Attention-based Policies for Collaborative Multi-Agent A2C

1st Hugo Garrido-Lestache
Department of Computer Science and Software Engineering
Milwaukee School of Engineering
Milwaukee, WI USA
garrido-lestacheh@msoe.edu

2nd Jeremy Kedziora
Department of Computer Science and Software Engineering
Milwaukee School of Engineering
Milwaukee, WI USA
kedziora@msoe.edu

***Abstract*—Collaboration is essential in solving complex real-world problems, as it enables tasks to be completed more efficiently and leverages the diverse perspectives and skillsets of team members. Multi-Agent Reinforcement Learning (MARL) can be utilized to train agents in environments where they must collaborate to achieve a common goal. MARL is challenging due to non-stationarity introduced into the environment by multiple agents learning simultaneously and its development has mainly focused on managing these challenges without any special focus on collaboration. Recent work on MARL has introduced the use of an attention mechanism in the critic of the Actor-Critic architecture that allows the critic to select relevant information from the other agents' observations, promoting better learning of the value function. In this paper, we study the impact of the attention mechanism on the quality of learned collaborative policies. We propose a new actor-critic architecture that incorporates an attention mechanism in the actor to model collaboration directly in the learned policies. We evaluate this architecture on a simulation of a soccer game that features competition between teams of collaborators and compare it to current state-of-the-art in MARL as well as classical reinforcement learning algorithms.**

***Index Terms*—Reinforcement Learning, Multi-Agent Systems, Actor-Critic, Attention Mechanism**

I. Introduction

Collaboration and communication are likely one of the most important aspects that make the human species the most advanced on earth. Because of this, exploring this skill in AI could drastically increase the magnitude and complexity of problems we can tackle through the ability to scale the number of agents we can assign to a problem while ensuring cohesion within the agents. In this paper, we present a new model algorithm with a unique focus of collaboration between agents in multi-agent reinforcement learning.

Attention was first brought to the spotlight by [**?**]. Since then, attention has been an amazing technique which has been used in a wide range of machine learning domains. Specifically in multi-agent reinforcement learning, attention was used by [**?**] to improve the advantage calculation. In this paper, we look at using

The attention mechanism on the model aim to allow agents to communicate and share information and thoughts in a selective manner. Intuitively, this works by allowing an agent to ask an open question to other agents, all agents then choose to which agents are fit to respond and the ones who are fit respond and transfer this information. In the example of a soccer game, a player with the ball may ask "who should I pass the ball to", agents which do not have a defender may be selected as being fit to respond and then the information of "pass to this agent" is transmitted.

II. Related Work

Many existing architectures exist for managing collaborative multi-agent systems.For example, In [**?**] they used the option-critic architecture first described in [**?**] and extended to cooperation between agents. In [**?**] they use a hierarchical approach where a top-level policy assigns roles to lower-level policies in the form of a latent vector. When it comes to using attention for the purpose of collaboration, [**?**] uses attention to determine whether agents should communicate with each other and then actually communicate through the use of Long-Short-Term-Memory architecture. The most similar paper to our work is [**?**] where they explored the use of attention mechanism in the critic. They additionally explored methods for extracting a new baseline (named multi-agent baseline) which consisted across averaging across the actions of all cooperative agents to generate a better estimate of the return.

III. Our approach

We begin by introducing the fundamental principles and notation underlying reinforcement learning. Following this, we provide a comprehensive description of our contributions.

A. Background and preliminaries

In reinforcement learning, control problems with multiple agents can be modeled as Markov Decision Games (MDG), which are defined by a set of states S that describe the current common environmental conditions facing a set of n agents, a set of actions A that all agents can take, where the probabilities $p(s \mid \vec{a}, s')$ for transitioning from state s to state s' given all of the agent's actions $a_1, a_2..., a_n$ denoted as $\vec{a}$, and a function $\vec{r} : S \times \vec{A} \times S \rightarrow \mathbb{R}^n$ so that $\vec{r}(s', \vec{a}, s)$ supplies the immediate rewards for each agent associated with this transition. In environments that take place over a finite number of discrete periods T, the sequence of periods in which the agents participate is referred to as an episode. At each time

step, each agent makes an observation $o_{t,i}$ of the global state such that $o_{t,i} = O_i(s_t)$, encapsulating all available information for agent i. The goal of agents is to learn a shared policy $\pi(a_i|o_i)$, which describes the probability that a trained agent should take action a_i given an observation o_i, to maximize its individual sequence of rewards throughout an episode: $\sum_{t=0}^{T} \gamma^t r_i(s_{t+1}, \vec{a}_t, s_t)$. Here $\vec{a}_t$ and s_t are the agents' actions and the common state at time t and $\gamma \in [0, 1]$ is the discount factor on future rewards.

B. Policy Gradients

Policy gradient methods differ from value-based methods through the direct modeling of the policy rather than estimating the value of actions given a state. To learn the policy parameters θ for a shared policy $\pi_\theta(a_i|o_i)$ for all agents, we use the policy gradient theorem to update the parameters in the direction of increase in the sum of discounted rewards. The policy gradient theorem is given by:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{i=1}^{n} \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta \big(a_{i,t} \mid o_{i,t}\big)\, G_{t,i} \right] \quad (1)$$

Where $G_{t,i}$ is the sum of discounted rewards (the return) from a given timestep t onwards for a given agent i and is defined as:

$$G_{t,i} = \sum_{k=t}^{T-1} \gamma^{\,k-t}\, r_i\big(s_{k+1},\, a_k,\, s_k\big). \quad (2)$$

Following the policy gradient theorem, we can take steps in the direction of $J(\theta)$, through the use of gradient ascent on sampled episodes, to reach an optimal shared policy.

C. Actor-Critic Architectures

Due to high variance in estimates of the return $G_{t,i}$, a baseline is often subtracted from the return. A common choice for the baseline is the state-action-value function $Q_\theta(o_{t,i}, a)$ which predicts the return for a specific agent given a chosen action, under the current shared policy π_θ. Replacing $G_{t,i}$ with $G_{t,i} - Q_\theta(o_{i,t}, a)$ in the policy gradient theorem we decrease the variance in the gradient $\nabla_\theta J(\theta)$. Actor-Critic architectures model the actor as the policy $\pi_{\theta_i}(a|o)$ and the critic as a parametric value function $V_{\theta,i}(o_t)$. The actor policy's parameters are learnt through the policy gradient theorem and the critic's parameters are learnt through value-based methods. The critic's parameters can also be updated via gradient descent on the difference between the critic's predicted state value and a target value, which can be either the observed $G_{t,i}$ or a bootstrapped value.

D. Attention Mechanism

The attention mechanism enables an agent to query other agents for the internal representations (or thoughts). Thereby enhancing the information within the policy allowing for more collaborative actions. In the critic, this aggregates information from multiple agent to more accurately evaluate an agents state-action value, as seen in [**?**]. To incorporate attention into the shared policy, we must condition the agents decision both on both it's own observation and the other agents' observations, this is denoted as $\pi_\theta(a_i|(o_i, o_{\setminus i}))$ where $o_{\setminus i}$ represent all observations other than i. Similarly, in the critic this is represented as $Q_\theta((o_i, o_{\setminus i}), (a_i, a_{\setminus i}))$.

E. Actor Attention Architecture

The actor architecture is described in detail below, describing the use of attention and how the agents' observations are transformed to a distribution over actions First, an embedding vector is generated for each agent using the agent's individual observation via a multi-layer perception MLP as seen below:

$$e_i = MLP(o_i) \quad (3)$$

The embeddings of all the agents' are then concatenated to generate the matrix E. This matrix is separately multiplied with a learned Query (W_Q), Key (W_K) and Value (W_V) weight matrix to produce query Q, key K and value V matrices:

$$Q = EW_Q, K = EW_K, V = EW_V \quad (4)$$

As seen in [**?**], Attention is computed as:

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \quad (5)$$

Where d_k is the dimensionality of the K matrix and is done to scale the values appropriately before the softmax function. In our implementation, we apply multiple attention heads in parallel through unique sets of Query, Key, and Value weight matrices [**?**]. The outputs of each attention head are concatenated and passed through a linear transformation to yield the final updated embeddings $\tilde{e}_i$.

Finally we pass these updated embeddings $\tilde{e}_i$ through an MLP, then softmax layer to generate a distribution over actions:

$$\pi_\theta(a_i|(o_i, o_{\setminus i})) = \text{Softmax}\Big(\text{MLP}(\tilde{e}_i)\Big) \quad (6)$$

F. Critic Attention Architecture

Taking inspiration from [**?**], we additionally incorporated attention into the critic. This is done by keeping the core attention mechanism the same, but changing how we generate the initial agent embedding and changing the final functions to generate a scalar which represents the state-action value. To create the initial agent embedding for the critic, we use the following equation which takes into account both the agent's observation and action:

$$e_i = MLP(o_i, a_i) \quad (7)$$

After passing e_i through the critic's set of attention heads, concatenating their output and passing it through a linear transformation we get a new embedding $\tilde{e}_i$. This is passed through a final MLP to generate the scalar which represents the critic's state-action value estimate.

$$Q_\theta((o_i, o_{\setminus i}), (a_i, a_{\setminus i})) = \text{MLP}(\tilde{e}_i) \tag{8}$$

G. Multi-Agent Baseline

The multi-agent baseline is given below:

$$b(o, a_{-i}) = \mathbb{E}_{a_i \sim \pi_\theta} \left[Q_i \left(o, \, (a_i, a_{i\setminus}) \right) \right] \tag{9}$$

The multi-agent baseline computes the expected value of the critic by taking a weighted sum of its outputs over all possible actions for a specific agent, where the weights are given by the agent's policy distribution. This reduces the variance in our baseline as well as generating a more accurate estimate of the true state-value under the current policy due to considering all possible actions and weighting them using the policy's distribution. Furthermore, by holding the other agents' actions fixed, we obtain a more accurate estimate of the agent's true value. This is used instead of the critic's state-action value as baseline when computing the gradient.

IV. Conformity Loss

Conformity loss works on the assumption that good collaborations require agents working on separate and diverse tasks, which may not be true in some environments. In environments where collaboration is best described as agents working on diverse, complementary task, conformity loss aims to encourage agents to work on different tasks. Conformity loss is defined as:

V. Experimentation

blare blare blare

Acknowledgment

This project was done through the use of the "ROSIE" supercomputer at the Milwaukee School of Engineering. This supercomputer has been essential in the development and experimentation of this paper, allowing for fast simultaneous experiments. Over 2000 jobs were run on the ROSIE supercomputer, with a mean of 4 hours for each job totaling over a year of runtime. Furthermore, training with HUGO model on ROSIE was 8x faster than on a conventional modern laptop due to ROSIE fast TPUs. Rosie helped make this project possible.

References