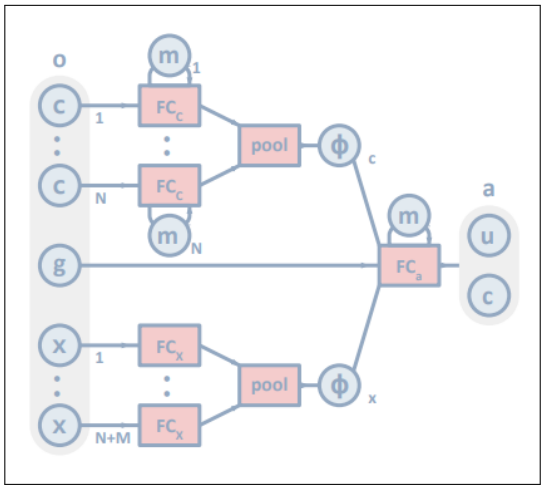

Figure 2. Overview of the policy architecture taken from (Igor Mordatch, 2017) Figure 3.

3.3. Algorithms

Algorithm 1 The Algorithm is motivated from the one in the (Ryan Lowe, 2017) with some extra steps In all of our experiments, we use the Adam optimizer with a learning rate of 0.01 and $\tau = 0.01$ for updating the target networks. γ is set to be 0.95. The size of the replay buffer is 10^4 and we update the network parameters after every 120 samples added to the replay buffer. We use a batch size of 256 episodes before making an update. We train on 3 random seeds for the environment.

3.4. Tables

Table 1 shows the total catches and catch rates between predator vs prey.

4. Conclusion

It is crucial to note that the field of multi-agent reinforcement learning tasks is continuously evolving, with researchers developing new techniques to improve model performance. The proposed Multi-Agent Deep Deterministic Policy Gradients (MADDPG) algorithm has shown promising results, as demonstrated in the plots of predator and prey interactions with communication streams (refer to the appendix).

However, it is worth mentioning that the simple 2-layer neural network, without any communication streams, outperforms the Policy Architecture in (Igor Mordatch, 2017). Several reasons could contribute to this observation. Firstly, the Policy Architecture is more complex, involving a higher number of weights and parameters. Given our training duration of 300 episodes, the network may not have had sufficient time to fully demonstrate its potential to develop an abstract compositional language from grounded experience between the predators. Another factor could be the chosen environment – a relatively small one with fewer landmarks. In this simpler setting, the straightforward neural network might be adequate for achieving satisfactory results. Due to computational constraints, we did not conduct longer training sessions, in contrast to (Ryan Lowe, 2017), where they trained for 25,000 episodes with a memory buffer of 10^6 and batch size of 1024 for the same task.

In future experiments, increasing the number of episodes and the memory buffer size could yield better results. Additionally, implementing features such as a perception radius or an Auxiliary Prediction Reward may further encourage the emergence of an abstract compositional language between the agents. The groundwork has been laid, offering exciting possibilities for future exploration.

Algorithm 1 Main Algorithm

init create Game g
init create n different environments e with m different landmarks
Input Prey and Predator Agents with networks
for epoch = 0 **to** epochs **do**
  done = 0
  for episode = 0 **to** max episode length **do**
    for each agent i select action $a_i = \mu_{\theta_i}(o_i)$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state x'
    Store (x, a, r, x0) in replay buffer D
    x $\leftarrow$ x'
    for agent i = 1 **to** N **do**
      Sample a random minibatch of S samples(x^j, a^j, r^j, x'^j) from D
      Set $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, \ldots, a'_N)|_{a'_k = \mu'_k(o_k^j)}$
      Update critic by minimizing the loss$L(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\mu}(x^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:
      $\nabla_{\theta_i} J \approx$
      $\frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i} Q_i^{\mu}(x^j, a_1^j, \ldots, a_i, \ldots, a_N^j)|_{a_i=\mu_i(o_i^j)}$
    end for
    Update target network parameters for each agent i:
    $\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$
  end for
  chose new environment e
  reset positions of agents
end for

Table 1. catches and catch rates along the different models.

PREDATOR VS PREY	TOTAL TRAIN CATCHES	TRAIN CATCH RATE	TOTAL TEST CATCHES	TEST CATCH RATE
COMS VS COMS	9	0.06	9	0.09
COMS VS SIMPLE	2	0.02	5	0.05
SIMPLE VS SIMPLE	18	0.12	20	0.2

Reproducibility

You can run the training or testing algorithm by simply executing the world.py file. There are several parameters to chose:

If you set load_models = True, then below the models with the name model_name will be loaded from the models folder are loaded and their parameters are updated accordingly.

If you set load_models = False, then the models will be trained or tested based on model_name and stored as model_name with their respective type (prey or predator with it's index)

depending on the value of test_mode the models are either trained or tested

simple_Prey and simple_Predator are to choose wheater to use the Simple Actor Neural Network or the Policy Architecture describe in the Networks section. These can also be seen in models.py

All the Hyperparameters and seed is set. We also always have the scenario with 2 Predators and 1 Prey agent always starting at the same position.

If you want to see the plots you can simply execute the utils.py file.

However there are comments all around the code to further explain what to do

If you want to see the agents act life, you can un-comment the game.render() in line 509 in world.py . However the training and testing works without the rendering

The Source code can also be seen on my Github: https://github.com/JoelSchnubel/GML.git If you have any additional question or something is unclear, please don't hesitate to write us an E-mail: schnubel.joel@gmail.com, or sedu00002@stud.uni-saarland.de.

Group Projects Acknowledgements

This is a group project done by Joel Schnubel and Senjuti Dutta. Joel Schnubel is responsible for carrying out the experiments and successfully handling the code. The report is a made by both Joel Schnubel and Senjuti Dutta.

References

Andreas, J. and Klein, D. Reasoning about pragmatics with neural listeners and speakers. In Su, J., Duh, K., and Carreras, X. (eds.), *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, Austin, Texas, November 2016. Association for Computational Linguistics. doi: 10.18653/v1/D16-1125. URL https://aclanthology.org/D16-1125.

Austin, J. *How to Do Things with Words*. Oxford, 1962.

Boutilier, C. Learning conventions in multiagent stochastic domains using likelihood estimates. *arXiv preprint arXiv:1302.3561*, 2013.

Chalkiadakis, G. and Boutilier, C. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pp. 709–716, 2003.

Foerster, J., Assael, I. A., De Freitas, N., and Whiteson, S. Learning to communicate with deep multi-agent reinforcement learning. *Advances in neural information processing systems*, 29, 2016.

Gauthier, J. and Mordatch, I. A paradigm for situated and goal-driven language learning. 10 2016.

Golland, D., Liang, P., and Klein, D. A game-theoretic approach to generating spatial descriptions. pp. 410–419, 10 2010.

Igor Mordatch, P. A. Emergence of grounded compositional language in multi-agent populations. *arXiv*, 2017. URL https://arxiv.org/abs/1703.04908.

Lake, B., Ullman, T., Tenenbaum, J., and Gershman, S. Building machines that learn and think like people. *Center for Brains, Minds Machines (CBMM) Memo No. 046*, arXiv, 04 2016. doi: 10.1017/S0140525X16001837.

Mordatch, I. and Abbeel, P. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Ryan Lowe, Yi Wu, A. T. J. H. P. A. I. M. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv*, 2017. URL https://arxiv.org/abs/1706.02275.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Sukhbaatar, S., Fergus, R., et al. Learning multiagent communication with backpropagation. *Advances in neural information processing systems*, 29, 2016.

Sukhbaatar, S., Lin, Z., Kostrikov, I., Synnaeve, G., Szlam, A., and Fergus, R. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Wang, S. I., Liang, P., and Manning, C. D. Learning language games through interaction. In Erk, K. and Smith, N. A. (eds.), *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2368–2378, Berlin, Germany, August 2016. Association for Computational Linguistics. doi: 10.18653/v1/P16-1224. URL https://aclanthology.org/P16-1224.

Winograd, T. A procedural model of language understanding. 1973.

A. Appendix

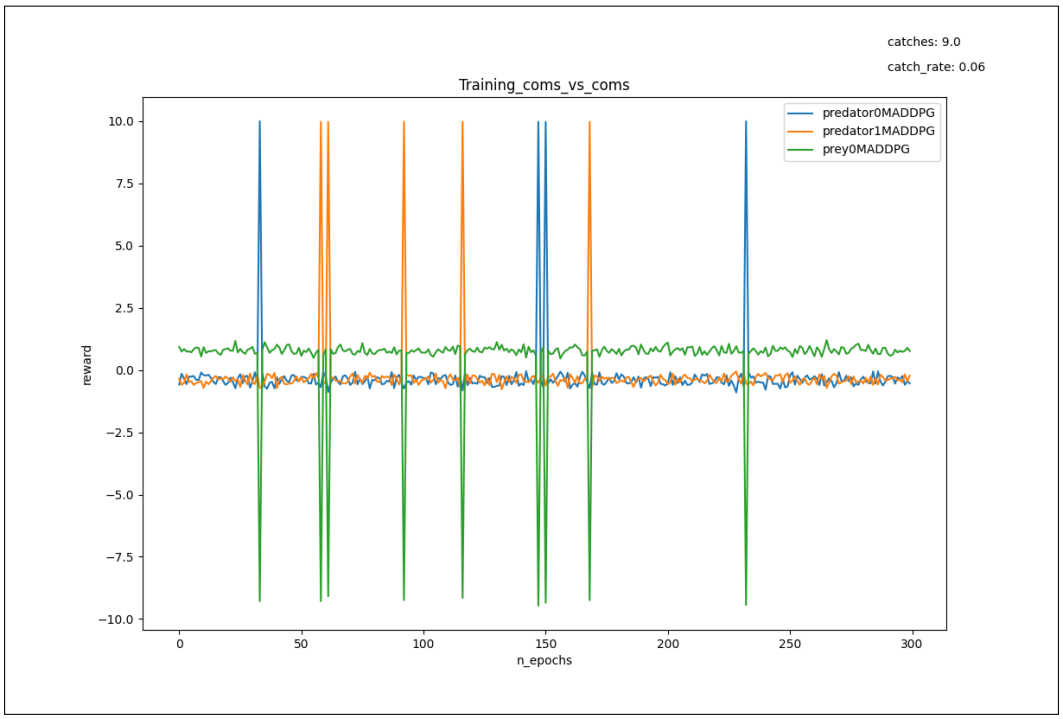

Figure 3. plot of training predator and prey both with communication streams

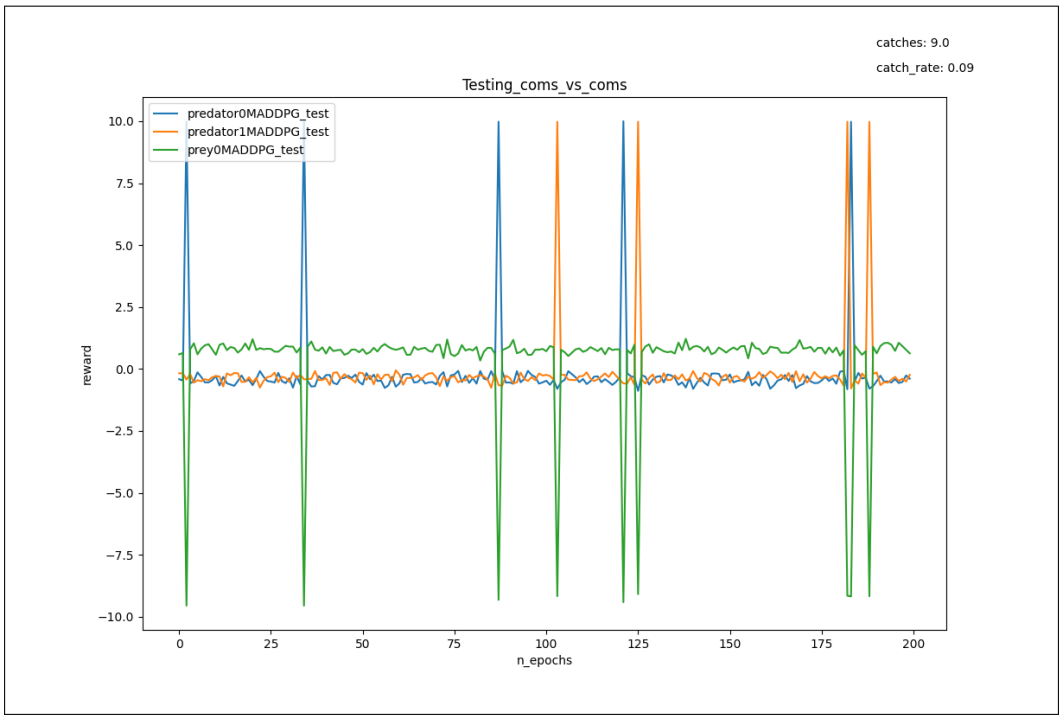

Figure 4. plot of testing predator and prey both with communication streams

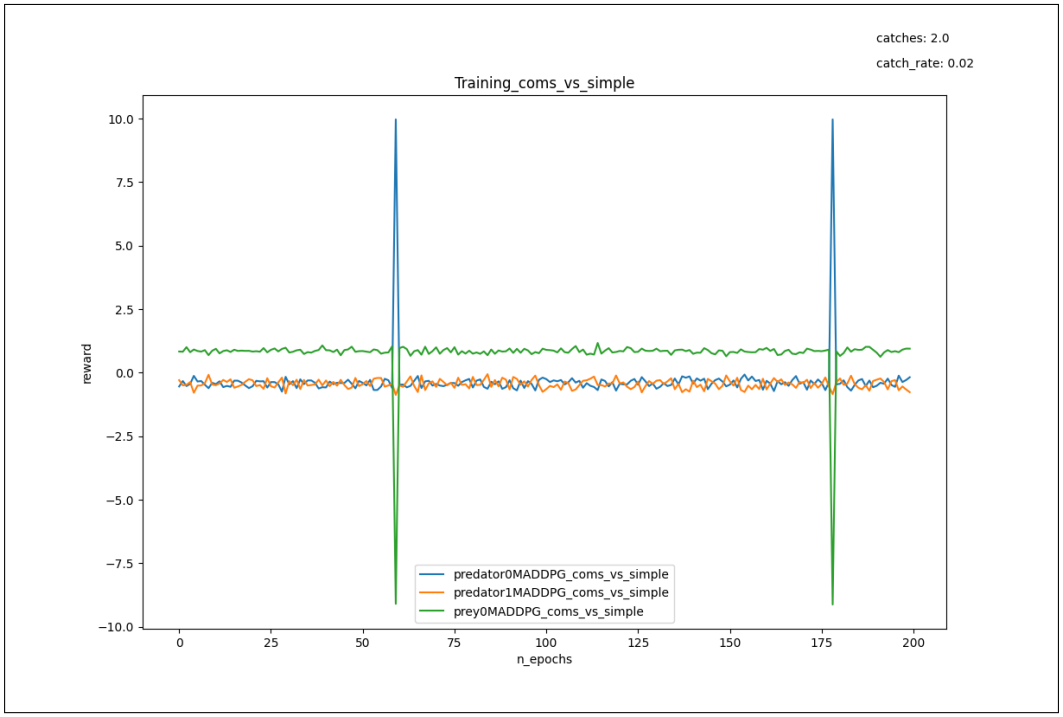

Figure 5. plot of training predator with communication streams and prey without

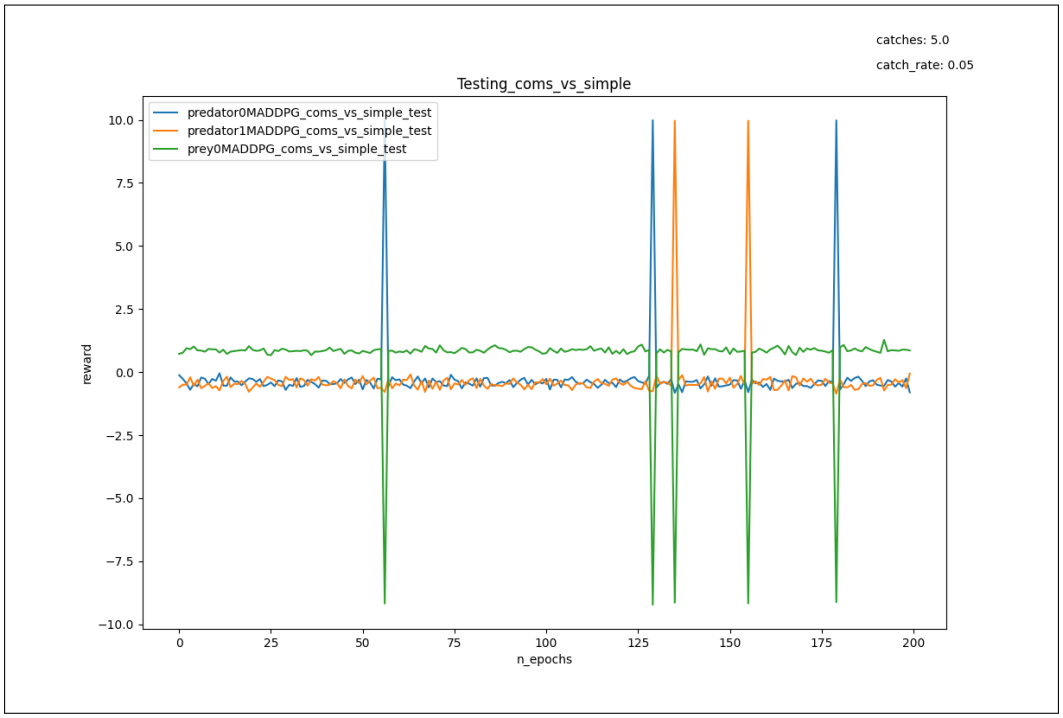

Figure 6. plot of testing predator with communication streams and prey without

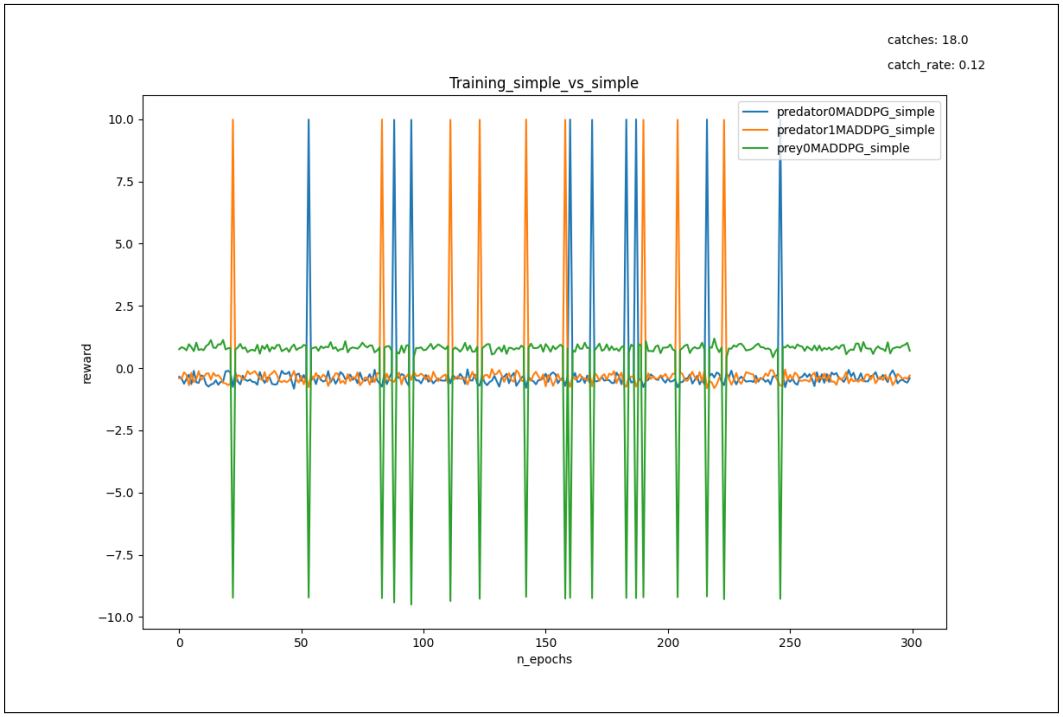

Figure 7. plot of training predator and prey both without communication streams

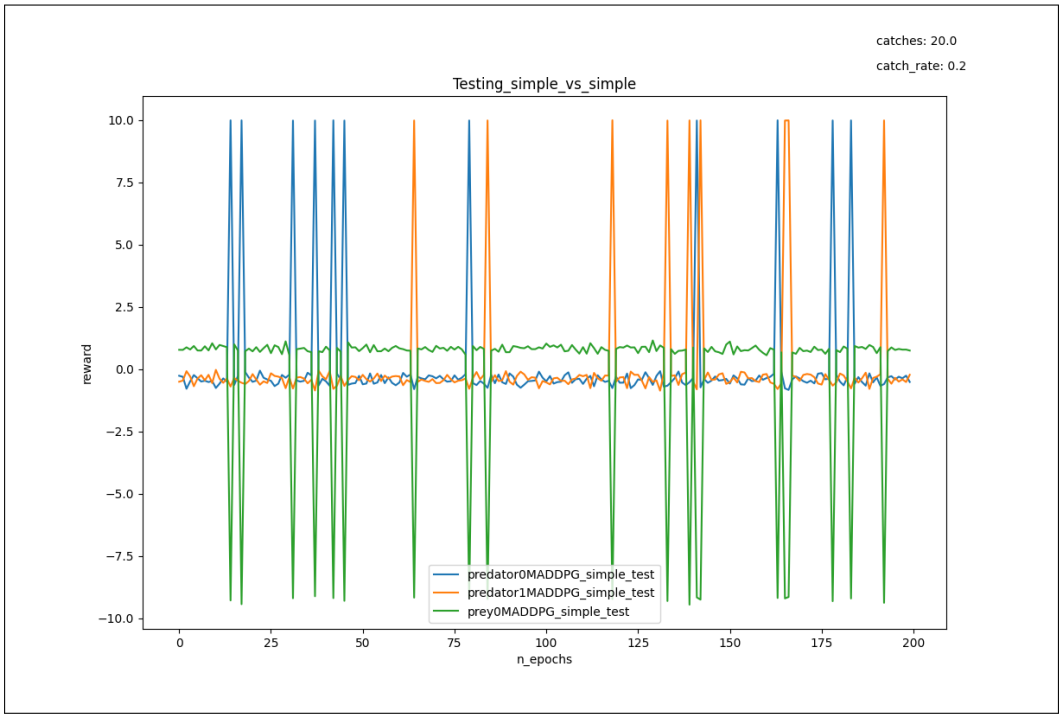

Figure 8. plot of testing predator and prey both without communication streams