Collaborative Drawing with Multi-Agent Reinforcement Learning (Draft)

Nitin Nataraj, Priyanka Pai

June 27, 2019

1 Abstract

Reinforcement Learning (RL) is a learning paradigm in which agents learn to make optimal decisions by interacting with an environment and maximizing cumulative rewards. Multi-Agent RL (MARL) is an extension of the RL setting to multi-agent domains, in which algorithms need to incorporate observations from multiple agents in cooperative or competitive environments, and derive policies for some or all of the agents. In this paper, we present a collaborative drawing system that uses a Multi-Agent Actor Critic algorithm (MADDPG) [**?**] to fill or trace image outlines. We introduce two MARL tasks, namely Image Tracing and Image Filling and their respective reward functions. To conduct our experiments, we develop the image-contour environment, which is built on top of OpenAI's multi-agent-particle environment [1], (first introduced in [**?**], and then used in [**?**]). We present promising qualitative results on the MNIST image dataset [**?**].

2 Introduction

Reinforcement Learning (RL) is a learning paradigm in which an agent learns to make optimal decisions in an unknown environment by trying to maximize cumulative rewards. Rewards are provided intermittently by a reward function when the agent interacts with the environment. In reality, RL is used in scenarios that require sequential decision making, and have constant environment interactions, such as computer games, robotics and autonomous driving.

The Reinforcement Learning (RL) problem is to control a system so as to maximize a numerical value which represents a long-term objective. The learner is known as the agent and everything external to the agent is known as the environment. The agent interacts with the environment by selecting actions from the an action space, and the environment provides feedback through a reward value. The agent aims to learn optimal policies, which are a series of actions,

[1] https://github.com/openai/multiagent-particle-envs

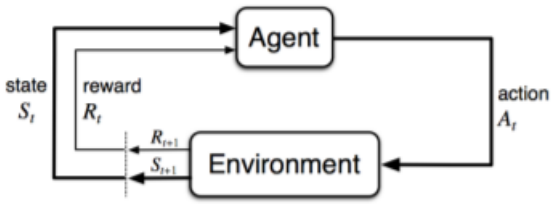

Figure 1: Action-reward feedback loop

that help the agent secure the maximum cumulative reward in the long term. Video games usually provide a consistent reward signal, thus making it easier for RL agents to learn. However, several scenarios, especially in the robotics domain, suffer from the lack of a good reward function. For example, if you want to teach a robot to grasp a water bottle, it is difficult to formulate a valid reward signal. Providing a +1 reward when the robot grasps the bottle, and -1 otherwise, is a poor reward signal as it is very sparse, making it very unlikely for the robot to stumble upon those specific states during training.

RL has existed for several decades [**?**], but has gained renewed traction over the past few years due to the successes of deep learning methods in other domains such as Computer Vision [**?**, **?**, **?**, **?**] and Natural Language Processing [**?**]. Deep Reinforcement Learning (DRL) is an extension of RL in which the learned policy is represented by some deep neural network, which serves as an effective function approximator. Most of the successes of RL have been in single agent domains [**?**, **?**], where modelling or predicting the behaviour of other actors in the environment is largely unnecessary.

2.1 Multi-Agent Systems

Multi-agent systems are rapidly nding applications in a variety of domains, including robotics, distributed control, telecommunications, and economics. The complexity of many tasks arising in these domains makes them difcult to solve with preprogrammed agent behaviors. The agents must instead discover a solution on their own, using learning. A signicant part of the research on multi-agent learning concerns reinforcement learning techniques.

A multi-agent system can be dened as a group of autonomous, interacting entities sharing a common environment, which they perceive with sensors and upon which they act with actuators. Multi-agent systems are nding applications in a wide variety of domains including robotic teams, distributed control, resource management, collaborative decision support systems, data mining, etc. They may arise as the most natural way of looking at the system, or may provide an

alternative perspective on systems that are originally regarded as centralized. For instance, in robotic teams the control authority is naturally distributed among the robots. In resource management, while resources can be managed by a central authority, identifying each resource with an agent may provide a helpful, distributed perspective on the system. Although the agents in a multi-agent system can be programmed with behaviors designed in advance, it is often necessary that they learn new behaviors online, such that

2.2 Multi-Agent RL

A reinforcement learning (RL) agent learns by interacting with its dynamic environment. At each time step, the agent perceives the state of the environment and takes an action, which causes the environment to transit into a new state. A scalar reward signal evaluates the quality of each transition, and the agent has to maximize the cumulative reward along the course of interaction. The RL feedback (the reward) is less informative than in supervised learning, where the agent would be given the correct actions to take (such information is unfortunately not always available). The RL feedback is, however, more informative than in unsupervised learning, where there is no explicit feedback on the performance. Well-understood, provably convergent algorithms are available for solving the single-agent RL task. Together with the simplicity and generality of the setting, this makes RL attractive also for multi-agent learning environment.

2.3 Policy-Gradients

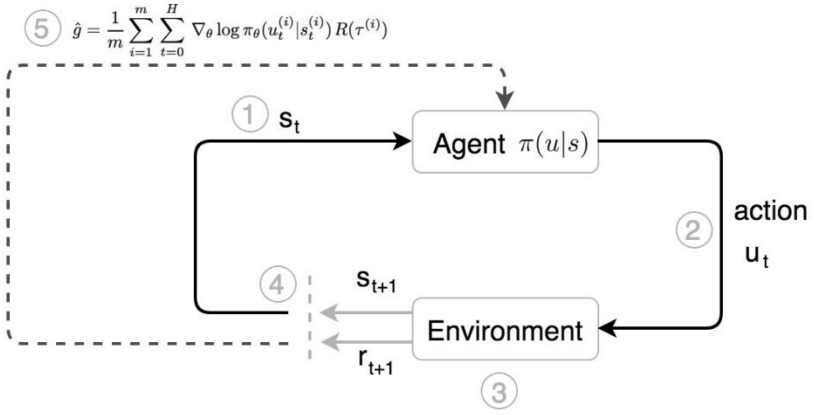

Figure 2: The agent-environment interaction in a Markov decision process

In RL, the instinct is defined as the probability of taking an action u given a state s, where π is the **policy**.

$$\pi = u|s \tag{1}$$

$$J(\theta) = \sum_{s \in S} d_{\pi_\theta}(s) V_{\pi_\theta}(s) \tag{2}$$

where $d_{\pi_\theta}(s)$ is stationary distribution of Markov chain for π_θ. It is natural to expect policy-based methods are more useful in continuous space, because there is an infinite number of actions and/or states to estimate the values for in continuous space and hence value-based approaches are computationally much more expensive.The policy gradient can be represented as an expectation. It means we can use sampling to approximate it. Also, we sample the value of r but not differentiate it. It makes sense because the rewards do not directly depend on how we parameterize the model. But the trajectories are. So what is the partial derivative of the $log\{\pi(\theta)\}$.

$$\nabla_\theta J(\theta) \approx \frac{1}{n} \sum_{i=1}^{N} (\sum_{t=1}^{T} \nabla_\theta log \pi_\theta(a_i, t|s_i, t)) (\sum_{t=1}^{T} r(s_i, t, a_i, t)) \tag{3}$$

And we use this policy gradient to update the policy θ

REINFORCE algorithm uses Monte Carlo rollout to compute the rewards.

1)Sample τ^i from π_θ $(a_t|s_t)$ (run the policy)
2)$\nabla_\theta J(\theta) \approx \sum_i (\sum_t \nabla_{\pi\theta}(a_t^i|s_t^i))(\sum_t r(s_t^i, a_t^i))$
3)$\theta \leftarrow \theta + \alpha \nabla_(\theta)$

2.4 Actor-critic

The two main components in policy gradient are the policy model and the value function.It makes a lot of sense to learn the value function in addition to the policy, since knowing the value function can assist the value update, such as reducing the gradient variance in vanilla policy gradients, and that is exactly what Actor-Critic method does.
Actor Critic methods consist of two models, which may optionally share parameters:

- Critic : Updates the function value parameter w and depending on the algorithm it could be action value $Q_w(a|s)$ or state-value $V_w(s)$
- Actor : updates the policy parameters θ for $\pi_\theta(a|s)$, in the direction suggested by the critic.

Action-value Action-Critic Algorithm

1. Initializes,θ,w at random,sample $a \sim \pi_\theta(a|s)$
2. For t=1...T:
   - Sample reward $r_t \sim R(s,a)$ and next state $s^l \sim P(s^l|s,a)$;
   - Then sample the next action $a^l \sim \pi_\theta(a^l|s^l)$:
   - Update the policy parameters $\theta \leftarrow \theta + \alpha_\theta Q_w(s,a)\nabla_{\pi\theta}(a|s)$
3. Compute the correction(TD error) for the action-value at time t: $\delta_t = r_t +_w (s^l, a^l) - Q_w(s,a)$ and update $a \leftarrow a^l, s \leftarrow s^l$

Our contributions in this project are twofold. Firstly, we present a new environment that generates outlines of images randomly selected from the MNIST dataset [**?**]. Second, we define two tasks with different reward functions in which several agents must cooperatively either align themselves on the image contour (image tracing); or move inside the image contour (image filling).

3 Algorithms

3.1 Deep Deterministic Policy Gradient

In DDPG algorithm, agent learns policy(function) to do a task in environment. Here the function is neural network (NN) which is trained through back-propagation. Policy learning is guided by Q-value, which itself is learned (NN) by Q-learning. (DDPG has many other component to achieve good training like, replay buffer, target network, etc. but i am not touching them as not required for current purpose) . Agent observers the environment (through sensors), based on these takes an action, and gets a reward from the environment. Using these reward signal it tries to learn policy. OpenAI adapted the above DDPG algorithm for multi agent environment. It uses a 'decentralized actor, centralized critic training approach'. In this approach, all agents have access to all other agents' state observations and actions during the critic training phase, but durinf inference, the agent's action is predicted using only its own state observation. This helps in easing the training as the environment becomes stationary for each agent.

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$

3.2 Multi-Agent Deep Deterministic Policy Gradient

In the multi-agent reinforcement learning (MARL) setting, we need to update the policies of two or more agents simultaneously in competitive or collaborative settings. Extending the single-agent RL paradigm by independently training each agent, results in poor performance, as the agents update their policies without considering the observations and actions of other agents in the environment. This causes the environment to appear non-stationary from the viewpoint of any one agent. While we can have non-stationary Markov processes, the convergence guarantees offered by many RL algorithms such as Q-learning requires stationary environments. For this project we use the Multi-Agent Deep Deterministic Policy Gradient (MADDPG) [**?**] algorithm. The primary motivation behind MADDPG is that if we know the actions taken by all agents, the environment is stationary even as the policies change, since $P(s'|s, a_1, a_2, ..., a_n, \pi_1, \pi_2, ..., \pi_n) = P(s'|s, a_1, a_2, ..., a_n) = P(s'|s, a_1, a_2, ..., a_n, \pi'_1, \pi'_2, ..., \pi'_n)$ for any $\pi_i \neq \pi'_i$. This is not the case if we do not explicitly condition on the actions of other agents, as done by most traditional RL algorithms.

In MADDPG, each agent's critic is trained using the observations and actions from all the agents, whereas each agent's actor is trained using just its own

Algorithm 1: Multi-Agent Deep Deterministic Policy Gradient for N agents

for episode $= 1$ to M **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state $\mathbf{x}$
  for $t = 1$ to max-episode-length **do**
    for each agent i, select action $a_i = \boldsymbol{\mu}_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t. the current policy and exploration
    Execute actions $a = (a_1, \ldots, a_N)$ and observe reward r and new state $\mathbf{x}'$
    Store $(\mathbf{x}, a, r, \mathbf{x}')$ in replay buffer $\mathcal{D}$
    $\mathbf{x} \leftarrow \mathbf{x}'$
    for agent $i = 1$ to N **do**
      Sample a random minibatch of S samples $(\mathbf{x}^j, a^j, r^j, \mathbf{x}'^j)$ from $\mathcal{D}$
      Set $y^j = r_i^j + \gamma\, Q_i^{\boldsymbol{\mu}'}(\mathbf{x}'^j, a'_1, \ldots, a'_N)\big|_{a'_k = \boldsymbol{\mu}'_k(o_k^j)}$
      Update critic by minimizing the loss $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j \left(y^j - Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_N^j)\right)^2$
      Update actor using the sampled policy gradient:

$$\nabla_{\theta_i} J \approx \frac{1}{S}\sum_j \nabla_{\theta_i} \boldsymbol{\mu}_i(o_i^j) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}^j, a_1^j, \ldots, a_i, \ldots, a_N^j)\big|_{a_i = \boldsymbol{\mu}_i(o_i^j)}$$

    end for
    Update target network parameters for each agent i:

$$\theta'_i \leftarrow \tau\theta_i + (1-\tau)\theta'_i$$

  end for
end for

observations. This allows the agents to be effectively trained without requiring other agents' observations during inference (because the actor is only dependent on its own observations).

4 Environments

To perform our experiments, we adopt the grounded communication environment proposed in [**?**] which consists of M agents and L landmarks inhabiting a two-dimensional world with continuous space and discrete time Figure 3 (left). Agents may take physical actions in the environment and communication actions that gets broadcast to other agents. Unlike [24], we do not assume that all agents have identical action and observation spaces, or act according to the same policy π. The original paper MADDPG paper considers both cooperative and competitive environments, but we choose to focus our efforts on cooperative environments (all agents maximize a shared return), since we are interested in learning agent policies to collaboratively fill or trace an image outline. For our present set of experiments, we do not incorporate any communication between agents, and allow them to only perform physical actions. We build upon the cooperative environment presented in Figure 3 (left), in which the agents need to move towards the landmarks. Our environment is shown in Figure 3 (right), and is described in the next sub-section.

Figure 3: Existing multi-agent cooperative particle environment (left) with M landmarks and M agents and our image-contour environment (right) that contains 1 polygon landmark (contour), and M agents.

4.1 Image-Contour Environment

The image-contour environment is created by obtaining the contour with maximum area using an image randomly selected from the MNIST dataset. We employ the contour discovery algorithm from OpenCV [2]. The image-contour $C(V)$, can be represented by a set of 2D points V, which is a matrix of dimensionality $N \times 2$, where N is the number of points lying on the contour. A polygon is rendered on the environment using the V. At any instant of time, the environment also contains M agents, $\{A_0, A_1, A_2, ..., A_M\}$, represented by 2D circles with centers $\{U_0, U_1, U_2, ..., U_M\}$ and radius p.

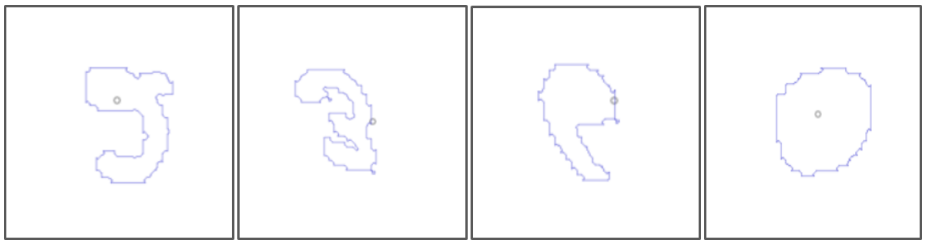

Figure 4: Sample image-contour environments generated using different images from the MNIST dataset.

It can be seen that the OpenCV contour discovery algorithm is not perfect in generating valid contours around the digit Figure 4. However, it is sufficient for our experiments, as we do not impose any design constraints on our image outlines. The image outline merely exists as a landmark for the agents to interact with. The MNIST dataset was chosen because of its widespread acceptance and

[2] https://opencv.org/

significance in the Machine Learning community, and in fact, can be replaced with any other dataset.

5 Tasks

5.1 Image Tracing

In this task, the agents are supposed to align themselves on the image outline. The goal of this task is to replicate or trace the image outline, using a sufficient number of agents and an appropriate agent size, thereby collaboratively "drawing" a figure. We need to carefully formulate the reward function, r, to achieve this objective.

We define a function $D(C, A)$ that takes in the image contour, and an agent as arguments, and determines whether the agent is inside a contour, outside, or lies on an edge (or coincides with a vertex). It returns a positive (inside), negative (outside), or zero (on an edge) value, correspondingly. The return value is the signed distance between the point and the nearest contour edge.

For this task, the reward value needs to decrease as the agent moves further away from the C, either inside or outside, and is maximum when the agent is located exactly on C. A collision penalty is also added to the reward function to discourage collisions between agents. If there are a total of $t <= M - 1$ collisions between the agent in concern and all other agents, then the penalty applied is $-t$. Thus the overall reward function for this task is as follows:

$$r = (\sum_{0}^{M} -|D(C, A_i)|) - t \tag{4}$$

5.2 Image Filling

The image filling task requires the agents to fill the area occupied by the image contour. Therefore, a positive reward is issued when an agent is inside the contour, zero reward on the edges, and negative reward when the agent moves outside the contour. In other words the distance function $D(C, A)$ is a sufficient reward function. After applying the collision penalty t, we get:

$$r = (\sum_{0}^{M} D(C, A_i)) - t \tag{5}$$

The reward functions can be represented graphically as shown in Figure 5.

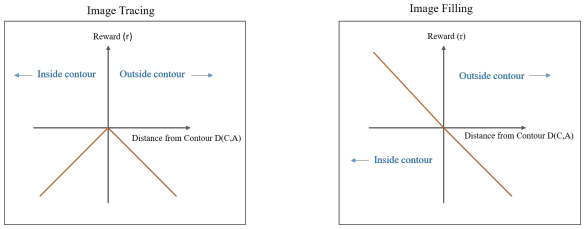

Figure 5: Graphical representations of reward functions for Image Tracing and Image Filling respectively.

6 Results

We present qualitative results on both the Image Tracing and Image Filling tasks using the MADDPG algorithm in Figure 6. RL algorithms are notorious for not converging in short periods of time, and this problem is inevitably more severe in the MARL case. However, we see that in both tasks, the agents learn to move towards the image contour. If we observe the Image Filling results Figure 6 (top), the agents occupy the inside of the contour as expected. Given a larger agent size and population, and more training time, improved performance could be observed.

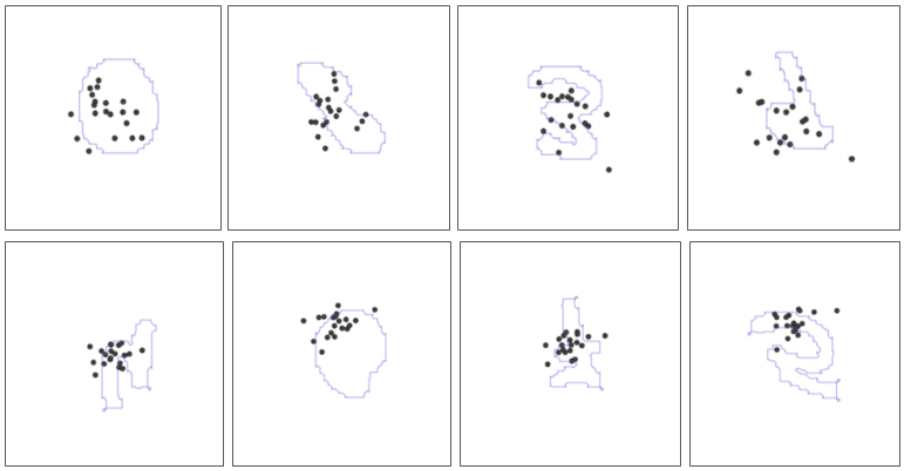

Figure 6: Qualitative results on our Image Tracing (top) and Image Filling (bottom) collaborative drawing tasks.

In the Image Tracing results in Figure 6 (bottom), we notice that the agents move toward the edges of the contour but cluster together, which results in poor tracing results. We believe that improved image tracing results can be

obtained by modifying the reward function in **??** to include a term that penalizes agents when they come too close to each other. Naturally, increasing the agent population increases the computational complexity of the system, and thus we limit the number of agents, $|A|$ to 20 in our experiments. We set the world size to (224, 244) and the agent radius, p to 0.02. The MADDPG algorithm was trained for approximately 10000 episodes on each task, consuming around 8 hours each on a Windows 10 64-bit PC with Core i7-7700HQ @ 2.80 GHz CPU, 16 GB RAM and a 1060 Ti Nvidia GPU, with 6 GB memory.

7 Acknowledgement

We would like to thank Alina for organizing this challenge and for supporting us throughout its entirety.

References

Peek into Reinforcement Learning

Citations

[1] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In Advances in neural information processing systems, pages 271–271. Morgan Kaufmann Publishers, 1993.

[2] Mean Field Multi-Agent Reinforcement Learning by Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. arXiv, 2017

[3] Multiagent Bidirectionally-Coordinated Nets for Learning to Play StarCraft Combat Games by Peng P, Yuan Q, Wen Y, et al. arXiv, 2017.

[4] Hierarchical multi-agent reinforcement learning by Makar, Rajbala, Sridhar Mahadevan, and Mohammad Ghavamzadeh. The fifth international conference on Autonomous agents, 2001.

[5] Reinforcement learning to play an optimal Nash equilibrium in team Markov games by Wang X, Sandholm T. NIPS, 2002.

[6] A reinforcement learning scheme for a partially-observable multi-agent game by Ishii S, Fujita H, Mitsutake M, et al. Machine Learning, 2005.

[7] Multi-agent Deep Reinforcement Learning with Extremely Noisy Observations by Ozsel Kilinc,Giovanni Montana,2018

[8] Gradient-Based Learning Applied to Document Recognition by Yann LeCun,Leon Bottou,Yoshua Bengio and Patrick Haffner,1998

[9] Multi-Agent Actor-Critic fro Mixed Cooperative-Competitive Environments by Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, Igor Mordatch, 2018

[10] Reinforcement Learning: An Introduction by Richard S. Sutton , Andrew G.Barto, 2014

[11] Transfer Learning in Natural Language Processing Tutorial Sebastian Ruder, Matthew Peters, Swabha Swayamdipta, Thomas Wolf, 2017

[12] Playing Atari with Deep Reinforcement Learning , Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Grave, Ioannis Antonoglou, Daan Wierstra, Martin Riedmiller

[13] Very Deep Convolutional Networks for Large-Scale Image Recognition Karen Simonyan Andrew Zisserman, 2015

[14] ImageNet Classification with Deep Convolutional Neural Networks Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton