Asynchronous Coagent Networks

James E. Kostas [*] [1] **Chris Nota** [*] [1] **Philip S. Thomas** [1]

Abstract

Coagent policy gradient algorithms (CPGAs) are reinforcement learning algorithms for training a class of stochastic neural networks called *coagent networks*. In this work, we prove that CPGAs converge to locally optimal policies. Additionally, we extend prior theory to encompass *asynchronous* and *recurrent* coagent networks. These extensions facilitate the straightforward design and analysis of hierarchical reinforcement learning algorithms like the option-critic, and eliminate the need for complex derivations of customized learning rules for these algorithms.

1. Introduction

Reinforcement learning (RL) policies are often represented by *stochastic neural networks* (SNNs). SNNs are networks where the outputs of some units are not deterministic functions of the units' inputs. Several classes of algorithms from various branches of RL research, such as those using options (Sutton et al., 1999) or hierarchical architectures (Bacon et al., 2017), can be formulated as using SNN policies (see Section 2 for more examples). In this paper we study the problem of deriving learning rules for RL agents with SNN policies.

Coagent networks are one formulation of SNN policies for RL agents (Thomas & Barto, 2011). Coagent networks are comprised of conjugate agents, or *coagents*; each coagent is an RL algorithm learning and acting cooperatively with the other coagents in its network. In this paper, we focus specifically on the case where each coagent is a policy gradient RL algorithm, and call the resulting algorithms *coagent policy gradient algorithms* (CPGAs). Intuitively, CPGAs cause each individual coagent to view the other coagents as part of the environment. That is, individual coagents learn and interact with the combination of both the environment and the other coagents as if this combination was a single environment.

Typically, algorithm designers using SNNs must create specialized training algorithms for their architectures and prove the correctness of these algorithms. Coagent policy gradient algorithms (CPGAs) provide an alternative: they allow algorithm designers to easily derive stochastic gradient update rules for the wide variety of policy architectures that can be represented as SNNs. To analyze a given policy architecture (SNN), one must simply identify the inputs and outputs of each coagent in the network. The theory in this paper then immediately provides a policy gradient (stochastic gradient ascent) learning rule for each coagent, providing a simple mechanism for obtaining convergent update rules for complex policy architectures. This process is applicable to SNNs across several branches of RL research.

This paper also extends that theory and the theory in prior work to encompass *recurrent* and *asynchronous* coagent networks. A network is *recurrent* if it contains cycles. A network is *asynchronous* if units in the neural network do not execute simultaneously or at the same rate. The latter property allows distributed implementations of large neural networks to operate asynchronously. Additionally, a coagent network's capacity for *temporal abstraction* (learning, reasoning, and acting across different scales of time and task) may be enhanced, not just through the network topology, but by designing networks where different coagents execute at different rates. Furthermore, these extensions facilitate the straightforward design and analysis of hierarchical RL algorithms like the option-critic.

The contributions of this paper are: **1)** a complete and formal proof of a key CPGA result on which this paper relies (prior work provides an informal and incomplete proof), **2)** a generalization of the CPGA framework to handle asynchronous recurrent networks, **3)** empirical support of our theoretical claims regarding the gradients of asynchronous CPGAs, and **4)** a proof that asynchronous CPGAs generalize the option-critic framework, which serves as a demonstration of how CPGAs eliminate the need for the derivation of custom learning rules for architectures like the option-critic. Our simple mechanistic approach to gradient derivation for the option-critic is a clear example of the usefulness of the coagent framework to any researcher or algorithm designer

[*] Equal contribution [1] College of Information and Computer Sciences, University of Massachusetts, Amherst, MA, USA. Correspondence to: James E. Kostas <jekostas@umass.edu>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020. Copyright 2020 by the author(s).

creating or analyzing stochastic networks for RL.

2. Related Work

Klopf (1982) theorized that traditional models of classical and operant conditioning could be explained by modeling biological neurons as *hedonistic*, that is, seeking excitation and avoiding inhibition. The ideas motivating coagent networks bear a deep resemblance to Klopf's proposal.

Stochastic neural networks have applications dating back at least to Marvin Minsky's *stochastic neural analog reinforcement calculator*, built in 1951 (Russell & Norvig, 2016). Research of stochastic learning automata continued this work (Narendra & Thathachar, 1989); one notable example is the *adaptive reward-penalty* learning rule for training stochastic networks (Barto, 1985). Similarly, Williams (1992) proposed the well-known REINFORCE algorithm with the intent of training stochastic networks. Since then, REINFORCE has primarily been applied to deterministic networks. However, Thomas (2011) proposed CPGAs for RL, building on the original intent of Williams (1992).

Since their introduction, CPGAs have proven to be a practical tool for defining and improving RL agents. CPGAs have been used to discover "motor primitives" in simulated robotic control tasks and to solve RL problems with high-dimensional action spaces (Thomas & Barto, 2012). They are the RL precursor to the more general stochastic computation graphs.

The formalism of *stochastic computation graphs* was proposed to describe networks with a mixture of stochastic and deterministic nodes, with applications to supervised learning, unsupervised learning, and RL (Schulman et al., 2015). Several recently proposed approaches fit into the formalism of stochastic networks, but the relationship has frequently gone unnoticed. One notable example is the *option-critic* architecture (Bacon et al., 2017). The option-critic provides a framework for learning *options* (Sutton et al., 1999), a type of high-level and temporally extended action, and how to choose between options. Below, we show that this framework is a special case of coagent networks. Subsequent work, such as the *double actor-critic architecture for learning options* (Zhang & Whiteson, 2019) and the *hierarchical option-critic* (Riemer et al., 2018) also fit within and can be informed by the coagent framework.

The Horde architecture (Sutton et al., 2011) is similar to the coagent framework in that it consists of independent RL agents working cooperatively. However, the Horde architecture does not cause the collection of all agents to follow the gradient of the expected return with respect to each agent's parameters—the property that allows us to show the convergence of CPGAs.

CPGAs can be viewed as a special case of *multi-agent reinforcement learning* (MARL), the application of RL in settings where multiple agents exist and interact. However, MARL differs from CPGAs in that MARL agents typically have separate manifestations within the environment; additionally, the objectives of the MARL agents may or may not be aligned. Working within the MARL framework, researchers have proposed a variety of principled algorithms for cooperative multi-agent learning (Guestrin et al., 2002; Zhang et al., 2007; Liu et al., 2014). An overview of MARL is given by Buşoniu et al. (2010).

3. Background

We consider an MDP, $M = (\mathcal{S}, \mathcal{A}, \mathcal{R}, P, R, d_0, \gamma)$, where $\mathcal{S}$ is the finite set of possible *states*, $\mathcal{A}$ is the finite set of possible *actions*, and $\mathcal{R}$ is the finite set of possible *rewards* (although this work extends to MDPs where these sets are infinite and uncountable, the assumption that they are finite simplifies notation). Let $t \in \{0, 1, 2, \dots\}$ denote the time step. S_t, A_t, and R_t are the state, action, and reward at time t, and are random variables that take values in $\mathcal{S}$, $\mathcal{A}$, and $\mathcal{R}$, respectively. $P{:}\mathcal{S}{\times}\mathcal{A}{\times}\mathcal{S}{\to}[0,1]$ is the *transition function*, given by $P(s,a,s') \coloneqq \Pr(S_{t+1}{=}s'|S_t{=}s, A_t{=}a)$. $R{:}\mathcal{S}{\times}\mathcal{A}{\times}\mathcal{S}{\times}\mathcal{R}{\to}[0,1]$ is the *reward distribution*, given by $R(s,a,s',r) \coloneqq \Pr(R_t{=}r|S_t{=}s, A_t{=}a, S_{t+1}{=}s')$. The *initial state distribution*, $d_0{:}\mathcal{S} \to [0,1]$, is given by $d_0(s) \coloneqq \Pr(S_0{=}s)$. The reward discount parameter is $\gamma \in [0,1]$. An *episode* is a sequence of states, actions, and rewards, starting from $t{=}0$ and continuing indefinitely. We assume that the discounted sum of rewards over an episode is finite.

A policy, $\pi : \mathcal{S} \times \mathcal{A} \to [0,1]$, is a stochastic method of selecting actions, such that $\pi(s,a) \coloneqq \Pr(A_t{=}a|S_t{=}s)$. A *parameterized policy* is a policy that takes a parameter vector $\theta \in \mathbb{R}^n$. Different parameter vectors result in different policies. More formally, we redefine the symbol π to denote a parameterized policy, $\pi : \mathcal{S} \times \mathcal{A} \times \mathbb{R}^n \to [0,1]$, such that for all $\theta \in \mathbb{R}^n$, $\pi(\cdot, \cdot, \theta)$ is a policy. We assume that $\partial\pi(s,a,\theta)/\partial\theta$ exists for all $s \in \mathcal{S}, a \in \mathcal{A}$, and $\theta \in \mathbb{R}^n$. An agent's goal is typically to find a policy that maximizes the *objective function* $J(\pi) \coloneqq \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t | \pi\right]$, where conditioning on π denotes that, for all t, $A_t \sim \pi(S_t, \cdot)$. The state-value function, $v^\pi : \mathcal{S} \to \mathbb{R}$, is defined as $v^\pi(s) \coloneqq \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \middle| S_t{=}s, \pi\right]$. The discounted return, G_t, is defined as $G_t \coloneqq \sum_{k=0}^{\infty} \gamma^k R_{t+k}$. We denote the objective function for a policy that has parameters θ as $J(\theta)$, and condition probabilities on θ to denote that the parameterized policy uses parameter vector θ.

A *coagent network* is a parameterized policy that consists of an acyclic network of nodes (coagents), which do not share parameters. Each coagent can have several inputs that may include the state at time t, a noisy and incomplete observation of the state at time t, and/or the outputs of

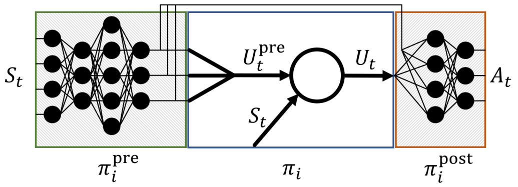

Figure 1. Diagram of the three-step process for action generation for a fully connected feedforward network (we do not require the network to have this structure). The circle in the middle denotes the i^{th} coagent. First, preceding nodes are executed to compute the inputs to this coagent. Second, the coagent uses these inputs to produce its output, U_t. Third, the remainder of the network is executed to produce an action. For each coagent, all inputs are optional. That is, our approach encompasses architectures where S_t and/or components of U_t^{pre} are not provided to some coagents.

other coagents. When considering the i^{th} coagent, θ can be partitioned into two vectors, $\theta_i \in \mathbb{R}^{n_i}$ (the parameters used by the i^{th} coagent) and $\bar{\theta}_i \in \mathbb{R}^{n-n_i}$ (the parameters used by all other coagents). From the point of view of the i^{th} coagent, A_t is produced from S_t in three stages: execution of the nodes prior to the i^{th} coagent (nodes whose outputs are required to compute the input to the i^{th} coagent), execution of the i^{th} coagent, and execution of the remaining nodes in the network to produce the final action. This process is depicted graphically in Figure 1 and described in detail below.

First, we define a parameterized distribution $\pi_i^{\text{pre}}(S_t, \cdot, \bar{\theta}_i)$ to capture how the previous coagents in the network produce their outputs given the current state. The output of the previous coagents is a random variable, which we denote by U_t^{pre}, and which takes continuous and/or discrete values in some set $\mathcal{U}^{\text{pre}}$. U_t^{pre} is sampled from the distribution $\pi_i^{\text{pre}}(S_t, \cdot, \bar{\theta}_i)$. Next, the i^{th} coagent takes S_t and U_t^{pre} as input. We denote this input, (S_t, U_t^{pre}), as X_t (or X_t^i if it is not unambiguously referring to the i^{th} coagent), and refer to X_t as the *local state*. Given this input, the coagent produces the output U_t^i (below, when it is unambiguously referring to the output of the i^{th} coagent, we make the i implicit and write U_t). The conditional distribution of U_t^i is given by the i^{th} coagent's policy, $\pi_i(X_t, \cdot, \theta_i)$. Although we allow the i^{th} coagent's output to depend directly on S_t, it may be parameterized to only depend on U_t^{pre}. Finally, A_t is sampled according to a distribution $\pi_i^{\text{post}}(X_t, U_t^i, \cdot, \bar{\theta}_i)$, which captures how the subsequent coagents in the network produce A_t. Below, we sometimes make $\bar{\theta}_i$ and θ_i implicit and write the three policy functions as $\pi_i^{\text{pre}}(S_t, \cdot)$, $\pi_i(X_t, \cdot)$, and $\pi_i^{\text{post}}(X_t, U_t^i, \cdot)$. Figure 2 depicts the setup that we have described and makes relevant independence properties clear.

4. The Coagent Policy Gradient Theorem

Consider what would happen if the i^{th} coagent ignored all of the complexity in this problem setup and simply learned and interacted with the combination of both the environment and the other coagents as if this combination was a single environment. From the i^{th} coagent's point of view, the actions would be U_t, the rewards would remain R_t, and the state would be X_t (that is, S_t and U_t^{pre}). Note that the coagent may ignore components of this local state, such as the $\mathcal{S}$ component. Each coagent could naively implement an unbiased policy gradient algorithm, like REINFORCE (Williams, 1992), based only on these inputs and outputs. We refer to the expected update in this setting as the *local policy gradient*, Δ_i, for the i^{th} coagent. Formally, the local policy gradient of the i^{th} coagent is:

$$\Delta_i(\theta_i) := \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t G_t \frac{\partial \ln\left(\pi_i\left(X_t, U_t, \theta_i\right)\right)}{\partial \theta_i} \middle| \theta\right].$$

The local policy gradient should not be confused with $\nabla J(\theta)$, which we call the *global policy gradient*, or with $\nabla J(\theta)$'s i^{th} component, $[\partial J(\theta)/\partial\theta_i]^\intercal$ (notice that $\nabla J(\theta) = \left[\frac{\partial J(\theta)}{\partial\theta_1}^\intercal, \frac{\partial J(\theta)}{\partial\theta_2}^\intercal, \dots, \frac{\partial J(\theta)}{\partial\theta_m}^\intercal\right]^\intercal$, where m is the number of coagents). Unlike a network following $\nabla J(\theta)$ or a coagent following its corresponding component of $\nabla J(\theta)$, a coagent following a local policy gradient faces a non-stationary sequence of partially-observable MDPs as the other coagents (part of its environment) update and learn. One could naively design an algorithm that uses this local policy gradient and simply hope for good results, but without theoretical analysis, this hope would not be justified.

The *coagent policy gradient theorem* (CPGT) justifies this approach: If θ is fixed and all coagents update their parameters following unbiased estimates, $\widehat{\Delta}_i(\theta_i)$, of their local policy gradients, then the entire network will follow an unbiased estimator of $\nabla J(\theta)$. For example, if every coagent performs the following update simultaneously at the end of each episode, then the entire network will be performing stochastic gradient ascent on J (without using backpropagation):

$$\theta_i \leftarrow \theta_i + \alpha \sum_{t=0}^{\infty} \gamma^t G_t \left(\frac{\partial \ln\left(\pi_i\left(X_t, U_t, \theta_i\right)\right)}{\partial \theta_i}\right).$$

In practice, one would use a more sophisticated policy gradient algorithm than this simple variant of REINFORCE.

Although Thomas & Barto (2011) present the CPGT in their Theorem 3, the provided proof is lacking in two ways. First, it is not general enough for our purposes because it only considers networks with two coagents. Second, it is missing a crucial step. They define a new MDP, the CoMDP, which models the environment faced by a coagent, but they do not show that this definition accurately describes

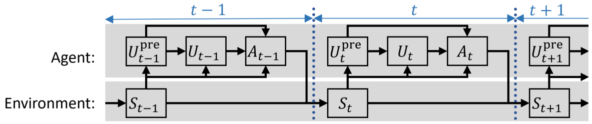

Figure 2. Bayesian network depicting the relationships of relevant random variables.

the environment that the coagent faces. Without this step, Thomas & Barto (2011) have shown that there is a new MDP for which the policy gradient is a component of $\nabla J(\theta)$, but not that this MDP has any relation to the coagent network.

4.1. The Conjugate Markov Decision Process (CoMDP)

In order to reason about the local policy gradient, we begin by modeling the i^{th} coagent's environment as an MDP, called the CoMDP. Given M, i, π_i^{pre}, π_i^{post}, and $\bar{\theta}_i$, we define a corresponding CoMDP, M^i, as $M^i := (\mathcal{X}^i, \mathcal{U}^i, \mathcal{R}^i, P^i, R^i, d_0^i, \gamma_i)$. Although our proof of the CPGT relies heavily on these definitions, due to space limitations the formal definitions for each of these terms are provided in Section A of the supplementary material. A brief summary of definitions follows.

We write $\tilde{X}_t^i$, $\tilde{U}_t^i$, and $\tilde{R}_t^i$ to denote the state, action, and reward of M^i at time t. These *random variables* in the CoMDP are written with tildes to provide a visual distinction between terms from the CoMDP and original MDP. Additionally, when it is clear that we are referring to the i^{th} CoMDP, we often make i implicit (for example, we might write $\tilde{X}_t^i$ as $\tilde{X}_t$).

The input (analogous to the state) to the i^{th} coagent is in the set $\mathcal{X}^i := \mathcal{S} \times \mathcal{U}_i^{\text{pre}}$. For $x \in \mathcal{X}$, we denote the $\mathcal{S}$ component as $x.s$ and the $\mathcal{U}^{\text{pre}}$ component as $x.u_{\text{pre}}$. We also sometimes denote an $x \in \mathcal{X}^i$ as $(x.s, x.u_{\text{pre}})$. $\mathcal{U}^i$ is an arbitrary set that denotes the output of the i^{th} coagent. $\mathcal{R}^i := \mathcal{R}$ and $\gamma_i := \gamma$ represent the reward set and the discount factor, respectively. We denote the transition function as $P^i(x, u, x')$, the reward function as $R^i(x, u, x', r)$, and initial state distribution as $d_0^i(x)$. We write $J_i(\theta_i)$ to denote the objective function of M^i.

4.2. The CoMDP Models the Coagent's Environment

Here we show that our definition of the CoMDP correctly models the coagent's environment. We do so by presenting a series of properties and lemmas that each establish different components of the relationship between the CoMDP and the environment faced by a coagent. The proofs of these properties and theorems are provided in Section B of the supplementary material.

In Properties 1 and 2, by manipulating the definitions of d_0^i and π_i^{pre}, we show that d_0^i and the distribution of $\tilde{X}_0.s$ capture the distribution of the inputs to the i^{th} coagent.

Property 1. $\forall x \in \mathcal{X}, d_0^i(x) = \Pr(S_0{=}x.s, U_0^{\text{pre}}{=}x.u_{\text{pre}})$.

Property 2. *For all* $s \in \mathcal{S}$, $\Pr(\tilde{X}_0.s{=}s) = d_0(s)$.

In Property 3, we show that P^i captures the distributions of the inputs that the i^{th} coagent will see given the input at the previous step and the output that it selected.

Property 3. *For all* $x \in \mathcal{X}$, $x' \in \mathcal{X}$, *and* $u \in \mathcal{U}$, $P^i(x, u, x') = \Pr(S_{t+1}{=}x'.s, U_{t+1}^{\text{pre}}{=}x'.u_{\text{pre}}$ $|S_t{=}x.s, U_t^{\text{pre}}{=}x.u_{\text{pre}}, U_t{=}u)$.

In Property 4, we show that R^i captures the distribution of the rewards that the i^{th} coagent receives given the output that it selected and the inputs at the current and next steps.

Property 4. *For all* $x \in \mathcal{X}$, $x' \in \mathcal{X}$, $u \in \mathcal{U}$, *and* $r \in \mathcal{R}$, $R^i(x, u, x', r) = \Pr(R_t{=}r|S_t{=}x.s, U_t^{\text{pre}}{=}x.u_{\text{pre}}, U_t{=}u,$ $S_{t+1}{=}x'.s, U_{t+1}^{\text{pre}}{=}x'.u_{\text{pre}})$.

In Properties 5 and 6, we show that the distributions of $\tilde{X}$ and $\tilde{X}_t.s$ capture the distribution of inputs to the i^{th} coagent.

Property 5. *For all* $s{\in}\mathcal{S}$ *and* $u_{\text{pre}}{\in}\mathcal{U}_i^{\text{pre}}$,

$$\Pr(\tilde{X}_t{=}(s, u_{\text{pre}})) = \Pr(S_t{=}s, U_t^{\text{pre}}{=}u_{\text{pre}}).$$

Property 6. *For all* $s \in \mathcal{S}$, $\Pr(\tilde{X}_t.s{=}s) = \Pr(S_t{=}s)$.

In Property 7, we show that the distribution of $\tilde{X}_t.u_{pre}$ given $\tilde{X}_t.s$ captures the distribution π_i^{pre}.

Property 7. *For all* $s \in \mathcal{S}$ *and* $u_{\text{pre}} \in \mathcal{U}_i^{\text{pre}}$, $\Pr(\tilde{X}_t.u_{\text{pre}}{=}u_{\text{pre}}|\tilde{X}_t.s{=}s) = \pi_i^{\text{pre}}(s, u_{\text{pre}})$.

In Property 8, we show that the distribution of $\tilde{X}_{t+1}.s$ given $\tilde{X}_t.s$, $\tilde{X}_t.u_{pre}$, and $\tilde{U}_t$ captures the distribution of the $\mathcal{S}$ component of the input that the i^{th} coagent will see given the input at the previous step and the output that it selected.

Property 8. *For all* $s \in \mathcal{S}$, $s' \in \mathcal{S}$, $u_{\text{pre}} \in \mathcal{U}_i^{\text{pre}}$, *and* $u \in \mathcal{U}$, $\Pr(\tilde{X}_{t+1}.s{=}s'|\tilde{X}_t.s{=}s, \tilde{X}_t.u_{\text{pre}}{=}u_{\text{pre}}, \tilde{U}_t{=}u)$ $= \Pr(S_{t+1}{=}s'|S_t{=}s, U_t^{\text{pre}}{=}u_{\text{pre}}, U_t{=}u)$.

In Property 9, we show that: Given the $\mathcal{S}$ component of the input, the $\mathcal{U}_i^{\text{pre}}$ component of the input that the i^{th} coagent will see is independent of the previous input and output.

Property 9. *For all* $s \in \mathcal{S}$, $s' \in \mathcal{S}$, $u_{\text{pre}} \in \mathcal{U}_i^{\text{pre}}$, $u'_{\text{pre}} \in \mathcal{U}_i^{\text{pre}}$, *and* $u \in \mathcal{U}$, $\Pr(\tilde{X}_{t+1}.u_{\text{pre}}{=}u'_{\text{pre}}|\tilde{X}_{t+1}.s{=}s')$ $= \Pr(\tilde{X}_{t+1}.u_{\text{pre}}{=}u'_{\text{pre}}|\tilde{X}_{t+1}.s{=}s', \tilde{X}_t{=}(s, u_{\text{pre}}), \tilde{U}_t{=}u)$.

In Property 10, we use Properties 6, 7, 8, 9, and 10 to show that the distribution of $\tilde{R}_t^i$ captures the distribution of the rewards that the i^{th} coagent receives.

Property 10. *For all* $r \in \mathcal{R}$, $\Pr(R_t{=}r) = \Pr(\tilde{R}_t^i{=}r)$.

We then use Properties 3 and 4 and the definition of M^i to show that:

Lemma 1. *M^i is a Markov decision process.*

Finally, in Lemma 2, we use the properties above to show that the CoMDP M^i (built from M, i, π_i^pre, π_i^post, and $\bar{\theta}_i$) correctly models the local environment of the i^th coagent.

Lemma 2. *For all $M, i, \pi_i^\text{pre}, \pi_i^\text{post}$, and $\bar{\theta}_i$, and given a policy parameterized by θ_i, the corresponding CoMDP M^i satisfies Properties 1-6 and Property 10.*

Lemma 2 is stated more specifically and formally in the supplementary material.

4.3. The Coagent Policy Gradient Theorem

Again, all proofs of the properties and theorems below are provided in Section B of the supplementary material. We use Property 10 to show that M's objective, $J(\theta)$, is equivalent to the objective $J_i(\theta_i)$ of the i^th CoMDP.

Property 11. *For all coagents i, for all θ_i, given the same $\theta = (\theta_i, \bar{\theta}_i)$, $J(\theta) = J_i(\theta_i)$.*

Next, using Lemmas 1 and 2, we show that the local policy gradient, Δ_i (the expected value of the naive REINFORCE update), is equivalent to the gradient $\frac{\partial J_i}{\partial \theta_i}$ of the i^th CoMDP.

Lemma 3. *For all coagents i, for all θ_i, $\frac{\partial J_i(\theta_i)}{\partial \theta_i} = \Delta_i(\theta_i)$.*

The CPGT states that the local policy gradients are the components of the global policy gradient. Notice that this intuitively follows by transitivity from Property 11 and Lemma 3.

Theorem 1 (Coagent Policy Gradient Theorem)**.** *$\nabla J(\theta) = [\Delta_1(\theta_1)^\intercal, \Delta_2(\theta_2)^\intercal, \dots, \Delta_m(\theta_m)^\intercal]^\intercal$, where m is the number of coagents and Δ_i is the local policy gradient of the i^{th} coagent.*

Corollary 1. *If α_t is a deterministic positive stepsize, $\sum_{t=0}^\infty \alpha_t = \infty$, $\sum_{t=0}^\infty \alpha_t^2 < \infty$, additional technical assumptions are met (Bertsekas & Tsitsiklis, 2000, Proposition 3), and each coagent updates its parameters, θ_i, with an unbiased local policy gradient update $\theta_i \leftarrow \theta_i + \alpha_t \widehat{\Delta}_i(\theta_i)$, then $J(\theta)$ converges to a finite value and $\lim_{t\to\infty}\nabla J(\theta)$=0.*

5. Asynchronous Recurrent Networks

Having formally established the CPGT, we now turn to extending the CPGA framework to asynchronous and cyclic networks—networks where the coagents *execute*, that is, look at their local state and choose actions, asynchronously and without any necessary order. This extension allows for distributed implementations, where nodes may not execute synchronously. This also facilitates temporal abstraction, since, by varying coagent execution rates, one can design algorithms that learn and act across different levels of abstraction.

We first consider how we may modify an MDP to allow coagents to execute at arbitrary points in time, including at points *in between* our usual time steps. Our motivation is to consider continuous time. As a theoretical construct, we can approximate continuous time with arbitrarily high precision by breaking a time step of the MDP into an arbitrarily large number of shorter steps, which we call *atomic time steps*. We assume that the environment performs its usual update regularly every $n \in \mathbb{Z}^+$ atomic time steps, and that each coagent *executes* (chooses an output in its respective $\mathcal{U}^i$) at each atomic time step with some probability, given by an arbitrary but fixed distribution that may be conditioned on the local state. On atomic time steps where the i^th coagent does not execute, it continues to output the last chosen action in $\mathcal{U}^i$ until its next execution. The duration of atomic time steps can be arbitrarily small to allow for arbitrarily close approximations to continuous time or to model, for example, a CPU cluster that performs billions of updates per second. The objective is still the expected value of G_0, the discounted sum of rewards from all atomic time steps: $J(\theta) = \mathbf{E}[G_0|\theta] = \mathbf{E}[\sum_{t=0}^\infty \gamma^t R_t|\theta]$. (Note that γ in this equation is the n^th root of the original γ, where n is the number of of atomic time steps per environment update.)

Next, we extend the coagent framework to allow *cyclic* connections. Previously, we considered a coagent's local state to be captured by $X_t^i = (S_t, U_t^\text{pre})$, where U_t^pre is some combination of outputs from coagents that come before the i^th coagent topologically. We now allow coagents to also consider the output of all m coagents on the *previous* time step, $U_{t-1}^\text{all} = (U_{t-1}^1, U_{t-1}^2, \dots U_{t-1}^m)$. In the new setting, the local state at time t is therefore given by $X_t^i = (S_t, U_t^\text{pre}, U_{t-1}^\text{all})$. The corresponding local state set is given by $\mathcal{X}^i = \mathcal{S} \times \mathcal{U}^\text{pre} \times \mathcal{U}^1 \times \dots \times \mathcal{U}^m$. In this construction, when $t = 0$, we must consider some initial output of each coagent, U_{-1}^all. For the i^th coagent, we define U_{-1}^i to be drawn from some initial distribution, h_0^i, such that for all $u \in \mathcal{U}^i, h_0^i(u) = \Pr(U_{-1}^i = u)$.

We redefine how each coagent selects actions in the asynchronous setting. First, we define a random variable, E_t^i, the value of which is 1 if the i^th coagent executes on atomic time step t, and 0 otherwise. Each coagent has a fixed *execution function*, $\beta_i : \mathcal{X}^i \times \mathbb{N} \to [0, 1]$, which defines the probability of the i^th coagent executing on time step t, given the coagent's local state. That is, for all $x \in \mathcal{X}^i$, $\beta_i(x) := \Pr(E_t^i = 1|X_t^i = x)$. Finally, the action that the i^th coagent selects at time t, U_t^i, is sampled from $\pi_i(X_t^i, \cdot, \theta_i)$ if $E_t^i = 1$, and is U_{t-1}^i otherwise. That is, if the agent does not execute on atomic time step t, then it should repeat its action from time $t - 1$.

We cannot directly apply the CPGT to this setting: the policy and environment are *non-Markovian*. That is, we cannot determine the distribution over the output of the network given only the current state, S_t, since the output may also depend on U^{all}_{t-1}. However, we show that the asynchronous setting can be reduced to the acyclic, synchronous setting using formulaic changes to the state set, transition function, and network structure. This allows us to derive an expression for the gradient with respect to the parameters of the original, asynchronous network, and thus to train such a network. We prove a result similar to the CPGT that allows us to update the parameters of each coagent using only states and actions from atomic time steps when the coagent executes.

5.1. The CPGT for Asynchronous Networks

We first extend the definition of the *local policy gradient*, Δ_i, to the asynchronous setting. In the synchronous setting, the local policy gradient captures the update that a coagent would perform if it was following an unbiased policy gradient algorithm using its local inputs and outputs. In the asynchronous setting, we capture the update that an agent would perform if it were to consider only the local inputs and outputs it sees when it executes. Formally, we define the *asynchronous local policy gradient*: $\Delta_i(\theta_i) \coloneqq \mathbf{E}\Big[\sum_{t=0}^{\infty} E^i_t \gamma^t G_t \frac{\partial \ln\left(\pi_i\left(X_t, U_t, \theta_i\right)\right)}{\partial \theta_i}\Big|\theta\Big]$.

The only change from the synchronous version is the introduction of E^i_t. Note that when the coagent does not execute ($E^i_t = 0$), the entire inner expression is 0. In other words, these states and actions can be ignored. An algorithm estimating G_t would still need to consider the rewards from *every* atomic time step, including time steps where the coagent does not execute. However, the algorithm may still be designed such that the coagents only perform a computation when executing. For example, during execution, coagents may be given the discounted sum of rewards since their last execution to serve as a summary of all rewards since that execution. The important question is then: does something like the CPGT hold for the *asynchronous* local policy gradient? If each coagent executes a policy gradient algorithm using unbiased estimates of Δ_i, does the network still perform gradient descent on the asynchronous setting objective, J? The answer turns out to be yes.

Theorem 2 (Asynchronous Coagent Policy Gradient Theorem)**.**
$\nabla J(\theta) = [\Delta_1(\theta_1)^\intercal, \Delta_2(\theta_2)^\intercal, \dots, \Delta_m(\theta_m)^\intercal]^\intercal$, where m is the number of coagents and Δ_i is the asynchronous local policy gradient of the i^{th} coagent.

Proof. The general approach is to show that for any MDP M, with an asynchronous network represented by π with parameters θ, there is an *augmented* MDP, $\grave{M}$, with objective $\grave{J}$ and an *acylic, synchronous* network, $\grave{\pi}$, with the same parameters θ, such that $J(\theta) = \grave{J}(\theta)$. Thus, we *reduce* the asynchronous problem to an equivalent synchronous problem. Applying the CPGT to this *reduced setting* allows us to derive Theorem 2.

The original MDP, M, is given by the tuple $(\mathcal{S}, \mathcal{A}, P, R, d_0, \gamma)$. We define the *augmented* MDP, $\grave{M}$, as the tuple, $(\grave{\mathcal{S}}, \grave{\mathcal{A}}, \grave{P}, \grave{R}, \grave{d}_0, \grave{\gamma})$. We would like $\grave{M}$ to hold *all* of the information necessary for each coagent to compute its next output, including the previous outputs of all coagents. This will allow us to construct an acyclic version of the network to which we may apply the CPGT. We define $\mathcal{U}^{\text{all}} = \mathcal{U}^1 \times \mathcal{U}^2 \times \cdots \times \mathcal{U}^m$ to be the combined output set of all m coagents in π and $\mathcal{E} = \{0, 1\}^m$ to be the set of possible combinations of coagent executions. We define the state set to be $\grave{\mathcal{S}} = \mathcal{S} \times \mathcal{U}^{\text{all}}$ and the action set to be $\grave{\mathcal{A}} = \mathcal{A} \times \mathcal{U}^{\text{all}} \times \mathcal{E}$. We write the random variables representing the state, action, and reward at time t as $\grave{S}_t$, $\grave{A}_t$, and $\grave{R}_t$ respectively. Additionally, we refer to the components of values $\grave{s} \in \grave{\mathcal{S}}$ and $\grave{a} \in \grave{\mathcal{A}}$ and the components of the random variables $\grave{S}_t$ and $\grave{A}_t$ using the same notation as for the components of X_t above (for example, $\grave{s}.s$ is the $\mathcal{S}$ component of $\grave{s}$, $\grave{A}_t.u^{\text{all}}$ is the $\mathcal{U}^{\text{all}}$ component of $\grave{A}_t$, etc.). For vector components, we write the i^{th} component of the vector using a subscript i (for example, $\grave{s}.u^{\text{all}}_i$ is the i^{th} component of $\grave{s}.u^{\text{all}}$).

The transition function, $\grave{P}$, captures the original transition function and the fact that $\grave{S}_{t+1}.u^{\text{all}} = \grave{A}_t.u^{\text{all}}$. For all $\grave{s}, \grave{s}' \in \grave{S}$ and $\grave{a} \in \grave{A}$, $\grave{P}(\grave{s}, \grave{a}, \grave{s}')$ is given by $P(\grave{s}.s, \grave{a}.a, \grave{s}'.s)$ if $\grave{s}'.u^{\text{all}}{=}\grave{a}.u^{\text{all}}$, and 0 otherwise. For all $\grave{s}, \grave{s}' \in \grave{S}$, $\grave{a} \in \grave{A}$, and $r \in \mathbb{R}$, the reward distribution is simply given by $\grave{R}(\grave{s}, \grave{a}, \grave{s}', r){=}R(\grave{s}.s, \grave{a}.a, \grave{s}'.s, r)$. The initial state distribution, $\grave{d}_0$, captures the original state distribution and the initialization of each coagent. For all $\grave{s}{\in}\grave{S}$, it is given by $\grave{d}_0(\grave{s}){=}d_0(\grave{s}.s)\prod_{i=1}^{m} h_0(\grave{s}.u_i)$. The discount parameter is $\grave{\gamma}{=}\gamma$. The objective is the usual: $\grave{J}(\theta){=}\mathbf{E}[\grave{G}_0|\theta]$, where $\grave{G}_0{=}\mathbf{E}[\sum_{t=0}^{\infty}\grave{\gamma}^t\grave{R}_t|\theta]$.

Next we define the synchronous network, $\grave{\pi}$, in terms of components of the original asynchronous network, π—specifically, each π_i, β_i, and θ_i. We must modify the original network to accept inputs in $\grave{\mathcal{S}}$ and produce outputs in $\grave{\mathcal{A}}$. Recall that in the asynchronous network, the local state at time t of the i^{th} coagent is given by $X^i_t = (S_t, U^{\text{pre}}_t, U^{\text{all}}_{t-1})$. In the augmented MDP, the information in U^{all}_{t-1} is contained in $\grave{S}_t$, so the local state of the i^{th} coagent in the synchronous network is $\grave{X}^i_t = (\grave{S}_t, \grave{U}^{\text{pre}}_t)$, with accompanying state set $\grave{\mathcal{X}}^i = \grave{\mathcal{S}} \times \grave{\mathcal{U}}^{\text{pre}}$. To produce the $\mathcal{U}^{\text{all}}$ component of the action, $\grave{A}_t.u^{\text{all}}$, we append the output of each coagent to the action. In doing so, we have removed the need for cyclic connections, but still must deal with the asynchronous execution.

The critical step is as follows: We represent each coagent in the asynchronous network by *two* coagents in the synchronous network, the first of which represents the execution function, β_i, and the second of which represents the original policy, π_i. At time step t, the first coagent accepts $\grave{X}_t^i$ and outputs 1 with probability $\beta_i((\grave{S}_t.s, \grave{U}_t^{\text{pre}}, \grave{S}_t.u))$, and 0 otherwise. We append the output of every such coagent to the action in order to produce the $\mathcal{E}$ component of the action, $\grave{A}_t.e$. Because the coagent representing β_i executes before the coagent representing π_i, from the latter's perspective, the output of the former is present in $\grave{U}_t^{\text{pre}}$, that is, $\grave{U}_t^{\text{pre}}.e_i = \grave{A}_t.e_i$. If $\grave{U}_t^{\text{pre}}.e_i = 1$, the coagent samples a new action from π_i. Otherwise, it repeats its previous action, which can be read from its local state (that is, $\grave{X}_t^i.u_i^{\text{all}} = \grave{U}_{t-1}^i$). Formally, for all $(\grave{s}, \grave{u}_{\text{pre}}) \in \grave{\mathcal{X}}^i$ and θ_i, the probability of the latter coagent producing action $\grave{u} \in \grave{\mathcal{U}}^i$ is given by: $\grave{\pi}_i((\grave{s}, \grave{u}_{\text{pre}}), \grave{u}, \theta_i) {:=} \pi_i((\grave{s}.s, \grave{u}_{\text{pre}}, \grave{s}.u^{\text{all}}), \grave{u}, \theta_i)$ if $\grave{u}_{\text{pre}}.e_i{=}1$, 1 if $\grave{u}_{\text{pre}}.e_i{=}0$ and $\grave{s}.u_i^{\text{all}}{=}\grave{u}$, and 0 otherwise. This completes the description of $\grave{\pi}$. In the supplementary material, we prove that this network exactly captures the behavior of the asynchronous network—that is, $\grave{\pi}((s,u),(a,u',e),\theta) = \Pr(A_t = a, U_t^{\text{all}} = u', E_t = e | S_t = s, U_{t-1}^{\text{all}} = u, \theta)$ for all possible values of $a, u, a, u', e,$ and θ in their appropriate sets.

The proof that $J(\theta) = \grave{J}(\theta)$ is given in Section C of the supplementary material, but it follows intuitively from the fact that 1) the "hidden" state of the network is now captured by the state set, 2) $\grave{\pi}$ accurately captures the dynamics of the hidden state, and 3) this hidden state does not materially affect the transition function or the reward distribution with respect to the original states and actions.

Having shown that the expected return in the asynchronous setting is equal to the expected return in the synchronous setting, we turn to deriving the asynchronous local policy gradient, Δ_i. It follows from $J(\theta) = \grave{J}(\theta)$ that $\nabla J(\theta) = \nabla \grave{J}(\theta)$. Since $\grave{\pi}$ is a synchronous, acylic network, and $\grave{M}$ is an MDP, we can apply the CPGT to find an expression for $\nabla \grave{J}(\theta)$. For the i^{th} coagent in the synchronous network, this gives us: $\frac{\partial \grave{J}(\theta)}{\partial \theta_i} = \mathbf{E}\Big[\sum_{t=0}^{\infty} \grave{\gamma}^t \grave{G}_t \frac{\partial \ln\left(\grave{\pi}_i\left((\grave{S}_t, \grave{U}_t^{\text{pre}}), \grave{U}_t, \theta_i\right)\right)}{\partial \theta_i}\Big| \theta\Big]$.

Consider $\partial \ln\left(\grave{\pi}_i((\grave{S}_t, \grave{U}_t^{\text{pre}}), \grave{U}_t, \theta_i)\right)/\partial \theta_i$, which we abbreviate as $\partial \grave{\pi}_i / \partial \theta_i$. When $\grave{U}_t^{\text{pre}}.e_i = 0$, we know that the action is $\grave{U}_t^i = \grave{S}_t.u_i = \grave{U}_{t-1}^i$ regardless of θ. Therefore, in these local states, $\partial \grave{\pi}_i / \partial \theta_i$ is zero. When $\grave{U}_t^{\text{pre}}.e_i = 1$, we see from the definition of $\grave{\pi}$ that $\partial \grave{\pi}_i / \partial \theta_i {=} \partial \pi_i / \partial \theta_i$. Therefore, we see that in all cases, $\partial \grave{\pi}_i / \partial \theta_i {=} (\grave{U}_t^{\text{pre}}.e_i) \partial \pi / \partial \theta_i$. Substituting this into the above expression yields: $\mathbf{E}\Big[\sum_{t=0}^{\infty} (\grave{U}_t^{\text{pre}}.e_i) \grave{\gamma}^t \grave{G}_t \frac{\partial \ln\left(\pi_i\left((\grave{S}_t.s, \grave{U}_t^{\text{pre}}, \grave{S}_t.u^{\text{all}}), \grave{U}_t, \theta_i\right)\right)}{\partial \theta_i}\Big| \theta\Big]$.

In the proof that $J(\theta) = \grave{J}(\theta)$ given in Section C of the supplementary material, we show that the distribution over all analogous random variables is equivalent in both settings (for example, for all $s \in \mathcal{S}$, $\Pr(S_t = s) = \Pr(\grave{S}_t.s = s)$). Substituting each of the random variables of $\grave{M}$ into the above expression yields precisely the asynchronous local policy gradient, Δ_i. □

To empirically test the Asynchronous Coagent Policy Gradient Theorem (ACPGT), we compare the gradient (∇J) estimates of the ACPGT with a finite difference method. The results are presented in Figure 5 (Section D) of the supplementary material; this data provides empirical support for the ACPGT.

6. Applications to Past and Future Work

Consider past RL approaches that use stochastic networks, such as stochastic computation graphs, hierarchical networks like the option-critic, or any other form of SNN; one could use the ACPGT to immediately obtain a learning rule for any of these varieties of SNN. In other words, the theory in the sections above facilitates the derivation of gradient and learning rules for *arbitrary* stochastic architectures. The ACPGT applies when there are recurrent connections. It applies in the asynchronous case, even when the units (coagents) have different rates or execution probabilities, and/or have execution functions that depend on the output of other units or the state. It also applies to architectures that are designed for temporal abstraction, like the option-critic depicted in Figure 3. Architectures adding additional levels of abstraction, such as the *hierarchical option-critic* (Riemer et al., 2018), can be analyzed with similar ease. Architectures designed for other purposes, such as partitioning the state space between different parts of the network (Sutton et al., 2011), or simplifying a high-dimensional action space (Thomas & Barto, 2012), can also be analyzed with the coagent framework.

The diversity of applications is not limited to network topology variations. For example, one could design an asynchronous deep neural network where each unit is a separate coagent. Alternatively, one could design an asynchronous deep neural network where each coagent is itself a deep neural network. In both cases, the coagents could run at different rates to facilitate temporal abstraction. In the latter case, the gradient generated by an algorithm following the ACPGT could be used to train each coagent internally with backpropagation. Notice that the former architecture is trained without backpropagation while the latter architecture combines an ACPGT algorithm with backpropagation: The ACPGT facilitates easy design and analysis for both types of architectures and algorithms.

Deriving the policy gradient for a particular coagent simply requires identifying the inputs and outputs and plugging them into the ACPGT formula. In this way, an algorithm designer can rapidly design a principled policy gradient algorithm for any stochastic network, even in the asynchronous and recurrent setting. In the next section, we give an example of this process.

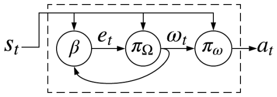

Figure 3. The option-critic framework, described by Bacon et al. (2017), depicted as a three-coagent network consisting of **1)** β, which contains the termination functions for each option, **2)** π_Ω, the policy over options, and **3)** π_ω, which contains the option policies and selects the action a_t. Each option has a corresponding *label*. Ω is the set of the available options' labels. The label corresponding to the option selected at time t is denoted as ω_t. β's local state is s_t and the previous option label, ω_{t-1} (the latter via a recurrent connection from π_Ω). β sends an action, $e_t \in \{0,1\}$, to π_Ω. π_Ω's execution function outputs 1 (a 100% chance of execution) if $e_t = 1$, and outputs 0 (no chance of execution) if $e_t = 0$. That is, π_Ω executes if and only if $e_t = 1$. π_Ω's local state is s_t and e_t. If it executes, π_Ω outputs a new option label $\omega_t \in \Omega$. π_ω chooses an action, $a_t \in \mathcal{A}$, based on its local state, s_t and ω_t. Internally, π_ω looks up the option policy corresponding to the option label ω_t and selects an action based on that option.

7. Case Study: Option-Critic

The coagent framework allows one to design an arbitrary hierarchical architecture, and then immediately produce a learning rule for that architecture. In this section, we study the well-known *option-critic* framework (Bacon et al., 2017) as an example, demonstrating how the CPGT drastically simplifies the gradient derivation process. The *option-critic* framework aspires to many of the same goals as coagent networks: namely, hierarchical learning and temporal abstraction. We show that the architecture is equivalently described in terms of a simple, three-node coagent network, depicted and described in Figure 3. More formal and complete definitions are provided in Section E.1 of the supplemental material, but Figure 3 is sufficient for the understanding of this section.

Bacon et al. (2017) give gradients for two of the three coagents. π_ω, the policy that selects the action, and β, the termination functions, are parameterized by weights θ and ϑ, respectively. Bacon et al. (2017) gave the corresponding policy gradients, which we rewrite as:

$$\frac{\partial J}{\partial \theta} = \sum_{x \in (\mathcal{S}\times\Omega)} d_\Omega^\pi(x) \sum_{a \in \mathcal{A}} \frac{\partial \pi_\omega(x,a)}{\partial \theta} Q_U(x,a), \quad (1)$$

$$\frac{\partial J}{\partial \vartheta} = -\sum_{x \in (\mathcal{S}\times\Omega)} d_\Omega^\pi(x) \frac{\partial \beta(x,0)}{\partial \vartheta} A_\Omega(x.s, x.\omega), \quad (2)$$

where $d_\Omega^\pi(x)$ is a discounted weighting of state-option pairs, given by $d_\Omega^\pi(x) := \sum_{t=0}^\infty \gamma^t \Pr(s_t{=}x.s, \omega_t{=}x.\omega)$, $Q_U(x,a)$ is the expected return from choosing option $x.\omega$ and action a at state s under the current policy, and $A_\Omega(s,\omega)$ is the advantage of choosing option ω, given by $A_\Omega(s,\omega) = Q_\Omega(s,\omega) - V_\Omega(s)$, where $Q_\Omega(s,\omega)$ is the expected return from choosing option ω in state s, and $V_\Omega(s)$ is the expected return from beginning in state s with no option selected.

Previously, the CPGT was written in terms of expected values. An equivalent expression of the local gradient for policy π_i is the sum over the local state set, $\mathcal{X}_i$, and the local action set, $\mathcal{U}_i$: $\partial J(\theta)/\partial\theta_i = \sum_{x\in\mathcal{X}_i} d_i^\pi(x) \sum_{u\in\mathcal{U}_i} \frac{\partial \pi_i(x,u)}{\partial \theta_i} Q_i(x,u)$, where $Q_i(x,u) = \mathbf{E}[G_t | X_t^i{=}x, U_t^i{=}u]$. Deriving the policy gradient for a particular coagent simply requires identifying the inputs and outputs and plugging them into this formula.

First consider the policy gradient for π_ω, that is, $\partial J/\partial\theta$: the input set is $X_\omega = \mathcal{S}\times\Omega$, and the action set is $\mathcal{A}$. The local initial state distribution (the d_i^π term) is given exactly by d_Ω^π, and the local state-action value function (the Q_i term) is given exactly by Q_U. Directly substituting these terms into the CPGT immediately yields (1). Note that this derivation is completely trivial using the CPGT: only direct substitution is required. In contrast, the original derivation from Bacon et al. (2017) required a degree of complexity and several steps.

Next consider $\partial J/\partial\vartheta$. The input set is again $X_\beta = \mathcal{S}\times\Omega$, but the action set is $\{0,1\}$. The local state distribution is again the distribution over state-option pairs, given by d_Ω^π. The PCGN expression gives us

$$\frac{\partial J}{\partial \vartheta} = \sum_{x \in (\mathcal{S}\times\Omega)} d_\Omega^\pi(x) \sum_{u \in \{0,1\}} \frac{\partial \beta(x,u)}{\partial \vartheta} Q_\beta(x,u).$$

In Section E.2 of the supplementary material we show that this is equivalent to (2). Notice that, unlike π_Ω, both of these coagents execute every atomic time step. The execution function, therefore, always produces 1 and therefore can be ignored for the purposes of the gradient calculation.

Finally, consider the gradient of J with respect to the parameters of π_Ω. Bacon et al. (2017) do not provide a policy gradient for π_Ω, but suggest policy gradient methods at the SMDP level, planning, or *intra-option Q-learning* (Sutton et al., 1999). One option is to use the ACPGT approach. The execution function is simply the output of the termination coagent at time t, e_t. We use the expected value form for simplicity, substituting terms in as above:

$$\partial J/\partial \mu = \mathbf{E}\Big[\sum_{t=0}^{\infty} e_t \gamma^t G_t \frac{\partial \ln\big(\pi_\Omega(X_t, \omega_t, \mu)\big)}{\partial \mu} \Big| \{\theta, \vartheta, \mu\}\Big],$$

where μ represents the parameters of π_Ω and $X_t \in (\mathcal{S}\times\{0,1\})$ is the local state of the π_Ω coagent at time t. While Bacon et al. (2017) introduce an asynchronous framework, the theoretical tools they use for the synchronous components do not provide a gradient for the asynchronous component. Instead, their suggestions rely on a piecemeal approach. While this approach is reasonable, it invokes ideas beyond the scope of their work for training the asynchronous component. Our approach has the benefit of providing a unified approach to training all components of the network.

The ACPGT provided a simplified and unified approach to these gradient derivations. Using the option-critic framework as an example, we have shown that the ACPGT is a useful tool for analyzing arbitrary stochastic networks. Notice that a more complex architecture containing many levels of hierarchy could be analyzed with similar ease.

8. Conclusion

We provide a formal and general proof of the coagent policy gradient theorem (CPGT) for stochastic policy networks, and extend it to the asynchronous and recurrent setting. This result demonstrates that, if coagents apply standard policy gradient algorithms from the perspective of their inputs and outputs, then the entire network will follow the policy gradient, even in asynchronous or recurrent settings. We empirically support the CPGT, and use the option-critic framework as an example to show how our approach facilitates and simplifies gradient derivation for arbitrary stochastic networks. Future work will focus on the potential for massive parallelization of asynchronous coagent networks, and on the potential for many levels of implicit temporal abstraction through varying coagent execution rates.

Acknowledgements

Research reported in this paper was sponsored in part by the CCDC Army Research Laboratory under Cooperative Agreement W911NF-17-2-0196 (ARL IoBT CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

We would like to thank Scott Jordan for help and feedback with the editing process. We would also like to thank the reviewers and meta-reviewers for their feedback, which helped us improve this work.

References

Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Barto, A. G. Learning by statistical cooperation of self-interested neuron-like computing elements. *Human Neurobiology*, 4(4):229–256, 1985.

Bertsekas, D. P. and Tsitsiklis, J. N. Gradient convergence in gradient methods with errors. *SIAM Journal on Optimization*, 10(3):627–642, 2000.

Buşoniu, L., Babuška, R., and De Schutter, B. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pp. 183–221. Springer, 2010.

Guestrin, C., Lagoudakis, M., and Parr, R. Coordinated reinforcement learning. In *ICML*, volume 2, pp. 227–234, 2002.

Klopf, A. H. *The hedonistic neuron: A theory of memory, learning, and intelligence*. Toxicology-Sci, 1982.

Liu, B., Singh, S., Lewis, R. L., and Qin, S. Optimal rewards for cooperative agents. *IEEE Transactions on Autonomous Mental Development*, 6(4):286–297, 2014.

Narendra, K. S. and Thathachar, M. A. *Learning Automata: an Introduction*. Prentice-Hall, Inc., 1989.

Riemer, M., Liu, M., and Tesauro, G. Learning abstract options. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R. (eds.), *NeurIPS 31*, pp. 10424–10434. Curran Associates, Inc., 2018. URL http://papers.nips.cc/paper/8243-learning-abstract-options.pdf.

Russell, S. J. and Norvig, P. *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited,, 2016.

Schulman, J., Heess, N., Weber, T., and Abbeel, P. Gradient estimation using stochastic computation graphs. In *NeurIPS*, pp. 3528–3536, 2015.

Sutton, R. S., Precup, D., and Singh, S. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2): 181–211, 1999.

Sutton, R. S., Modayil, J., Delp, M., Degris, T., Pilarski, P. M., and White, A. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pp. 761–768, 2011.

Thomas, P. S. Policy gradient coagent networks. In *NeurIPS*, pp. 1944–1952, 2011.

Thomas, P. S. and Barto, A. G. Conjugate markov decision processes. In *ICML*, pp. 137–144, 2011.

Thomas, P. S. and Barto, A. G. Motor primitive discovery. In *Procedings of the IEEE Conference on Development and Learning and Epigenetic Robotics*, pp. 1–8, 2012.

Williams, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3–4):229–256, 1992.

Zhang, S. and Whiteson, S. Dac: The double actor-critic architecture for learning options. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *NeurIPS 32*, pp. 2010–2020. Curran Associates, Inc., 2019.

Zhang, X., Aberdeen, D., and Vishwanathan, S. Conditional random fields for multi-agent reinforcement learning. In *ICML*, pp. 1143–1150. ACM, 2007.