Deep Reinforcement Learning in Continuous Multi Agent Environments

Ang Li **Michael Kuchnik** **Yixin Luo** **Rohan Sawhney**

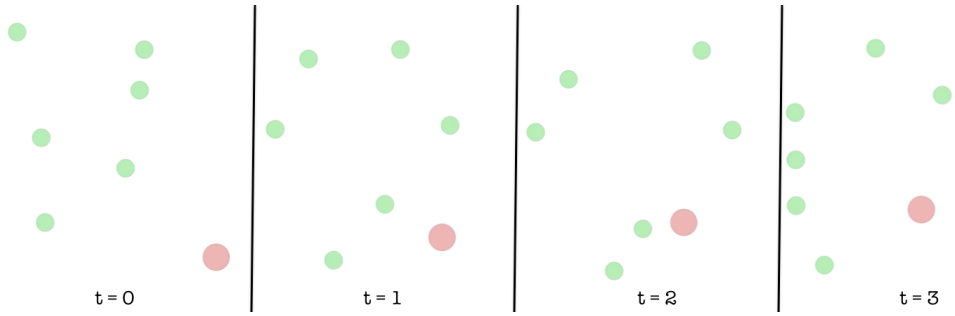

Figure 1: Illustrations from an episode of Predator Prey with 1 predator (red) and 6 preys (green)

1 Problem Statement

Many of the recent successes of deep reinforcement learning have been in single agent domains with discrete low dimensional action spaces. However, several important and interesting problems in communication, robotics, control and gaming have continuous high dimensional action spaces that involve interactions between multiple agents. Naively discretizing action spaces has many limitations, most notably the curse of dimensionality. On the other hand, single agent techniques struggle in non stationary multiagent environments as they do not take the actions of other agents into account while modeling an agent's actions and estimate of futures return. In this project report, we apply Deep Q Network (DQN) [4], Deep Deterministic Policy Gradient (DDPG) [2] and Multi Agent Deep Deterministic Policy Gradient (MADDPG) [3] to continuous multi-agent domains with both competitive and cooperative scenarios. In particular, we test the effectiveness of these methods in the classic predator-prey game [Figure 1], where slower agents chase faster adversaries.[1]

2 Background and Related Work

2.1 Continuous and High Dimensional Action Spaces

Many tasks, and in particular those related to physical control, have continuous (real valued) and high dimensional action spaces. A straightforward approach to extending reinforcement learning methods designed for discrete action spaces (e.g. Q learning) to continuous domains is to simply discretize the action space. However, since tasks requiring fine control of actions require a correspondingly finer grained discretization, this leads to an explosion in the number of discrete actions. Large action spaces are difficult to explore efficiently and can make training intractable with traditional methods. In this project, we evaluate the performace of algorithms designed to work with discrete (DQN) as well as continuous actions spaces (DDPG, MADDPG).

[1] Videos and code for this project can be found in our repository: `https://github.com/camellyx/10707-deep-learning-project`.

10707 Class Project (2017).

2.2 Multi-Agent Setting

Traditional single agent reinforcement learning algorithms typically formulate their environment as a Markov Decision Process (MDP). The multi-agent extension of MDPs with N agents is defined by a set of states S that describe the possible configurations of all agents, a set of actions $A_1, ..., A_N$ and a set of observations $O_1, ..., O_N$. To choose actions, each agent i uses a stochatic policy $\pi_{\theta_i} : O_i \times A_i \rightarrow [0, 1]$ which produces the next state according to the transition function $T : S \times A_1 \times ... \times A_N$. Each agent receives a reward $r_i : S \times A_i \rightarrow \mathbb{R}$ which is a function of the state and the agent's action. Each agent also receives an (potentially partial) observation of the enviorment. The aim of each agent i is to maximize its own total expected return $R_i = \sum_{t=0}^{T} \gamma^t r_i^t$, where γ is the discount factor and T is the reward.

2.3 Q-Learning and Deep Q Network (DQN)

Q-Learning is a popular method for reinforcement learning that makes use of an action value function $Q^\pi(s, a) = \mathbb{E}[R|s^t = s, a^t = a]$. Many algorithms use the the Bellman equation $Q^\pi(s, a) = \mathbb{E}_{s'}[r(s, a) + \gamma \mathbb{E}_{a' \sim \pi}[Q^\pi(s', a')]]$ to recursively rewrite and estimate this Q function. A DQN successfully learns the action value function Q^* corresponding to the optimal policy by minimizing the loss:

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim D}[(r + \gamma \max_{a'} Q^*(s', a') - \bar{Q}^*(s, a|\theta))^2] \tag{1}$$

where $\bar{Q}$ is the target Q function. Compared to previous Q-Learning techniques, DQN uses deep function approximators in a stable and robust way due to two innovations - 1) the network is trained with a target Q network to give consistent targets during temporal difference backups 2) the network is trained off-policy with samples from a replay buffer D to minimize correlations between samples.

In the multi-agent setting, Q-Learning can be applied by having each agent i learn an independently optimal function Q_i. However, there are two fundamental problems in using a DQN in a multi-agent enviroment. First, from any agent's point of view, since the environment is affected by actions taken by other agents unknown to that agent, the state transition function becomes non stationary, violating MDP criteria. Second, and for a similar reason, the experience replay memory becomes inaccurate in approximating the state transitions probabilities.

2.4 Policy Gradient and Actor Critic Methods

Another popular choice for single agent reinforcement learning algorithms are the Policy Gradient methods that directly adjust the parameters θ of the policy in order to maximize the objective $J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta}[R]$ by taking steps in the direction of $\nabla_\theta J(\theta)$. There are several advantages of Policy Gradient over value-based techniques, most notably that they are effective in high-dimensional or continuous action spaces, and can learn stochastic policies. The policy gradient theorem gives the following expression

$$\nabla_\theta J(\theta) = \mathbb{E}_{s \sim p^\pi, a \sim \pi_\theta}[\nabla_\theta log(\pi_\theta(a|s)) Q^\pi(s, a)] \tag{2}$$

for $\nabla_\theta J(\theta)$, where p^π is the state distribution. Most Policy Gradient methods differ in how they estimate Q^π. For example, REINFORCE is a popular Monte Carlo algorithm that simulates an entire episode to obtain an estimate of the return. Like other single-agent techniques, Policy Gradient methods can also be naively applied to multi-agent reinforcement learning problems. However, since the reward of each agent is highly dependent on other agents' actions the variance of the gradient updates can be large, making it difficult to train and converge to an optimal policy.

Actor-Critic methods mitigate the high variance problem of Policy Gradient techniques by combining it with value function based techniques. More specifically, Actor-Critic methods learn an approximation of the true action-value function $Q^\pi(s, a)$ by e.g., temporal difference learning or deep function approximators. $Q^\pi(s, a)$ is called the critic while the actor estimates the policy.

2.4.1 Deep Deterministic Policy Gradient (DDPG)

DDPG is an actor critic method that adapts the underyling success of Deep Q-learning to the continuous action domain. DDPG is based on the deterministic policy gradient algorithm (DPG) [5]

which rewrites the gradient of the objective $J(\theta)$(Section 2.2) as:

$$\nabla_\theta J(\theta) = \mathbb{E}_{s\sim D}[\nabla_\theta \mu_\theta(a|s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}] \tag{3}$$

where $\mu_\theta : S \rightarrow A$ are deterministic policies. DDPG approximates both the policy μ and the critic Q^μ with deep neural networks, sampling trajectories from a replay buffer of experiences and using a target network as in DQN. Note that in the multi-agent setting, each agent is trained individually with DDPG, there is no combined optimization of the agents. However, the MADDPG algorithm, explained in the next section, naturally extends DDPG to the multi-agent setting during training, potentially resulting in much richer behavior between agents.

2.4.2 Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

MADDPG is a recently developed general-purpose multi-agent learning algorithm that is a simple extension of actor-critic policy gradient methods (specifically DDPG) where the critic is augmented with extra information about the policies of other agents while the actor only has access to local information (i.e., its own observations). In this framework of centralized training with decentralized execution, agents don't need to access the central critic at test time; they learn approximate models of other agents and effectively use them in their own policy learning procedure. Furthermore, since the centralized critic is learned independently for each agent, this approach can be used to model arbitrary reward structures between agents, including adversarial cases where the rewards are opposing.

More concretely, consider a game with N agents with policies parameterized by $\theta = \theta_1, ..., \theta_N$ and let $\mu = \mu_1, ..., \mu_N$ be the set of all agent policies. Then, the gradient of the expected return for agent i with deterministic policies μ_{θ_i} are given by:

$$\nabla_{\theta_i} J(\mu_i) = E_{x,a\sim D}[\nabla_{\theta_i} \mu_i(a_i|o_i) \nabla_{a_i} Q_i^\mu(x, a_1, ..., a_N)|_{a_i=\mu_i(o_i)}] \tag{4}$$

Here the experience replay buffer D contains the tuples $(x, x_0, a_1, ..., a_N, r_1, ..., r_N)$, recording experiences of all agents. Q_i^μ is the centralized action-value function that takes as input the actions of all agents, in addition to some state information x, and outputs the Q-value for agent i. Since each Q_i^μ is learned separately, agents can have arbitrary reward structures, including conflicting rewards in a competitive setting.

A primary motivation behind MADDPG is that, if the actions taken by all the agents are known, then the environment is stationary as the policies change. This is not the case if the actions of other agents are not explicity conditioned on, as done for most traditional reinforcement learning. Finally, note that the policies of other agents needs to be known to compute the loss. Knowing the observations and policies of other agents is not a particularly restrictive assumption; if the goal is to train agents to exhibit complex communicative behaviour in simulation, this information is often available to all agents. However, MADDPG also allows for this assumption to be relaxed by learning the policies of other agents from observations.

3 Multi-Agent Environment

We use the multi-agent extension of the OpenAI Gym framework [1] to setup our predator prey environment. We employ this framework as it is quickly becoming the standard in terms of environments to benchmark reinforcement learning algorithms in. In particular, the framework defines the state and action space for every agent based on the environment, which allows researchers to focus on making agents act intelligently. The framework is turn based, i.e., each agent performs an action (including no action) at every time step. The environment in turn provides each agent with a reward as well as an observation that describes the environment's state.

3.1 The Predator-Prey Environment

In the predator prey environment, slower (red) agents chase faster (green) adversaries. Multiple agents can exist on either team and the goal of each agent is to maximize its own reward in this mixed cooperative and competitive setting. The environment returns a list of states, one per agent, once each agent performs an action. An agent's state consists of the (1) agent's position and velocity, (2) the relative position of any landmarks, (3) the relative position of all other agents. The environment

also returns a reward per agent. Agents receives -50 reward if they leaves the arena. If they do, the environment is reset, i.e., the next episode begins in a random configuration. Red agents receive a +100 reward for any green agent they intercept. Green agents, on the other hand, receive a -100 reward if they are intercepted by a red agent. To help training converge faster, we also enabled an l2 penalty. Red agents were given a negative reward proportional to their distance from green agents to prevent them from drifting aimlessly during the early stages of training. Green agents were given a positive reward proportional to their distance from red agents to similarly help them train faster.

The action space consists of a list of actions, one per agent. Each agent's action is described by 5 numbers, which represent whether an agent should stay put or move up/down/left/right (this representation seems to contain redundancies, but it is specified by the environment and should not be modified by the algorithm designer). Each number is clamped between 0 and 1.

4 Network Architecture and Hyperparameters

Each agent in our continuous multi-agent predator prey environment is assigned a deep reinforcement learning model (DQN, DDPG or MADDPG) to determine its actions. The losses these models minimize and the gradients they use are described in Sections 2.3 and 2.4. Each DQN and DDPG model receives as input the current state, next state, action and reward of the agent it is modeling at every time step. On the other hand, MADDPG receives the current states, next states and actions of all agents in the environment, as its critic attempts to model its estimated future returns based on everyone's policies.

The DQN estimates its Q values using a neural network with 2 hiddens layers. Each layer containts 50 hidden units and relu activations. Note that since DQN can only handle discrete and low dimensional action spaces, its action at any time step is one hot encoded, i.e., it can either not move, or move up/down/left/right. Unlike DDPG and MADDPG, its action cannot be a combination of these movements.

Since DDPG and MADDPG are Actor-Critic models, they both contain 2 networks each. In both cases, the Actor network contains 2 hiddens layers with 50 hidden units each and relu activations. The Critic network also contains 2 hidden layers, however, state and action inputs to this network are merged between the first and second hidden layers. The intermediate activation is a relu unit, while the output activation is a sigmoid (since the action values need to clamped between 0 and 1).

To stabilize learning, all networks are trained with a target Q network to give consistent targets during temporal difference backups. The parameters of the target network are updated using the following equation:

$$\theta_i' \leftarrow \tau\theta_i + (1-\tau)\theta_i' \tag{5}$$

where $\tau = 0.001$. Furthermore, during training, minibatches of size 128 are randomly sampled from replay buffers with the latest 10000 state, action, reward and next state tuples. All models are trained with the Adam Optimizer and with a learning rate of 0.001 for 6 hours each. Finally, the DQN is implemented in Keras, while DDPG and MADDPG are implemented in tensorflow.

5 Experiments

For this project, we trained our reinforcement learning models on 3 scenarios in the predator prey setting - 1) 1 green agent vs 1 red agent, 2) 2 green agents vs 1 red agent and 3) 1 green agent vs 2 red agents. We did not use any landmarks, as none of the algorithms we employ explicitly model the landmarks in their optimization, i.e., there is no constraint/penalty that prevents agents from attempting to "move through" the landmarks. Each of the 3 scenarios described above were performed with DQN, DDPG and MADDPG agents separately. The number of steps, collisions, average reward and average loss per episode were collected for all experiments. TODO: DQN vs MADDPG

5.1 1 Green Agent vs. 1 Red Agent

In the 1 green agent vs 1 red agent scenario, we expect our reinforcement learning models to learn that an agent should not go outside the arena. We also expect the number of collisions between both agents to decrease over time as the faster green agent learns to out maneuver the slower red agent.

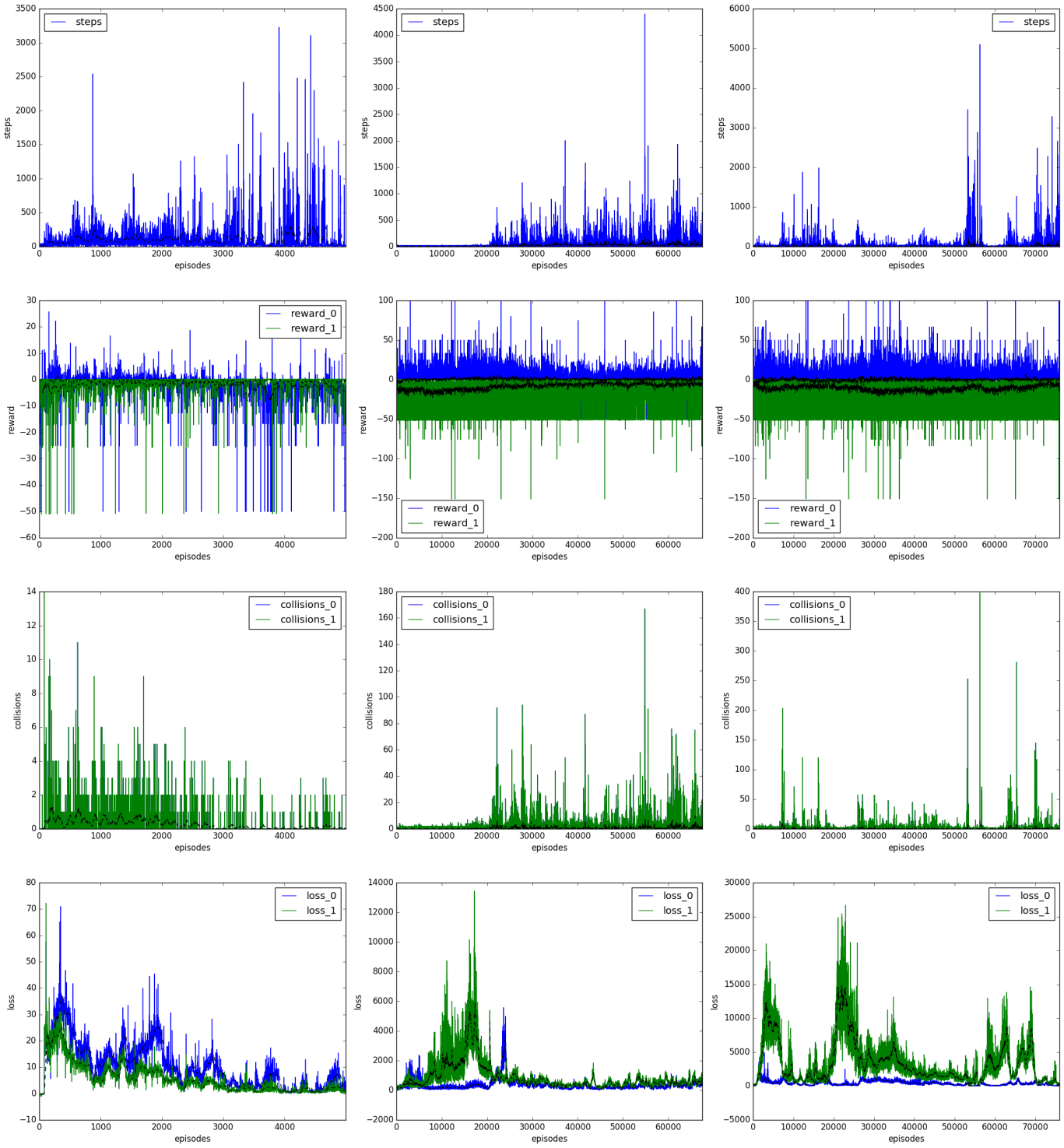

Figure 2: Plots for steps, average reward, collisions and loss per episode for 1 green vs 1 red agent. *Left Column*: DQN, *Middle Column*: DDPG, *Right Column*: MADDPG

In Figure 2, we observe that the average number of steps (i.e., how long each episode lasts) increases over time for all models, confirming that the agents learn not to go outside the arena. If the rewards for collisions and stepping outside the arena are ignored, then this scenario is zero sum. This is confirmed by the rewards vs episodes plots which suggest that the sum of the average rewards for both agents is close to zero for all models. For the DQN agents, we notice that the average number of collisions decreases over episodes. However, the collisions and average loss plots suggest that unlike the DQN agents, the DDPG and MADDPG agents have not yet formulated stable policies.

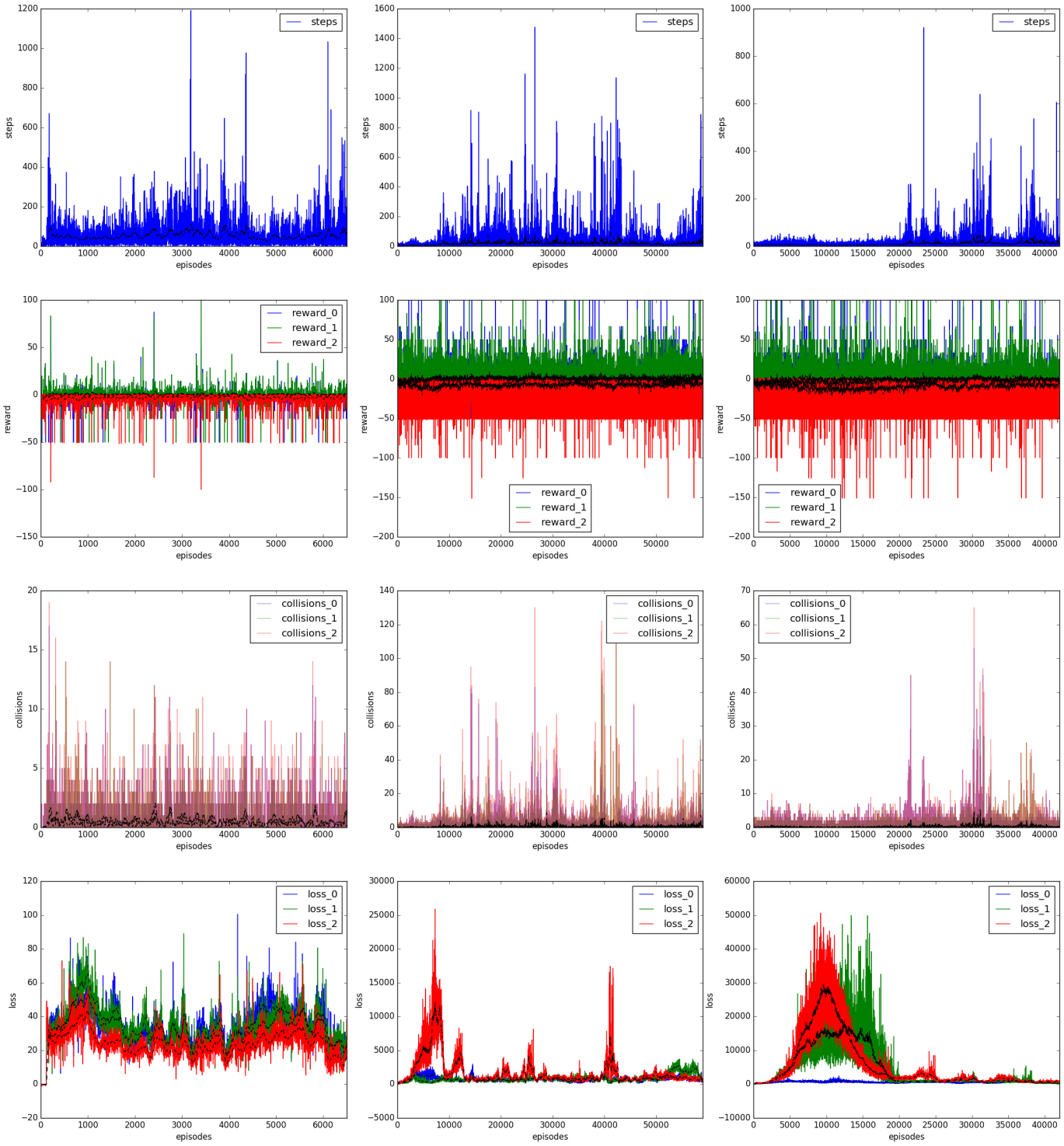

Figure 3: Plots for steps, average reward, collisions and loss per episode for 2 green vs 1 red agent. *Left Column*: DQN, *Middle Column*: DDPG, *Right Column*: MADDPG

5.2 2 Green Agents vs. 1 Red Agent

Unlike the previous scenario, we expect the 2 green agents to now form cooperative strategies against the red agent. This is in fact noticeable in our videos[`https://github.com/camellyx/10707-deep-learning-project`] - green agents stay close together when the red agent is far away from them, and move in opposite directions when the red agent gets close to them. More complex and intelligent behaviours can be noticed in Figure 1, where several green agents either group together or

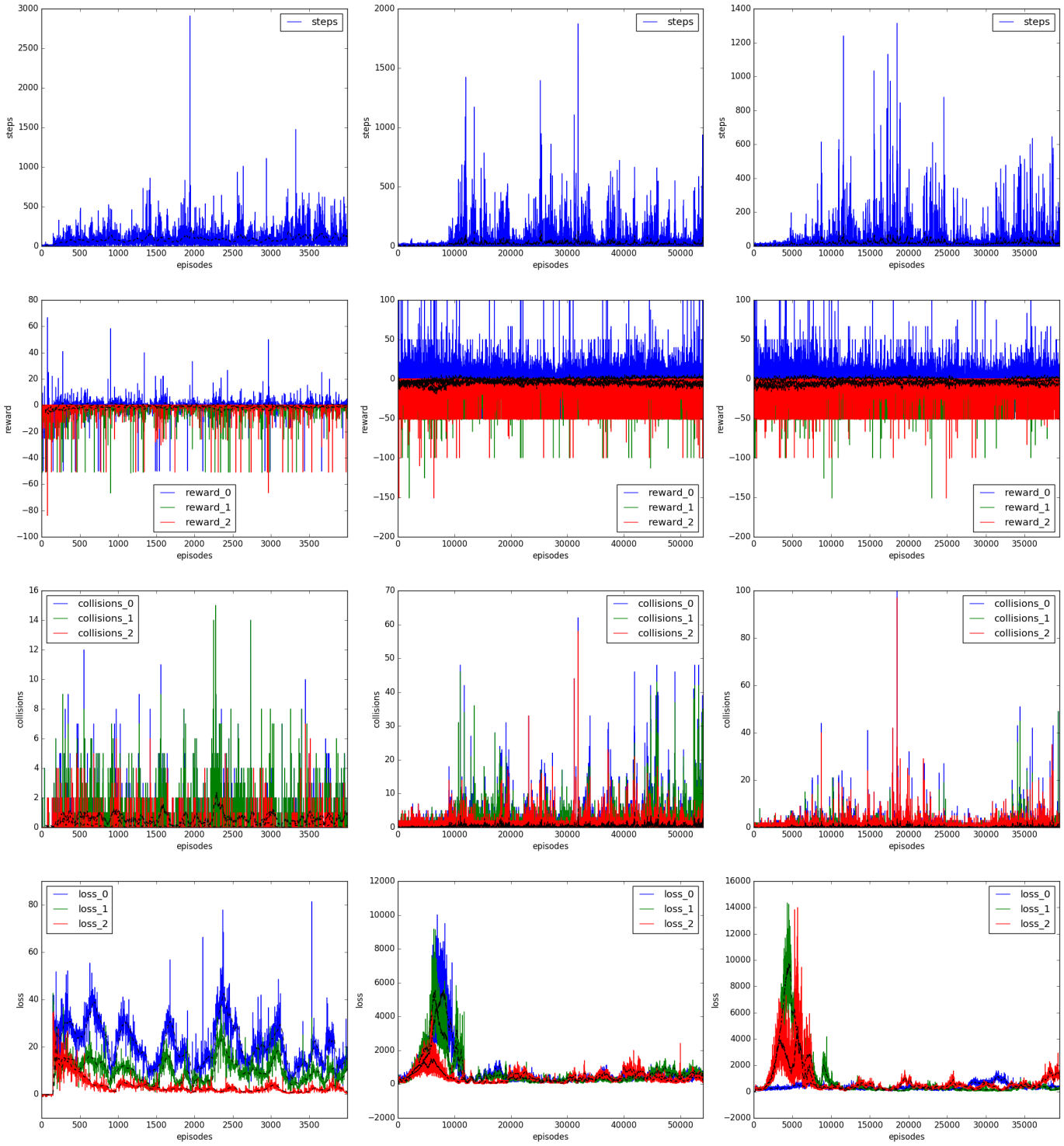

Figure 4: Plots for steps, average reward, collisions and loss per episode for 1 green vs 2 red agents. *Left Column*: DQN, *Middle Column*: DDPG, *Right Column*: MADDPG

enter into a circle configuration based on the red agent's location. The commentary for Figure 3 is similar to that for Figure 2.

5.3 1 Green Agent vs. 2 Red Agents

This time, red agents form cooperative strategies against the green agent. Our videos highlight how red agents corner the green agent by entering into a triangular configuration with it. Once they are close to the green agent, the closer red agent attacks first, while its partner waits directly behind it to

intercept the green agent if it escapes [See Figure 5]. We noticed interesting strategies develop with all 3 deep reinforcement learning models in all scenarios, however, DDPG and MADDPG agents jittered a lot more than DQN agents (possibly due to non stable policies during learning).

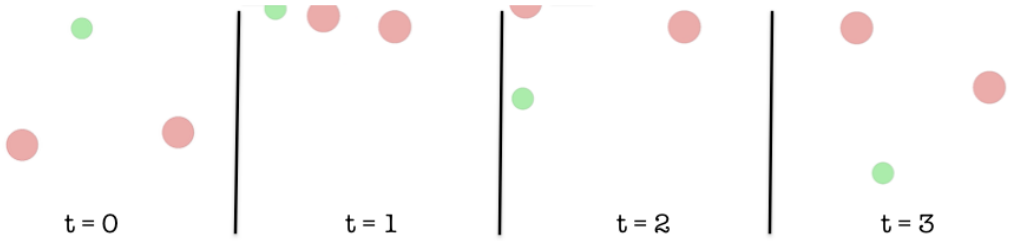

Figure 5: Two red agents forming a cooperative strategy to chase the green agent

6 Conclusion

While the results from our DQN network are quite impressive (which was not expected), DDPG and MADDPG are larger networks (especially MADDPG which takes the states and actions of all agents into account) and most likely require more hyperparameter tuning and longer training times. They should not just be competitive, but better at formulating cooperative and competitive strategies than DQN agents in multi-agent scenarios.

Furthermore, the high resolution of the OpenAI predator prey grid did not disibilitate the DQN networks' discrete and low dimensional action space. DQN agents were able to maneuver themselves out of tight situations (e.g. when a green agent was cornered by red agents) and make sharp (non vertical or horizontal) movements as perceived visually. However, continuous action policies are absolutely integral for physical control. It would be worthwhile applying our DQN, DDPG and MADDPG implementations to environments that highlight the weaknesses of discrete action spaces.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[2] Timothy Lillicrap, Jonathan Hunt, Alexander Pritzel, Nicolas Hees, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2016.

[3] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*, 2017.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, and Martin Wierstra, Daan Wand Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[5] David Silver, Guy Lever, Nicolas Hees, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. *International Conference on Machine Learning*, 2014.