Counterfactual Multi-Agent Policy Gradients

Triantafyllos Afouras∗

May 31, 2017

1 Introduction

Many complex reinforcement learning (RL) problems such as the coordination of autonomous vehicles, network packet delivery, and distributed logistics are naturally modelled as cooperative multi-agent systems. However, RL methods designed for single agents typically perform poorly on such tasks, mainly due to that the joint action space of the agents grows exponentially with the number of agents. To cope with such complexity, it is often necessary to resort to decentralised policies, where each agent selects its own action conditioned only on its local action-observation history and without any knowledge of the other agents’ intentions. Furthermore, partial observability during execution may necessitate the use of decentralised policies even when the joint action space is not prohibitively large.

Another crucial challenge is multi-agent credit assignment. In cooperative settings, the reward signal is usually global and depends on the joint action. Therefore there is no easy way for an agent to infer its own contribution to the team’s success or failure. It is sometimes possible to design individual reward functions that take into account each agent’s individual actions. However these rewards are not generally available, are difficult to craft, especially in cases where the rewards are sparse and delayed, and often fail to encourage individual agents to act altruistically even though their sacrifice might benefit the team as a whole.

In this work, we propose a new multi-agent RL method called counterfactual multi-agent (COMA) policy gradients in order to address the above issues. COMA takes an actor-critic (Konda & Tsitsiklis, 1999) approach, in which every agent has a separate actor that provides a decentralized policy on execution time, conditioned only on the agent’s own action-observation history, but is trained in a centralized way, following a gradient estimated by a centralized critic that conditions on the joint action and all available state information. We furthermore use a counterfactual baseline, an idea inspired by difference rewards (Tumer & Agogino, 2007), where each agent learns from a shaped reward that compares the global reward to the reward received when that agent’s action is replaced with a default action. While difference rewards are a powerful way to perform multi-agent credit assignment, they typically require access to a simulator and, in many applications, it is unclear how to choose the default action. We will discuss how our method addresses this on section 3, where we present it in detail.

We evaluate COMA on StarCraft unit micromanagement1, which has recently emerged as a challenging RL benchmark task with high stochasticity, a large state-action space, and delayed rewards. Previous works (Usunier et al., 2016; Peng et al., 2017) have made use of a centralised control policy that conditions on the entire state and can use powerful macro-actions, provided by StarCraft’s built-in planner, that combine movement and attack actions. To produce a meaningfully decentralised benchmark that proves challenging for scenarios with even relatively few agents, we propose a variant that massively reduces each agent’s field-of-view and removes access to these macro-actions.

∗This report is based on a paper that was submitted to the 30th Neural Information Processing Systems Conference (under review at the time of writing), authored by Jakob Foerester, Gregory Farquhar (the two main authors), Triantafyllos Afouras, Nantas Nardelli and Shimon Whiteson. It is publicly available at https://arxiv.org/abs/1705.08926.

1StarCraft and its expansion StarCraft: Brood War are trademarks of Blizzard Entertainment™.
2 Background

We first introduce notation and basic formulations for the single-agent case and then extend them to our setting where partial observability and multiple agents are concerned.

2.1 Single agent reinforcement learning

A single-agent, fully observable reinforcement learning setting considers a single agent interacting with an environment through a sequence of actions, state transitions and rewards. This is commonly formalized as an episodic Markov Decision Process, defined as a tuple \((S, U, P, R, \gamma)\). At every time step \(t \in 1 \cdots T\) the agent observes the current state of the environment \(s_t \in S\) and chooses a discrete action \(u_t \in U\) according to a policy \(\pi\). The environment then transitions to a new state \(s_{t+1}\) with probability \(P(s_{t+1}|s_t, u_t)\) and the agent receives a reward \(r_t = R(s_t, u_t)\). The objective of the agent is to find a policy that maximizes the expected sum of discounted future rewards \(G_t = \sum_{l=0}^{T-t} \gamma^l r_{t+l}\), known as the return. We refer to states to which the environment transitions to on the last time step \(T\) of the episode as terminal states.

Under a given policy, the value function, \(V^\pi\) of a state \(s_t\), approximates the expected return when starting from that state and following the policy \(\pi\) from then on: \(V^\pi(s_t) = \mathbb{E}_{s_{t+1}, \tau \sim P, u_t, \tau \sim \pi}[G_t|s_t]\). Similarly an action-value function predicts the expected return under the given policy when the agent starts from state \(s_t\), takes action \(u_t\) and follows \(\pi\) thereafter: \(Q^\pi(s_t, u_t) = \mathbb{E}_{s_{t+1}, \tau \sim P, u_{t+1}, \tau \sim \pi}[G_t|s_t, u_t]\). When the action-value function is known, one can easily calculate the value function as follows: \(V^\pi(s) = \sum_u \pi(u|s)Q^\pi(s, u)\). An optimal policy \(\pi^*\) maximizes the value function, \(V^\pi^*(s) = \max_u V^\pi^*(s, u)\), and the optimal action value function \(V^\pi^*(s_t)\) obeys the Bellman optimality equation: \(V^\pi^*(s_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma V^\pi^*(s_{t+1})]\). Similarly in terms of action-value functions: \(Q^\pi^*(s_t, u_t) = \max_u Q^\pi^*(s, u)\) and \(Q^\pi^*(s_t, u_t) = \mathbb{E}_{s_{t+1}}[r_t + \gamma \max_u Q^\pi^*(s_{t+1}, u|s, u_t)]\).

Usually the dynamics of the MDP are not available. Methods referred to as model-based first learn a model of \(P\) and \(R\) and then use it to find the optimal (action-)value function. In contrast, model-free methods operate without a model of the underlying MDP, therefore they need to sample trajectories by interacting with the environment in order to explore the state-action space.

When an action-value function is available, a greedy policy can be directly obtained by simply selecting the best action for every state: \(\pi_{\text{greedy}}(s) = \arg\max_u Q(s, u)\). In the case where a value-function is approximated, a model of the environment is needed to derive the greedy-policy: \(\pi_{\text{greedy}}(s) = \arg\max_u \sum_{s'} P(s'|u, s) V(s')\). This makes Q-value functions more suitable for model-free approaches, however value functions can still be useful for methods such as the actor-critic ones that will be presented below.

2.2 Q-learning, Sarsa, and DQN

Q-learning \cite{Watkins1989} is a popular model-free method that uses an action-value function, which is iteratively updated from experience with regard to the greedy policy:

\[
Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \eta [r_t + \gamma \max_u Q(s_{t+1}, u_t) - Q(s_t, u_t)]
\]

The term within brackets is referred to as the temporal difference (TD) error, while \(r + \gamma \max_u Q(s_{t+1}, u_t)\) as the one-step bootstrap. This update is used for every transition, except for the ones leading to a terminal state, where we consider \(Q(s_{T+1}, u) = 0, \forall u\). Under this update rule and as long as sufficient state-action exploration is achieved, the action-value function is guaranteed to converge to \(Q^\pi^*\). This is usually accomplished by following a stochastic behavior policy such as epsilon-greedy, where an action is picked at random with probability \(\epsilon\), otherwise the agent selects the best one, i.e. the one with the highest Q-value.

Q-learning approximates an action-value function corresponding to a different policy from the one used for exploration. Therefore it belongs to a family of algorithms called off-policy. A similar approach, Sarsa \cite{Rummery1994}, learns the action-value function of the actual policy used while exploring: \(Q(s_t, u_t) \leftarrow Q(s_t, u_t) + \eta [r_t + \gamma Q(s_{t+1}, u_{t+1}) - Q(s_t, u_t)]\) and is labeled as an on-policy method. Sarsa converges to \(Q^*\) provided that all state-action pairs are visited an infinite number of times and the behavior policy converges to the greedy policy. When using e-greedy exploration for example, this can be achieved by annealing \(\epsilon\) linearly to 0.
In settings where the state space $S$ is finite and tractable, $Q$ can be represented with a table, where an entry is maintained for every possible state-action pair. However when $S$ is continuous or $S \times A$ is prohibitively large, a function approximator with parameters $\theta$ needs to be used instead. In the case where non-linear function approximators such as neural networks are used, theoretical guarantees for convergence are poor. [Mnih et al., 2015] use a convolutional neural network (Deep-Q-network, DQN) to approximate the optimal action-value function and apply two practical modifications to stabilise learning. The algorithm alternates between an experience gathering and a training phase. They maintain an experience replay buffer where they store a transition $(s_t, u_t, r_t, s_{t+1})$ for each exploratory step. During training, a mini-batch of transitions is sampled from the replay memory and the network parameters are updated in order to minimize the TD-error:

$$\Delta \theta_t \propto (r + \gamma \max_u Q(s_{t+1}, u; \theta^*) - Q(s_t, u_t; \theta)) \nabla \theta Q(s_t, u_t; \theta)$$  \hspace{1cm} (1)$$

The bootstrap is calculated using a second network with separate weights $\theta^*$, called the target network. The target network’s parameters are updated periodically every $C$ episodes. The replay memory and target network are critical for stabilising Q-learning with deep neural networks, despite the absence of theoretical guarantees for convergence.

When dealing with multiple agents, the maximization over actions of Q-learning is intractable, since the joint action space grows exponentially with the number of agents. Thus we have to resort to on-policy methods such as Sarsa and actor-critic, as will be explained in detail when our method is presented in following sections.

### 2.3 TD($\lambda$)

The Q-learning update rule of equation $[1]$ minimizes a loss of the general form

$$L_t(\theta) = (y - f(s_t, \theta))^2,$$ \hspace{1cm} (2)

with the target $y$ being set the one-step bootstrap and $f = Q$. We can substitute any estimate of the expected return for $y$, as this is what value functions are approximating.

The actual return $G_t$ for the current episode is an unbiased estimate of $E_\pi [G_t]$ but is high-variance due to the noise of the reward signal. The TD-target reduces this variance but is a biased sample, since we do not have access to the actual action-value function of the policy but just $V^\pi(s; \theta)/Q^\pi(s, u; \theta)$ which is a current approximation of it. The two choices mentioned above, TD and monte-carlo, define the limits of a spectrum. We can obtain targets that lie within it by calculating $n$-step bootstraps instead: $G^{(n)}_t = \sum_{l=0}^{n-1} \gamma^l r_{t+l} + \gamma V^\pi(s_{t+n}; \theta)$.

**TD($\lambda$) methods** [Sutton, 1988] use a mixture of $n$-step bootstraps $G^{(n)}_t$ as a target, providing a single parameter $\lambda$ that controls the bias-variance trade-off:

$$y^{(\lambda)} = (1 - \lambda) \sum_{n=1}^{T-t} \lambda^{n-1} G^{(n)}_t + \lambda^{T-t} G_t.$$  

### 2.4 Policy gradient and actor-critic methods

An alternative to using action-value functions for deriving and learning policies are policy gradient methods [Sutton et al., 1999]. Such methods optimise a single agent’s policy, parameterised by $\theta^\pi$, by performing gradient ascent on an estimate of the expected return $E_\pi [G_t]$. Perhaps the simplest form of policy gradient is REINFORCE [Williams, 1992], in which the gradient is:

$$g = E_{s_0:T, u_0:T} \left[ \sum_{t=0}^{T} G_t \nabla \theta \log \pi(u_t|s_t) \right].$$ \hspace{1cm} (3)

As discussed in the outline of the value-based methods above, the monte-carlo sample $G_t$ can be very noisy. In order to reduce the variance, $G_t$ can be replaced by any expression equivalent to $Q(s_t, u_t) - b(s_t)$ [Weaver & Tao, 2001], where the baseline $b(s_t)$ can be any function of the state $s_t$.

A straightforward choice is $b(s_t) = V(s_t)$, in which case $G_t$ is replaced by the advantage $A(s_t, u_t)$. Another option is to replace $G_t$ with the TD error $r_t + \gamma V(s_{t+1}) - V(s_t)$, which is an unbiased
estimate of $A(s_t, a_t)$. Both of the above options require an (action-)value function estimation. In actor-critic approaches (Kimura et al. 2000; Schulman et al. 2015; Wang et al. 2016; Hafner & Riedmiller 2011) this is carried out by the critic, while the actor, i.e. the policy, maintains a separate set of weights that are trained by following a gradient that depends on the critic’s advantage estimate. In practice, the gradient must be estimated from trajectories sampled from the environment, and the (action-)value functions must be estimated with function approximators. Consequently, the bias and variance of the gradient estimate depends strongly on the exact choice of estimator (Konda & Tsitsiklis 1999).

In this work we train critics $f^c(\cdot, \theta^c)$ using Sarsa($\lambda$) adapted for use with deep neural networks and multiple agents. In particular the critic parameters $\theta^c$ are updated to minimise the following loss:

$$
L^c_t(\theta^c) = (y^{(\lambda)} - f^c(t, \theta^c))^2,
$$

where $y^{(\lambda)}$ are the TD($\lambda$) targets, calculated using a target network with parameters copied periodically from $\theta^c$ as in DQN.

2.5 Multi-agent Reinforcement Learning under partial observability

We will now extend the formulations presented above to the multi-agent domain. We consider a fully cooperative multi-agent task that can be described as a stochastic game $G$, defined by a tuple $G = (S, U, P, R, Z, O, n, \gamma)$, in which $n$ agents identified by $a \in A = \{1, \ldots, n\}$ choose sequential actions. The environment has a true state $s \in S$. At each time step, each agent takes an action $u^a \in U$, forming a joint action $u \in U = U^n$ which induces a transition in the environment according to the state transition function $P(s'|s, u) : S \times U \times S \rightarrow [0, 1]$. The agents all share the same reward function $R(s, u) : S \times U \rightarrow \mathbb{R}$ and $\gamma \in [0, 1]$ is a discount factor.

We consider a partially observable setting, in which agents draw observations $z \in Z$ according to the observation function $O(s, a) : S \times A \rightarrow Z$. Each agent has an action-observation history $\tau^a \in T = (Z \times U)^*$, on which it conditions a stochastic policy $\pi^a(\tau^a) : T \times U \rightarrow [0, 1]$. We denote joint quantities over agents in bold, and joint quantities over agents other than a given agent $a$ with the superscript $-a$.

The discounted return is, as in the single-agent case, $G_t = \sum_{l=0}^{T} \gamma^l r_{t+l}$. The agents’ joint policy induces a value function, $V^\pi(s_t) = E_{s_{t+1:T}, u_{t:T}} [G_t|s_t]$, and an action-value function $Q^\pi(s_t, u_t) = \mathbb{E}_{s_{t+1:T}, u_{t+1:T}} [G_t|s_t, u_t]$. The advantage function is given by $A^\pi(s_t, u_t) = Q^\pi(s_t, u_t) - V^\pi(s_t)$.

Following previous work (Oliehoek et al. 2008; Kraemer & Banerjee 2016; Foerster et al. 2016; Jorge et al. 2016), our problem setting allows centralised training but requires decentralised execution. This is a natural paradigm for a large set of multi-agent problems where training is carried out using a simulator with additional state information, but the agents must rely on local action-observation histories during execution. To condition on this full history, a deep RL agent may make use of a recurrent neural network (Hausknecht & Stone 2015), typically making use of a gated model such as LSTM (Hochreiter & Schmidhuber 1997) or GRU (Chung et al. 2014). In Section 3 we develop a new multi-agent policy gradient method for tackling this setting.

3 Methods

We can now describe our approaches for extending the policy gradient methods we introduced above to the multi-agent domain.

3.1 Independent Actor-Critic

The most straightforward way to use policy gradient methods with multiple agents is to assign separate actors and critics to each agent and have all of them learn independently, from their own action-observation histories. This is essentially the idea behind independent Q-learning (Ian 1993), which is perhaps the most popular multi-agent reinforcement learning approach, but with actor-critic replacing Q-learning as the learning algorithm. Therefore, we call this approach independent actor-critic (IAC).

In our implementation of IAC, we speed up learning by sharing parameters among the agents, i.e., we learn only one actor and one critic, which are used by all agents. The agents can still behave
differently because they receive different observations and we further facilitate this by including an agent-specific ID in them. Learning remains independent in the sense that each agent’s critic estimates only a local value function, i.e., one that conditions on \( u^a \) and not on the joint action \( u \). Although we are not aware of any previous applications of this algorithm, we do not consider it a significant contribution and will instead use it as a baseline.

We consider two variants of IAC. In the first one, which we call IAC-\( V \), each agent’s critic estimates the value function \( V(\tau^a) \) and produces a gradient based on the TD error, as described in Section 2. In the second one, each agent’s critic estimates the individual action-value function \( Q(\tau^a, u^a) \) and produces a gradient based on the advantage: \( A(\tau^a, u^a) = Q(\tau^a, u^a) - V(\tau^a) \), where \( V(\tau^a) = \sum_{a^a} \pi(a^a | \tau^a) Q(\tau^a, u^a) \). We will refer to this variant as IAC-\( Q \). Independent learning is straightforward, but the lack of information sharing between the agents at training time makes it difficult for them to learn coordinated strategies that depend on interactions between them, or for an individual agent to estimate the contribution of its actions to the team’s reward.

### 3.2 Counterfactual Multi-Agent Policy Gradients

The difficulties discussed above arise because IAC fails to exploit the fact that our setting allows for the learning to be centralised. We will now present our method, counterfactual multi-agent (COMA) policy gradients, which overcomes this limitation. There are three main ideas behind it: The centralisation of the critic, the use of a counterfactual baseline, and a critic representation that allows the baseline to be evaluated efficiently. The remainder of this section describes them.

First, COMA uses a centralised critic. Note that in IAC, each actor \( \pi(u^a | \tau^a) \) and each critic \( Q(\tau^a, u^a) \) or \( V(\tau^a) \) conditions only on the agent’s own action-observation history \( \tau^a \). However, the critic is used only during learning and only the actor is needed during execution. Since learning is centralised, we can use a centralised critic that conditions on the true global state \( s \), or/and the joint action-observation histories \( \tau \). Each actor conditions on its own action-observation history \( \tau^a \), with parameter sharing, as in IAC. Figure 1 illustrates this setup.

![Figure 1](image)

(A) Information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.

A naive way to use this centralised critic would be to calculate the gradient based on the TD error:

\[
\nabla \theta^\pi = \frac{\partial}{\partial \theta^\pi} \log \pi(u | \tau^a) \left( r + \gamma V(s_{t+1}) - V(s_t) \right).
\]

(5)

However, such an approach fails to address a key credit assignment problem. Because the TD error considers only global rewards, the gradient computed for each actor does not explicitly reason about how that particular agent’s actions contribute to that global reward. Since the other agents may be exploring, this gradient can become very noisy, particularly when there are many agents.

To address this, we use a counterfactual baseline, which is inspired by difference rewards (Tumer & Agogino 2007). There, each agent learns from a shaped reward \( D^a = r(s, u) - r(s, (u^{-a}, c^a)) \) that compares the global reward to the reward received when the action of agent \( a \) is replaced with a default action \( c^a \). Any action by agent \( a \) that improves \( D^a \) also improves the true global reward \( r(s, u) \),

---

**Figure 1:** In (a), information flow between the decentralised actors, the environment and the centralised critic in COMA; red arrows and components are only required during centralised learning. In (b) and (c), architectures of the actor and critic.
since \( r(s, (u^{-a}, c^a)) \) does not depend on agent \( a \)'s actions. Difference rewards are a powerful way to perform multi-agent credit assignment, however the estimation of \( r(s, (u^{-a}, c^a)) \) typically requires access to a simulator, which is not always available. Even in the cases where one is already being used for learning, this method increases the number of simulations needed, since each agent’s difference reward requires a separate counterfactual simulation. Furthermore, the choice of \( c^a \) is unclear in many applications.

COMA uses a centralized critic to implement difference rewards in a way that avoids the above problems. The critic learns the action-value function \( Q(s, u) \) that estimates \( Q \)-values for the joint action \( u \) conditioned on the central state \( s \). For each agent \( a \) we can then compute an advantage function that compares the \( Q \)-value for action \( u^a \) that the agent took to a counterfactual baseline that marginalises out \( u^a \), while keeping the other agents’ actions \( u^{-a} \) fixed:

\[
A^a(s, u^a) = Q(s, u) - \sum_{u^a} \pi^a(u^a|\tau^a)Q(s, (u^{-a}, u^a)).
\]  

(6)

This advantage function replaces extra simulations with evaluations of the critic, however those evaluations may themselves be expensive when the critic is a deep neural network. Furthermore, in a typical representation, the number of output nodes of such a network would equal \(|U|^n\), the size of the joint action space, making it impractical to train. We address both issues, by using a cretil routine that allows for efficient evaluation of the baseline. In particular, the actions of the other agents, \( u^{-a} \), are passed as part of the input to the network, which outputs a \( Q \)-value for each of agent \( a \)'s actions, as shown in Figure 1b. Consequently, the counterfactual advantage can be calculated efficiently by a single forward pass of the actor and critic, for each agent. The number of outputs is therefore only \(|U|\). One caveat with this approach is that the resulting network has an input space that scales linearly in the number of agents and actions, however deep neural networks typically generalize well across such spaces.

**Algorithm 1** Counterfactual Multi-Agent (COMA) Policy Gradients

Initialise \( \hat{\theta}^c_i, \hat{\theta}^a_i, \theta^\pi \)

for each training episode \( e \) do

Empty buffer

for \( c_c = 1 \) to \( \text{BatchSize} \) do

for each agent \( a \) do

\( s_1 = \text{initial state}, t = 0, h^a_0 = 0 \)

while \( s_t \neq \text{terminal} \) and \( t < T \) do

\( t = t + 1 \)

for each agent \( a \) do

\( h^a_t = \text{Actor}(\pi^a_t, h^a_{t-1}, u^a_{t-1}, a, u; \theta^c_t) \)

Sample \( u^a_t \) from \( \pi(h^a_t, c(e)) \)

Get reward \( r_t \) and next state \( s_{t+1} \)

Add episode to buffer

Collate episodes in buffer into single batch

for \( t = 1 \) to \( T \) do // from now processing all agents in parallel via single batch

Batch unroll RNN using states, actions and rewards

Calculate TD(\( \lambda \)) targets \( y^G_t \) using \( \hat{\theta}^c_t \)

for \( t = T \) down to \( 1 \) do

\( \Delta Q^a_t = y^G_t - Q(s^a_t, u) \)

\( \nabla \theta^c = \nabla \theta^c + \frac{\Delta Q^a_t}{\text{BatchSize}} \) // calculate critic gradient

\( \hat{\theta}^c_{t+1} = \hat{\theta}^c_t + \alpha \nabla \theta^c \) // update critic weights

Every C steps reset \( \hat{\theta}^c_t = \theta^c_t \)

for \( t = T \) down to \( 1 \) do

\( A^a(s^a_t, u) = Q(s^a_t, u) - \sum_u Q(s^a_t, u, u^{-a})\pi(u|\tau_t) \) // calculate COMA

\( \nabla \theta^a = \nabla \theta^a + \frac{\Delta \theta^a_t}{\text{BatchSize}} \log \pi(u|\tau_t)A^a(s^a_t, u) \) // accumulate actor gradients

\( \theta^a_{t+1} = \theta^a_t + \alpha \nabla \theta^a \) // update actor weights

6
4 Experimental Setup

In this section, we describe the StarCraft testbed on which we evaluate COMA, as well as provide details about the state features, network architectures, training regimes, and ablations we used.

**Decentralised StarCraft Micromanagement.** StarCraft is a rich environment with stochastic dynamics that cannot be easily emulated. Many simpler multi-agent settings, such as Predator-Prey (Tan 1993) or Packet World (Weyns et al. 2005), by contrast, have full simulators with controlled randomness that can be freely set to any state in order to perfectly replay experiences. This makes it possible, though computationally expensive, to compute difference rewards via extra simulations. In StarCraft, as in the real world, this is not possible.

We focus on the micromanagement problem in StarCraft, which refers to the low-level control of individual units’ positioning and attack commands as they fight enemies. This task is naturally represented as a multi-agent system, where each StarCraft unit is replaced by a decentralised controller. We consider several scenarios with symmetric teams, consisting of: 3 marines (3m), 5 marines (5m), 5 wraiths (5w), or 2 dragoons with 3 zealots (2d,3z). The enemy team is controlled by the StarCraft AI, which uses a set of reasonable but suboptimal hand-crafted heuristics.

We allow the agents to choose from a set of discrete actions: move[direction], attack[enemy_id], stop, and noop. In the StarCraft game, when a unit selects an attack action, it first moves into attack range before firing, using the game’s built-in pathfinding to choose a route. These powerful attack-move macro-actions make the control problem considerably easier.

To create a more challenging benchmark that is meaningfully decentralised, we impose a restricted field of view on the agents, equal to the firing range of the weapons of the ranged units, as shown in Figure 2. This departure from the standard setup for centralised StarCraft control has three effects. First, it introduces significant partial observability. Second, it means units can only attack when they are in range of enemies, removing access to the StarCraft macro-actions. Third, agents cannot distinguish between enemies who are dead and those who are out of range and so can issue invalid attack commands at such enemies, which results in no action being taken. This substantially increases the average size of the action space, which in turn increases the difficulty of both exploration and control.

Under these difficult conditions, scenarios with even relatively small numbers of units become much harder to solve. As seen in Table 1, we compare against a simple hand-coded heuristic that instructs the agents to run forwards into range and then focus their fire, attacking each enemy in turn until it dies. This heuristic achieves a 98% win rate on m5 with a full field of view, but only 66% in our setting. To perform well in this task, the agents must learn to cooperate by positioning properly and focussing their fire, while remembering which enemy and ally units are alive or out of view.

All agents receive the same global reward at each time step, equal to the sum of damage inflicted on the opponent units minus half the damage taken. Killing an opponent generates a reward of 10 points, and winning the game generates a reward equal to the team’s remaining total health plus 200. This damage-based reward signal is comparable to that used by Usunier et al. (2016). Unlike Peng et al. (2017), our approach does not require estimating local rewards.

**State Features.** The actor and critic receive different input features, corresponding to local observations and global state, respectively. Both include features for allies and enemies. Units can be either allies or enemies, while agents are the decentralised controllers that command ally units.

The local observations for every agent are drawn only from a circular subset of the map centred on the unit it controls and include for each unit within this field of view: distance, relative x, relative y, unit type and shield. All features are normalized by their maximum values. Unlike Usunier et al. (2016), we do not include any information about the units’ current target. The global state representation consists of similar features, but for all units on the map regardless of fields of view. Absolute distance is not included, and x-y locations are given relative to the

---

2 After firing, a unit’s cooldown is reset, and it must drop before firing again. Shields absorb damage until they break, after which units start losing health. Dragoons and zealots have shields but marines do not.
centre of the map rather than to a particular agent. The global state also includes health points and cooldown for all agents. The representation fed to the centralised \( Q \)-function critic is the concatenation of the global state representation with the local observation of the agent whose actions are being evaluated. Our centralised critic that estimates \( V(s) \), and is therefore agent-agnostic, receives the global state concatenated with all agents’ observations. The observations contain no new information but include the egocentric distances relative to that agent.

**Architecture & Training.** The actor consists of 128-bit gated recurrent units (GRUs) (Cho et al., 2014) that use fully connected layers both to process the input and to produce the output values from the hidden state, \( h^a_t \). The IAC critics use extra output heads appended to the last layer of the actor network. Action probabilities are produced from the final layer, \( z \), via a bounded softmax distribution that lower-bounds the probability of any given action by \( \epsilon/|U|: P(u) = (1 - \epsilon) \text{softmax}(z)_u + \epsilon/|U| \). We anneal \( \epsilon \) linearly from 0.5 to 0.02 across 750 \( \frac{\text{BatchSize}}{\text{NumAgents}} \) training episodes. The centralised critic is a feedforward network with multiple ReLU layers combined with fully connected layers.

Training is performed in batch mode, with a batch size of 30. Due to parameter sharing, all agents can be processed in parallel, with each agent for each episode and time step occupying one batch entry. The training cycle progresses in three steps (completion of all three steps constitutes as one episode in our graphs), as outlined in algorithm 1. First, \( \frac{30}{\text{NumAgents}} \) episodes are collected. Second, the critic training loop is executed, applying a gradient update to the feed-forward critic for each time step, starting at the end of the episode. Last, the actor is trained, by fully unrolling its recurrent part, aggregating the gradients in the backward pass across all time steps, and applying a gradient update.

Hyperparameters were coarsely tuned on the m5 scenario and then used for all other maps. We found that the most sensitive parameter was TD(\( \lambda \)), but settled on \( \lambda = 0.8 \) which worked best for both COMA and our baselines. We use a target network for the critic, which updates every 150 training steps for the feed-forward centralised critics and every 50 steps for the recurrent IAC critics. The feed-forward critic receives more learning steps, since it performs a parameter update for each timestep. Both the actor and the critic networks are trained using RMS-prop with learning rate 0.0005 and alpha 0.99, without weight decay. We set gamma to 0.99 for all maps.

Our implementation uses TorchCraft (Synnaeve et al., 2016) and Torch 7 (Collobert et al., 2011). Although tuning the skip-frame in StarCraft can improve absolute performance (Peng et al., 2017), we use a default value of 7, since the main focus is a relative evaluation between COMA and the baselines.

**Ablations.** We perform ablation experiments to validate the three key elements of COMA that we highlighted. First, we test the importance of centralising the critic by comparing against the two IAC variants we introduced in section 3, namely IAC-\( Q \) and IAC-\( V \). These take the same decentralised input as the actor, and share the actor network parameters up to the final layer. IAC-\( Q \) then outputs \( |U| \) \( Q \)-values, one for each action, while IAC-\( V \) outputs a single state-value. Second, we test the significance of learning a state-action value function \( Q \) instead of a value function \( V \). The method central-\( V \) uses a central state for the critic, similarly to COMA, but learns a value function \( V(s) \), and uses the TD error to estimate the advantage for policy gradient updates according to equation 6. Last, we test the utility of our counterfactual baseline. The method central-\( QV \) learns \( Q \) in the same way as COMA but also maintains a separate network \( V \) which conditions only on the global state and not the observation of the respective agent (hence is the same for all agents due to parameter sharing) and simultaneously estimates the advantage as \( Q - V \), replacing our counterfactual baseline with \( V \). All methods use the same architecture and training scheme for the actors, and all critics are trained with TD(\( \lambda \)).

5 Results

Figure 3 shows the average win rates as a function of the number of training episodes for each method and each StarCraft scenario. For each method, we conducted 35 independent trials and froze learning every 100 training episodes to evaluate the learned policies across 200 episodes per method. We plot the average across episodes and trials. Also shown is one standard deviation in performance. The results show that COMA is superior to the IAC baselines in all scenarios.

---

35w DQN and GMEZO benchmark performances are of a policy trained on a larger map and tested on 5w.
Table 1: Mean win rates averaged across final 1000 evaluation episodes for the different maps, for all methods and the hand-coded heuristic in the decentralised setting with a limited field of view. The best result for this setting is in bold. Also shown, maximum win rates for COMA (decentralised), in comparison to the heuristic and published results (evaluated in the centralised setting).

<table>
<thead>
<tr>
<th>map</th>
<th>IAC-V</th>
<th>cnt-V</th>
<th>cnt-QV</th>
<th>COMA_mean</th>
<th>COMA_best</th>
<th>best</th>
</tr>
</thead>
<tbody>
<tr>
<td>3m</td>
<td>.35</td>
<td>.47</td>
<td>.56</td>
<td>.83</td>
<td>.87</td>
<td>.98</td>
</tr>
<tr>
<td>5m</td>
<td>.66</td>
<td>.63</td>
<td>.58</td>
<td>.67</td>
<td>.71</td>
<td>.81</td>
</tr>
<tr>
<td>5w</td>
<td>.70</td>
<td>.18</td>
<td>.57</td>
<td>.65</td>
<td>.76</td>
<td>.82</td>
</tr>
<tr>
<td>2d_3z</td>
<td>.63</td>
<td>.27</td>
<td>.19</td>
<td>.36</td>
<td>.39</td>
<td>.47</td>
</tr>
</tbody>
</table>

Figure 3: Win rates for COMA and competing algorithms on four different scenarios. COMA outperforms all baseline methods. Centralised critics also clearly outperform their decentralised counterparts. The legend at the top applies across all plots.

Interestingly, the IAC methods eventually learn reasonable policies in m5v5, although they need substantially more episodes to do so. This may seem counter-intuitive since in the IAC methods, the actor and critic networks share parameters in their early layers (see Section 4). One would expect that to speed up learning, but these results suggest that the improved accuracy of policy evaluation made possible by conditioning on the global state outweighs the overhead of training a separate network.

Furthermore, it is apparent that COMA strictly dominates central-QV, both in terms of training speed and in final performance on all scenarios. This strongly indicates that our counterfactual baseline is crucial when using a central Q-critic to train decentralised policies.

Last, we find that COMA outperforms our baseline central-V in final performance, even though learning a state-value function gives the latter the advantage of not conditioning on the joint action.
COMA typically achieves good policies faster, which supports our hypothesis that COMA provides a shaped training signal. Training is also more stable than with central-$V$, which we believe to be a consequence of the COMA gradient tending to zero as the policy becomes greedy. Overall, COMA is the best performing and most consistent method. [Usunier et al. (2016)] report the performance of their best agents trained with their state-of-the-art centralised controller labelled GMEZO (greedy-MDP with episodic zero-order optimisation), and for a centralised DQN controller. Both are given a full field of view and access to the attack-move macro-actions we described in section 4. We compare out best agents (over the 35 trials) trained with COMA against these results in Table 1. Although our agents are restricted to decentralized policies and partial observability in the form of the local field of view, they achieve performances comparable to the best published win rates.

6 Conclusions & Future Work

We presented COMA, a method that uses a centralised critic in order to estimate a counterfactual advantage used for training decentralised actors in multi-agent RL. COMA addresses the challenges of multi-agent credit assignment by using a counterfactual baseline that marginalises out a single agent’s action, while keeping the other agents’ actions fixed. Our results in a decentralised StarCraft unit micromanagement benchmark show that it significantly improves final performance and training speed over other multi-agent actor-critic methods and remains competitive with state-of-the-art centralised controllers under best-performance reporting. An interesting future direction is scaling COMA to scenarios with large numbers of agents, where centralised critics are more difficult to train and exploration is harder to coordinate.
Bibliography


