University of the Witwatersrand, Johannesburg
School of Computer Science and Applied Mathematics

Impact of Noise on Learned Value Functions at Depth in CoAgent Networks for Neural Network Credit Assignment

Christopher Fourie
358183

WITS UNIVERSITY

Supervised by:
Prof. Benjamin Rosman

A research report submitted to the Faculty of Science, University of Witwatersrand, in partial fulfilment of the requirements for the degree of Master of Science.

Abstract

CoAgent networks (CoANs), networks of reinforcement learning agents, have been shown to be a biologically plausible alternative to backpropagation for solving the neural network structural credit assignment problem [Gupta *et al.* 2021]. This is accomplished where many agents, each as a neuron in a stochastic neural network, use only their local policy gradient and a global reward. Noise is an important consideration in the learning dynamics of any stochastic neural network [Schoenholz *et al.* 2017]. We investigate the impact of noise on learnt value functions for baselines and Actor-Critic methods in CoANs at depth. We demonstrate that with additional layers, CoANs using REINFORCE, REINFORCE with a baseline or Actor-Critic methods perform significantly worse. However unbiased variance reduction methods are effective at alleviating this to a moderate extent. For CoANs of increasing depth and width using Actor-Critic methods we show that learned value functions are more sensitive to noise. We show as well that large bootstrapping bias impacts Actor-Critic CoAN methods significantly.

Acknowledgements

There have been many amazing people who have helped make this possible:

My academic supervisor, Dr. Benjamin Rosman (Benji), their unwavering belief in my ability has helped me believe in myself. Despite my best efforts at erratic twists and turns in my research direction, has maintained a kindhearted demeanour throughout. You provided me an opportunity to thrive in a space that was completely new to me.

My parents, Sandy and John. Without your long (long) standing support and love I would probably be a practising medical doctor. Thank you for everything you have done along the journey that has helped lead me to a place that I enjoy thoroughly. I have always appreciated you and the values you instilled throughout my life.

My brother Simon, you've have always been looking out for me and I have always been looking up to you, even when you are not literally in the sky. I hope to be able to reciprocate the many kindnesses you have shown me over the years.

My sister and other mother, Kina and Rebecca you have helped keep me sane throughout this journey.

Jade, my amazing unicorn of a life partner, your support on the front line sustained me through some of the toughest moments of juggling research and a day job.

Krupa, Shahil, Elan, Geraud, Devon, Steve your interest in myself and my research is heart warming. You are superb friends, thank you for all the time and consideration you afforded me to bounce ideas around, helping to clarify the mystical arts of RL.

Kale-ab, Neelan, Pete, Korsti, Iffy the comradery and mutual support through the coursework components of this journey were crucial toward becoming fluent in this area of science. It was a fun time!

Dean, I knew how to code but you taught me how to debug, the latter has served me far better.

George, I am very grateful for the space and time to get the research done, with the assurance that I would be able to return to work when I am ready to.

Tara, thank you for feeding me and reminding me to "just do your best sweet pea".

Contents

Chapter 1

Introduction

1.1 Motivation

A common theme in machine learning and artificial intelligence literature is that for very narrowly defined problems, existing methods achieve excellent performance, in some cases even better than that of human intelligence [Goodfellow *et al.* 2015]. However, unlike biological learners, for machine learning there is yet to be a method that can be applied generally to a wide variety of problems. It is not known what is missing. There are opinions and speculation throughout the field.

Towards artificial general intelligence (AGI) and understanding the mechanisms of our own human intelligence, there have been few recent innovations as exciting and flexible as CoAgent networks (CoAN). Thomas and Barto [2011] introduced the concept of a CoAN, which built on top of Conjugate Markov Decision Process (CoMDP). CoMDPs consist of an MDP being used to solve an MDP. Consider that a reinforcement learning (RL) agent can have policies represented by stochastic neural networks (SNN). If a SNN policy is tasked with solving an MDP, we can then describe a CoMDP for each neuron of the SNN to solve. Each neuron of the SNN that is solving a CoMDP is then referred to as a conjugate RL agent (CoAgent) and the SNN of CoAgents is referred to as a CoAgent network (CoAN) [Thomas 2011].

CoANs exhibit three characteristics that could be important for generalising learning.

The first characteristic is the ability of CoANs to describe potential solutions to problems at various scales, since they can be defined recursively. A CoAgent can be made up of a network of CoAgents. In other words, a CoAgent, given it is also an RL agent, can also have a policy represented by a SNN and thus be indefinitely composed of further CoAgents. This ability to approach problems at various scales is something that is observed in agents in complex biological systems, where optimisation occurs at the level of a gene, a cell, an organ, an organism and an ecosystem simultaneously.

The second characteristic is that CoANs are general enough to solve both supervised learning and RL problems, bridging the two spaces. Kostas *et al.* [2019] show that CoANs, can be used both to describe hierarchical RL as multi-agent RL (MARL) systems and are a biologically plausible, asynchronous alternative for backpropagation (BP) in neural networks (NN) when using policy gradient methods.

Finally, CoANs allow for biologically plausible implementation of distributed neural networks

that can learn complex and non-differentiable activation functions as well as update their weights independently, using only local policy gradient information and a global reward signal [Kostas *et al.* 2019].

Recently Gupta *et al.* [2021] provided an empirical study of CoANs, used directly as a neural network, to solve supervised learning classification and regression tasks, open sourcing their implementation code. Their investigation focused on practical learning dynamics of the network and a benchmark against backpropagation. They demonstrated that, under their experimental conditions, CoANs performed significantly worse than backpropogation.

They additionally demonstrated that the bias from actor critic methods resulted in poor performance. However the reason for poor performance remains an open question.

It is clear that despite the promising characteristics of CoANs, little is known about the learning dynamics of CoANs, and thus further research is necessary.

1.2 Objectives

This research moves towards a better understanding of the learning dynamics of CoANs, by answering the open question raised in Gupta *et al.* [2021]: Why is the bias introduced by a critic in an Actor-Critic CoAN detrimental to learning? To answer this we:

1. Show that biased value functions are sensitive to the increase in noise in CoANs as depth and width increases. We vary depths and breadths of CoANs to understand how the network is affected by the increase in noise.
2. Show the impact of bootstrapping bias on performance.

More specifically, we compare performance of two popular variance reduction methods (namely baselines and temporal difference actor-critic [Weber *et al.* 2019]), to the control method (namely, REINFORCE). Experiments were run on two popular machine learning benchmarks (MNIST and FashionMNIST), for which the CoANs must perform credit assignment in a neural network to perform classification, similarly to the setup provided by Gupta *et al.* [2021].

1.3 Outline

In the following research report, we provide background to CoAgent Networks in Chapter 2, building up relevant concepts in RL and policy gradient methods in Section 2.1, and Section 2.2. We follow with a primer on CoANs in Section 2.3. We then discuss our research methodology in Chapter 3, describing our problem setup in Section 3.4. We introduce our central hypotheses with an overview of our experiments in Section 3.6 and then proceed in depth with each experiment, its results and discussion in Chapter 4, concluding in Chapter 5.

Chapter 2

Background and Related Work

In the following sections we build up concepts used generally by reinforcement learning (RL) (in Section 2.1 and Section 2.2). To bridge the gap between neural networks and MARL, we provide a neural network formulation of multi-agent reinforcement learning (MARL) with CoAgent networks in Section 2.4, and Conjugate Markov Decision Processes (CoMDPs) in Section 2.3.

2.1 RL and MDPs

RL is a solution space for *learning agents*, something capable of making sequential decisions, that must learn behaviour (a policy) through trial-and-error interactions with an environment [Kaelbling *et al.* 1996]. This is formalised using optimal control of incompletely-known Markov decision processes (MDPs) with delayed rewards (reinforcement), a concept adapted from dynamical systems theory [Sutton and Barto 2018a]. MDPs are a classical formalisation of sequential decision making problems. Here the most important aspects of the problem are described by a state that an agent can sense to some extent, an action that the agent can take that will affect that state and a goal or goals relating to the state of that environment.

When describing the RL problem with an MDP where agents move from state to state. These movements, called transitions, are assumed to have the *Markov Property*, where all information of past agent-environment interactions are contained in the current state. Said another way, given the present, the future does not depend on the past [Kaelbling *et al.* 1996].

While RL can benefit from incorporating both supervised and unsupervised learning, it is distinct from both. Supervised learning attempts to extrapolate or generalise its responses so that it acts correctly in situations not contained in a labelled training set, which alone is not adequate for learning from interactions. Unsupervised learning attempts to find hidden structure in unlabelled data. RL instead tries to maximise a reward signal present in the environment, accessed via its interactions with that environment.

2.1.1 MDPs

Markov decision processes are intended to include just three aspects, sensation, action and goal.

MDPs describe the environment that an agent interacts with. This description is expressed as

the tuple, $M = (\mathcal{S}, \mathcal{A}, \mathcal{R}, P, R, d_0, \gamma)$ [Sutton and Barto 2018b], where:

$\mathcal{S}$ is the set of all possible *states*
$\mathcal{A}$ is the set of all possible *actions*
$\mathcal{R}$ is the set of all possible *rewards*
Let $t \in \{0, 1, 2, \ldots\}$ 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 is the *transition function*

$$\begin{aligned} &P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1] \\ &\text{given by } P(s, a, s') := \Pr(S_{t+1} = s' \mid S_t = s, A_t = a) \end{aligned} \tag{2.1}$$

R is the *reward distribution*

$$\begin{aligned} &R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \times \mathcal{R} \to [0, 1] \\ &\text{given by } R(s, a, s', r) := \Pr(R_t = r \mid S_t = s, A_t = a, S_{t+1} = s') \end{aligned} \tag{2.2}$$

d_0 is the *initial state distribution*,

γ is the discount factor, $\gamma \in [0, 1]$

Any method well suited to solving such problems can be considered a RL method [Sutton and Barto 2018a].

2.1.2 Other elements of RL

Challenges exist in RL such as the trade-off between exploration and exploitation, that do not in other types of learning. This is demonstrated by a typical *Epsilon-Greedy* policy method, where a policy, π, is a mapping from an agent's state, s_t in the environment at a particular time, to an action, a_t, by the agent that modifies that state. Policies can be deterministic or stochastic mappings from an agent's state to action. Epsilon is then a tune-able variable describing the percent of purely stochastic actions an agent takes to explore an environment. Epsilon and therefore exploration, typically decreases over time as an agent learns more about its environment.

Using a stochastic method, a policy π returns a probability of an action being selected when in a specific state during a specific time step

$$\pi : \mathcal{S} \times \mathcal{A} \to [0, 1], \text{ such that} \tag{2.3}$$

$$\pi(s, a) := \Pr(A_t = a \mid S_t = s) \tag{2.4}$$

The goal of reinforcement learning is to find the optimal policy for an agent to maximise an *objective function*, such that the average rewards are maximised.

$$J(\pi) := \mathbf{E}\left[\sum_{t=0}^{\infty} \gamma^t R_t \mid \pi\right] \tag{2.5}$$

Knowing the estimated value of a state is necessary for long term optimisation. That is, optimisation over many sequential decisions. A reward or reward signal is immediate and primary, provided directly by the environment, while values, as estimated predictions of rewards are secondary [Sutton and Barto 2018a]. Values are described as the estimated average rewards for that state when using some policy

$$V^{\pi}(s) := \mathbf{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s, \pi\right] \tag{2.6}$$

Similarly, *the action-value function for policy* π is defined as

$$Q^{\pi}(s,a) := \mathbf{E}\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k} \mid S_t = s, A_t = a\right] \tag{2.7}$$

where q is the value or *quality* of taking action a in state s under a policy π

RL methods can be considered on episodic and continuing / non-terminating tasks. Here our focus is on continuing tasks however many RL expressions have been constructed to accommodate both.

To express the infinite accumulative discounted reward that is used to estimate the value of a state for a continuing task, *recursion* is used to define a *return* G as

$$\begin{aligned} G_t &:= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + \cdots \\ &= R_{t+1} + \gamma \underbrace{\left(R_{t+2} + \gamma R_{t+3} + \gamma^2 R_{t+4} + \cdots\right)}_{\text{return of next state}} \\ &= R_{t+1} + \gamma G_{t+1} \end{aligned} \tag{2.8}$$

The value functions V^{π} and Q^{π}, can be estimated from experience. For example, if an agent follows policy π and stores an average of the sampled returns that have followed that state, the average will converge to that state value V^{π}, as the number of times that state is visited tends to infinity.

$$\begin{aligned} V^{\pi}(s) &:= E_{\pi}\left[G_t \mid S_t = s\right] \\ &= E_{\pi}\left[R_{t+1} + \gamma G_{t+1} \mid S_t = s\right] \\ &= \sum_a \pi(a \mid s) \sum_{s'} \sum_r p\left(s', r \mid s, a\right) \left[r + \gamma \underbrace{E_{\pi}\left[G_{t+1} \mid S_{t+1} = s'\right]}_{value\ of\ next\ state}\right] \\ &= \underbrace{\sum_a \pi(a \mid s) \sum_{s',r} p\left(s', r \mid s, a\right)}_{probability} \underbrace{\left[\overbrace{r}^{sampled\ reward} + \gamma V^{\pi}\left(s'\right)\right]}_{return} \end{aligned}$$

If an action is considered under its optimal policy $*$, then the corresponding optimal value q^* under the optimal policy $*$ is

$$Q^* := \sum_{s',r} p\left(s', r \mid s, a\right) \left[r + \gamma \max_{a'} q^*\left(s', a'\right)\right] \tag{2.9}$$

It is this optimal value that we try and solve for in value based RL methods.

2.2 Policy Gradient (PG) Algorithms

Policy gradient (PG) methods [Williams 1992; Sutton *et al.* 2000; Sutton and Barto 2018a] estimate the gradient of an agent's expected returns with respect to the parameters of its policy. They represent a completely different class of solutions to traditional value based methods seen in the previous Section 2.1.2. So where value based methods use value and policy iteration [Sutton and Barto 2018b] to find a policy that produces an optimal value, policy based methods are able to directly follow a policy gradient toward an optimal policy. Policy gradient methods are useful in the context of our research as mechanism by which policy gradient CoAgent networks find solutions [Thomas 2011].

We can use a parameterised function approximator, for instance a neural network to describe the policy of an agent, where the network weights are represented by the parameter θ and the *parameterised policy* is represented by π^θ.

$\pi^\theta : \mathcal{S} \times \mathcal{A} \times R^n \to [0, 1]$, such that for all $\theta \in R^n, \pi^\theta(\cdot, \cdot)$ is a policy.

For PG methods it is assumed that the policies are differentiable, so that $\partial\pi(s, a, \theta)/\partial\theta$ exists for all $s \in \mathcal{S}, a \in \mathcal{A}, \theta \in R^n$

We use the subscript and superscript θ to show that an element is sampled using the parameterised policy π^θ, which is itself conditioned or dependant on the parameter vector θ. This can also be understood as an element being conditioned on a trajectory τ as sampled from an environment using the parameterised policy π^θ, i.e. θ implies $\tau \sim \pi^\theta$ when used in subscripts or superscripts. When considering expectation, this is used as a subscript, when used elsewhere, this is used as a superscript. There does then exist some redundancy in our notation, however this is intentional.

The formulation of the policy gradient theorem presented by Sutton *et al.* [2000] was given for two objectives: the average reward objective for the infinite horizon setting [Mahadevan 1996] and the discounted objective, J_γ, for the episodic setting.

Let us consider the episodic setting, as this is the setting used in our research here. In this setting, typically the objective used is the discounted objective

$$J(\theta) = E_\theta \left[\sum_{t=0}^{\infty} \gamma^t R_t \mid \theta\right] \tag{2.10}$$

Where the objective J is a function of the policy parameters θ. This can be expressed as the expectation for the sum of discounted rewards in an episode, also known as the discounted return G, given a set of policy parameters θ. However, in line with work from Gupta *et al.* [2021] we do not apply discounting. That is we set $\gamma = 1$. It is however important to note that errors in the PG formulation for cases where $\gamma < 1$ are common in highly cited literature [Nota and Thomas 2019]. We use then the undiscounted objective

$$J(\theta) = E_\theta \left[\sum_{t=0}^{\infty} R_t \mid \theta \right] \tag{2.11}$$

The undiscounted PG estimate can be expressed as

$$\nabla J(\theta) = E_\theta \left[\sum_{t=0}^{\infty} \psi^\theta \left(S_t, A_t\right) Q^\theta \left(S_t, A_t\right) \mid \theta \right] \tag{2.12}$$

Proofs for this equality exist in a variety of notational forms [Williams 1992; Sutton *et al.* 2000], with this particular formulation being derived from Nota and Thomas [2019].

Here $Q^\theta \left(S_t, A_t\right)$, is some *action-value function* under parameterised policy π^θ. However a *sampled return* G^θ, an *advantage function* $A^\theta(s,a)$ or a *temporal difference error* δ^θ can also be used. Advantage functions are discussed in section 2.2.2 and temporal difference methods are discussed in section 2.2.3.

$\psi^\theta \left(S_t, A_t\right)$ are the *compatible features* of a parameterised policy and represent how much θ may be changed in order for a specific action to be more likely in a specific state. These are defined as $\psi(s,a) := \frac{\partial}{\partial \theta} \ln \pi^\theta(s,a) = \nabla \ln \pi^\theta(s,a)$.

In the following subsections, we consider the impact of variance on PG methods as well as providing some background on advantage functions and temporal difference methods in the context of PG methods.

2.2.1 Variance of gradient estimates

PG methods provide an unbiased estimate of the gradient, however in practice exhibit high variance [Williams 2004; Sutton *et al.* 1999a; Baxter and Bartlett 2000]. For PG methods, a source of instability in behaviour is the variance of gradient estimates. Decrease in PG estimate variance can increase process stability [Zhao *et al.* 2012].

For PG methods, for which high variance of gradient estimates even in single agent environments is already problematic, increases exponentially with the addition of other agents. Lowe *et al.* [2017] show that the probability of taking a gradient step in the correct direction decreases exponentially with the number of agents N in an environment. They do this by providing a proof for

$$P(\langle \hat{\nabla} J, \nabla J \rangle > 0) = (0.5)^N \tag{2.13}$$

where $\hat{\nabla} J$ is the PG estimator from a single sample, and ∇J is the true gradient in a simple cooperative scenario with N agents and binary actions: $P\left(a_i = 1\right) = \theta_i$. There the reward is defined to be 1 if all actions are the same $a_1 = a_2 = \ldots = a_N$ and 0 otherwise.

This shows that methods to decrease variance of gradient estimates for MARL problems to be a good target when trying to stabilise learning.

2.2.2 Baselines and advantages

A popular method to decrease variance but to keep the gradient estimate unbiased is to use a generalisation of the PG theorem that includes a comparison of the action value to an arbitrary **baseline** b. Intuitively this baseline can be thought of as the action that is best to take on average. Or said differently, what is the advantage of the action I have just taken with respect to the average action. This notion is also captured in the concept of an advantage function as shown in Equation 2.18.

Making gradient updates only considering the distance from this average helps to decrease the magnitude of the gradient update. This means that the variance in the sizes of gradient updates is decreased. This helps to stabilise learning as the policy parameters change less with respect to one another for each update.

If consider the undiscounted case, where $\gamma = 1$, then baseline can be expressed as

$$\nabla E_\theta[R(\tau)] = E_\theta\left[\sum_{t=0}^{T-1} \nabla \ln \pi^\theta\left(A_t \mid S_t\right)\left(\sum_{t'=t}^{T-1} R_{t'} - b\right)\right] \tag{2.14}$$

where b is any function or random variable not dependant on action a [Sutton and Barto 2018a].

Often a baseline that is an learned estimate of the state value is used $b = v_w(s)$, where w is an arbitrary parameter vector. However one could even sample a baseline sample from a random variable.

Considering the case of the learned state value estimate baseline, it can be shown that, despite the inclusion of a baseline, the gradient estimates remain unbiased [Schulman *et al.* 2018; Takeshi 2017].

Given

$$\nabla E_\theta[R(\tau)] = E_\theta\left[\sum_{t=0}^{T-1} \nabla \ln \pi^\theta\left(A_t \mid S_t\right)\left(\sum_{t'=t}^{T-1} R_{t'} - b\left(S_t\right)\right)\right] \tag{2.15}$$

this can be equivalently represented as

$$\nabla E_\theta[R(\tau)] = E_\theta\left[\sum_{t=0}^{T-1} \nabla \ln \pi^\theta\left(A_t \mid S_t\right)\left(\sum_{t'=t}^{T-1} R_{t'}\right) - \sum_{t=0}^{T-1} \nabla \ln \pi^\theta\left(A_t \mid S_t\right) b\left(S_t\right)\right] \tag{2.16}$$

then just considering the term containing the baseline

$$\begin{aligned} E_\theta\left[\nabla \ln \pi^\theta\left(A_t \mid S_t\right) b\left(S_t\right)\right] &= E_{s_{0,t}, a_{0:t-1}}\left[E_{s_{t+1:T}, a_{t:T-1}}\left[\nabla \ln \pi^\theta\left(A_t \mid S_t\right) b\left(S_t\right)\right]\right] \\ &= E_{s_{0,t}, a_{0,t-1}}\left[b\left(S_t\right) \cdot E_{s_{t+1:T}, a_{t-T-1}}\left[\nabla \ln \pi^\theta\left(A_t \mid S_t\right)\right]\right] \\ &= E_{s_{0:t}, a_{0:t-1}}\left[b\left(S_t\right) \cdot E_{A_t}\left[\nabla \ln \pi^\theta\left(A_t \mid S_t\right)\right]\right] \\ &= E_{s_{0:t}, a_{0:t-1}}\left[b\left(S_t\right) \cdot 0\right] = 0 \end{aligned} \tag{2.17}$$

We can then interpret the difference of the expected action-value estimate and a baseline as the advantage A of taking an action as compared to a baseline value. This gives us

$$A^{\theta}(s,a) = Q^{\theta}(s,a) - V^{\theta}(s) \tag{2.18}$$

We use Equation 2.18 to formulate what is known as *advantage* Actor-Critic that we use in Experiments 5 and Experiment 3. We deal with Actor-Critic methods in the next Section 2.2.3

2.2.3 Temporal Difference and Actor-Critic Methods

Temporal difference (TD) learning provides a means for state value estimate updates in the continuing learning case. This contrasts episodic updates from methods such as Monte Carlo (MC) learning [Sutton and Barto 2018a, Chapter 6]. This is achieved by bootstrapping off immediate estimates of the return instead of waiting until the end of an episode to sample the return.

The policy update for temporal difference learning is expressed as:

$$q^{\text{new}}(s,a) \leftarrow \underbrace{q^{old}(s,a)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Bigg(\underbrace{\underbrace{r}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_{a'} q^{old}(s',a')}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{q^{old}(s,a)}_{\text{old value}} \Bigg)}^{\text{temporal difference}} \tag{2.19}$$

Here one can see bootstrapping as updating a value estimate using a previous estimate. Understandably when taking an estimate of an estimate, one will accumulate estimation error. This is referred to as bootstrapping bias [Sutton and Barto 2018b].

If the bootstrapping bias introduced is not too high, then this method can be used to increase performance via variance reduction. Variance arises due to the accumulated stochasticity and noise in a trajectory. Variance is reduced when an update occurs over a step or number of steps that are less than the total steps in an episode. For example a TD(0) method that updates using 1 step trajectories, will accumulate less randomness and noise in a single transition than a MC method that uses an update from an episode length trajectory.

Actor-critic methods combine the state value estimation (the critic) with PG methods (the actor) to enable the use of TD methods for the case of continuing learning and thus a means for variance reduction. An overview of how these two parts interact is illustrated in Figure 2.1.

The Critic looks to solve the temporal credit assignment problem. The Critic can assign positive values to sensory states that are not associated with immediate reward but predict that reward will be obtained in the future.

After each action selection, the critic evaluates the new state to determine whether things have gone better or worse than expected. That evaluation is the TD error:

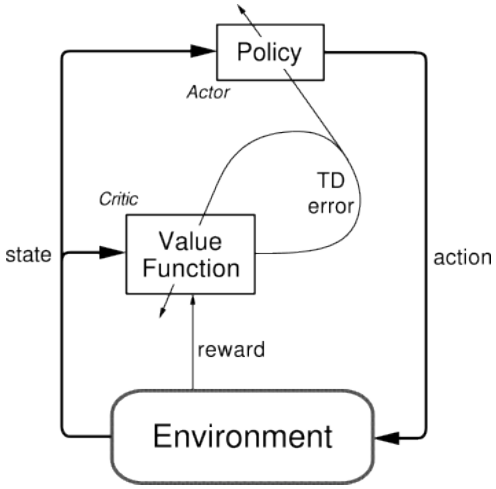


Figure 2.1: Actor-critic overview [Sutton and Barto 2018a]

$$\nabla J(\theta) = E_\theta \left[\sum_{t=0}^{T-1} \nabla \ln \underbrace{\pi^\theta \left(A_t \mid S_t \right)}_{actor} \underbrace{\delta_t^\theta}_{critic} \right] \tag{2.20}$$

where

$$\delta_t^\theta = R_{t+1}^\theta + \gamma V^\theta \left(S_{t+1} \right) - V^\theta \left(S_t \right) \tag{2.21}$$

Noting that for our research, we use the undiscounted formulation where $\gamma = 1$.

For MARL problems various centralised and decentralised actor-critic techniques have been developed that, for instance, might pre-train a centralised critic at scale, to then have each actor maintain its own critic once deployed. [Lowe *et al.* 2017; Iqbal and Sha 2019].

2.3 Conjugate MDPs and CoAgents

Broadly a Conjugate Markov Decision Process (CoMDP) can be considered as using an MDP to solve an MDP, although its interpretation is flexible [Thomas and Barto 2011].

Formally, a CoMDP is a direct optimisation of the expected return. Phrased in a general way: it is the search for a mapping, f, from the state set to some other set or space $\mathcal{U}$. $\mathcal{U}$ generalises some types of destinations for policy mappings of an agent. That is, for representation discovery, $\mathcal{U}$ would be the feature space, for motor primitive discovery $\mathcal{U}$ would be the action set of a MDP, for skill discovery it would be the termination probabilities. The goal is to find the mapping f^*, that maximises the agent's expected return. While an agent solves a MDP, a CoAgent, solves the search for f^*, the CoMDP. In this way it can be viewed as a form of meta learning.

The usefulness of CoMDPs arise as they are a general approach to search *any* mapping used by *any* algorithm attempting to solve an MDP, where we are interested in the maximum expected return. An example of this usefulness is the generalisation of the hierarchical RL option-critic architecture [Bacon *et al.* 2017] as seen in Kostas and Thomas (2019) Kostas *et al.* [2019] using

CoAgent networks and CoMDPs. Here this generality allows use to describe a hierarchical network of RL agents architected as a feedforward neural network.

Figure 2.2 depicts the recursive nature of a CoMDP with an agent comprised of sub-agents / CoAgents. Figure 2.3 is an example of a feedforward network using CoAgents as neurons, where the entire network is the agent.

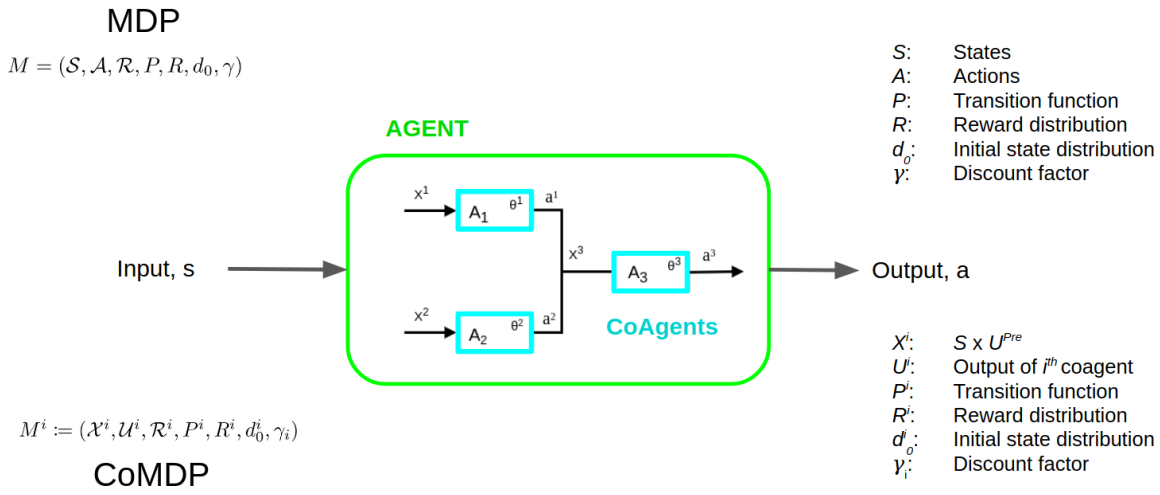


Figure 2.2: CoMDP compared to MDP

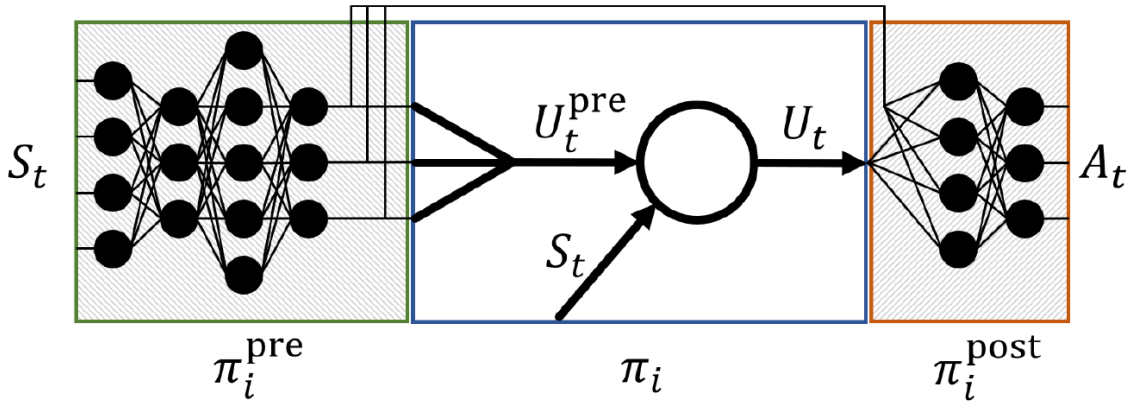


Figure 2.3: Example depicting feedforward network of CoAgents. Action generation is a three step process. The node in the middle denotes the i^{th} CoAgent. In the first step, nodes preceding the i^{th} CoAgent are executed to compute its inputs, U_t^{pre}. In the second step, the i^{th} CoAgent uses these inputs to produce its output U_t. In the third step the remainder of the network is executed to produce an action. [Kostas *et al.* 2019]

2.3.1 CoAgent PG Theorem

For a CoAgent network we can describe the parameters as

$$\theta = \left(\theta_i, \bar{\theta}_i\right) \tag{2.22}$$

where θ are the parameters that describe the policy of an agent solving the original MDP, θ_i are the parameters of the i^{th} CoAgent and $\bar{\theta}_i$ are the parameters of all other CoAgents, excluding the i^{th} CoAgent.

Where each CoAgent is cooperating to maximise the team reward, [Kostas *et al.* 2019, Property 11] assert a property of the CoAgent Policy Gradient Theorem that an individual CoAgent's objective function is equivalent to the team objective function, such that $J(\theta) = J_i(\theta_i)$

The theorem [Kostas *et al.* 2019, Theorem 1], then states that the gradient of the team objective function $\nabla J(\theta)$ with respect to θ, is a composition of CoAgent's policy gradients such that:

$$\nabla J(\theta) = \left[\Delta_1(\theta_1)^\top, \Delta_2(\theta_2)^\top, \ldots, \Delta_m(\theta_m)^\top\right]^\top \tag{2.23}$$

where m is the number of coagents and Δ_i is the local policy gradient of the i $^{\text{th}}$coagent.

The proof is below and follows from other properties of CoAgents provided by Kostas *et al.* [2019]

$$\begin{aligned}
\nabla J(\theta) &= \left[\frac{\partial J(\theta)^\top}{\partial \theta_1}, \frac{\partial J(\theta)^\top}{\partial \theta_2}, \ldots, \frac{\partial J(\theta)^\top}{\partial \theta_m}\right]^\top \\
&= \left[\frac{\partial J_1(\theta_1)^\top}{\partial \theta_1}, \frac{\partial J_2(\theta_2)^\top}{\partial \theta_2}, \ldots, \frac{\partial J_m(\theta_m)^\top}{\partial \theta_m}\right]^\top \\
&= \left[\frac{\Delta_1(\theta_1)^\top}{\partial \theta_1}, \frac{\Delta_2(\theta_2)^\top}{\partial \theta_2}, \ldots, \frac{\Delta_m(\theta_m)^\top}{\partial \theta_m}\right]^\top
\end{aligned} \tag{2.24}$$

We can then express the local policy gradient updated for the i^{th} CoAgent as

$$\Delta_i(\theta_i) := \mathbf{E}_\theta\left[\sum_{t=0}^{\infty} \gamma^t G_t \frac{\partial \ln(\pi_i(X_t, U_t, \theta_i))}{\partial \theta_i} \mid \theta\right] \tag{2.25}$$

$\Delta_i(\theta_i)$ is equivalent to the policy gradient of the i^{th} CoMDP, making the update rule:

$$\theta_i \leftarrow \theta_i + \alpha \sum_{t=0}^{\infty} \gamma^t G_t \left(\frac{\partial \ln(\pi_i(X_t, U_t, \theta_i))}{\partial \theta_i}\right) \tag{2.26}$$

One of the interpretations of CoMDPs being used to form CoAgent networks is as a multi-agent reinforcement learning system, which is dealt with in the next Section 2.4.

2.4 Multi-Agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) is where a team of agents interacts with an environment and each other to learn distributed policies for sequential decision making. MARL has already been shown to be applicable to a variety of domains, including robotics, distributed

control, telecommunications, resource management, collaborative decision support systems, and economics [Buşoniu *et al.* 2010]. Its usefulness is however limited due to problems that arise when scaling the number of agents [Bernstein *et al.* 2002; Zhang *et al.* 2019b]. This ability to scale is crucial for many problems in the multi-agent setting where agents are numerous [Canese *et al.* 2021b].

When trying to scale any MARL system, one can consider a solution to lie on either side of a spectrum.

On one side of the spectrum, suppose we try model the entire system as the *joint action space* of a single RL agent attempting to represent the actions of many or all agents in an environment. While this may be feasible for a very small joint action space, it fails to scale, due to the curse of dimensionality. In MARL systems the dimension of the joint action space increases exponentially with the number of agents [Hernandez-Leal *et al.* 2017]. Each additional agent's action space is added to the existing joint-action space combinatorially.

One the other side of the spectrum, is the case of *independent learners*, where each RL agent considers every other agent as part of the environment. From the perspective of an individual agent, concurrent learning of RL agents makes the environment appear to be *non-stationary* or *unstable*, which in turn causes learning to be unstable. This is one of the most well-known issues in MARL [Buşoniu *et al.* 2010; Tuyls and Weiss 2012; Hernandez-Leal *et al.* 2017]. The actions and subsequent policy updates of one agent affects the rewards of other agents and the evolution of the state. This results in each agent having to account for how every other agent behaves and adapts to the joint behaviour. For single agent RL, convergence guarantees have been developed with the assumption that environments are static and Markovian. In MARL, since any single agent's environment is dynamic and non-stationary, standard single agent algorithms are not able to accommodate this. It has been shown that ignoring the existence of other agent's behaviours may cause a failure to converge [Tan 1997; Claus and Boutilier 1998].

In our CoAN each neuron is an independent learner, using only local observations and a global reward signal to update its policy. Scaling this MARL system can be considered a problem of coordination in distributed control.

A popular example of this problem [Hu *et al.* 2021; Zhang *et al.* 2019a] is to try coordinate the movement and interaction of motor vehicles at a traffic intersection. We provide a toy gridworld example in Figure 2.4, where all agents receive a global positive reward if any of them get to their destination or goal quickly and a global negative reward for collisions. While this environment does not directly relate to our problem, we believe this example serves to help orient our intuition for agents that need to learn in the presence of other agents.

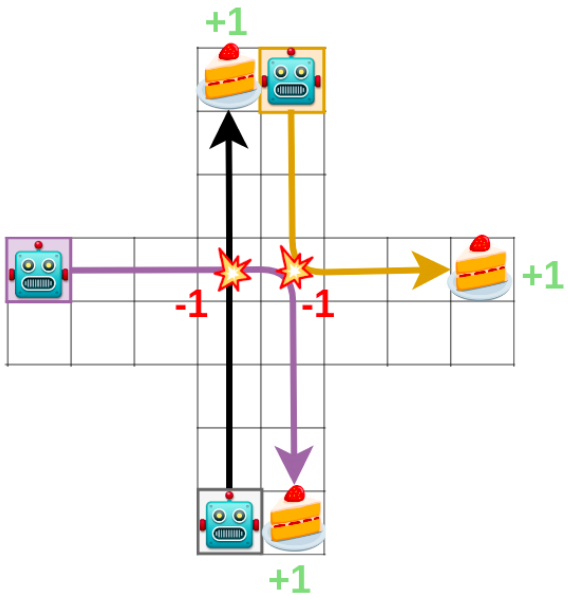


Figure 2.4: Coordinating a grid world traffic intersection as a distributed control problem.

This is serves to elucidate a problem at the heart of our cooperative MARL problem. With an increase in the number of agents, so too does the potential for unwanted collisions increase.

More generally, a collision is any combination of actions that when occurring together, drive down the global reward. In the context of our CoAN where agents have stochastic policies, the more stochastic the policy, the less predictable the RL agent, the higher the chance for uncoordinated actions. Toward a solution for this problem we could perhaps assign blame to agents that are responsible for collisions. A naïve approach may be to decrease the stochasticity of all agents to decrease the number of collisions. However, with low stochasticity exploration is hindered and learning can slow down immensely or get stuck indefinitely in local optima. Effectively deciding who is to blame for failure and who is to credit for successes when there is only a global reward available is at the heart of the cooperative MARL credit assignment problem. That is, MARL credit assignment is described as deducing the individual contribution of an agent to the team's success [Nguyen *et al.* 2018]. Quantifying the credit that is assigned to an agent and how that can be used to increase team success is non-trivial and can depend on the problem description.

Credit assignment in a neural network (NN) context, for instance is described as the contribution of each synapse to the network's overall performance [Lillicrap and Santoro 2019]. Backpropagation of error signals is the canonical method for assigning credit (or blame) in a NN. Derivatives of the error between the target output of the NN and the actual output are used to adjust the synaptic weights between neurons. Data that has been labelled is used as the target output of the NN, a foundation of supervised learning.

In our context we use CoANs to address the neural network credit assignment using a global reward signal generated, given the difference between the labelled data and our CoANs output action. The use of global reward signal the CoAgents are able to follow a policy gradient and progressively assign credit to weights in a network without the need for backpropagation throughout the entire network.

2.5 Related work

Work on CoMDPs and CoAgent networks was initiated by Thomas and Barto [2011], providing the theoretical principles and interpretations for the space, that include the interpretation of CoAgent networks as a cooperative MARL system. They present a novel approach to accomplish both representation and skill or option discovery [Sutton *et al.* 1999b] using reinforcement learning. Additionally, they describe how group coordinate ascent of CoAgents using policy gradient methods [Williams 2004] can solve problems together. They provided an example of the first CoAN that is shown to successfully to solve a high dimensional navigation task.

This work was extended by Thomas [2011], introducing novel Actor-Critic methods for CoANs and emphasises the biological realism of this approach. They empirically showed the effects of various interventions to reduce the variance of CoANs, highlighting the importance of variance consideration in this space. Following this Thomas and Barto [2012] continue work on motor primitive discovery.

Kostas *et al.* [2019] builds on the previous works, with a strong emphasis on providing more extensive theoretical proofs for CoANs using policy gradient methods. They also provide an example an option-critic [Bacon *et al.* 2017] as a CoAN to show how CoANs can be used to generally describe hierarchical reinforcement learning setups as MARL systems.

Recently Gupta *et al.* [2021] provide an empirical study of CoANs used directly as a neural network to solve supervised learning classification and regression tasks, open sourcing their implementation code[1]. Investigations focus on practical learning dynamics of the network and a benchmark against backpropagation.

Our research extends this work with a focus toward understanding how value functions used by variance reduction methods are affected by the noisy signals propagated within CoANs. We apply this to use of variance reduction for deeper CoANs.

CoANs are stochastic neural networks (SNNs), where the policies of each neurons can themselves be described as stochastic neural networks. Being able to generally describe and link deterministic, stochastic neural networks and reinforcement learning is the focus of Schulman *et al.* [2016]; Weber *et al.* [2019]. Their framework is general enough to include CoANs and is a useful tool for formalising ideas of baselines and actor-critics for CoANs.

Toward a better understanding of the learning dynamics for SNNs, Merkh and Montúfar [2019] provide proofs and discuss the theoretical bounds of SNN width and depth as each relate to the SNNs ability carry out universal function approximation. Specifically on the depth of information propagation in SNN, Schoenholz *et al.* [2017] discuss the limits of depth for deterministic NNs with respect to their initialisation. They showed that with the correct critical initialisation an arbitrarily deep network can be trained. However they also show that if a network is made stochastic using Dropout, then this property is lost and a theoretical limit exists. Pretorius *et al.* [2020] extend this with an investigation of critical initialisation's usefulness in the absence of an arbitrary depth training guarantee. They conclude that "it is likely to be a fruitless endeavour". These works helped orient our intuition and expectations for the possible depths of CoANs as SNNs.

SNN are used in the neuroscience space to model spiking neural networks. Aenugu *et al.* [2019] showed that CoANs can be applied similarly to spiking neural networks. Lastly Zini *et al.* [2020] provide a theoretical summary of CoANs with a focus on hierarchical reinforcement learning and

[1] https://openreview.net/attachment?id=nz2iUi-iZLQname=code

an empirical study that includes option-critic and tuning of various hyperparameters. Their code is openly available[2].

2.6 Summary

In this chapter we build up concepts around RL. First on MDPs to describe sequential decision making problems and how RL can be used to solve these types of problems. We continue within RL methods, specifically on policy gradient methods and how they suffer from high variance in practice. We show how this high variance can be mitigated using temporal difference learning and how temporal difference learning can be enabled in a policy gradient context through the introduction of a critic that uses value based methods.

We build up the concept of CoAgents starting with conjugate MDPs, that is, MDPs describing MDPs. We then provide a high level introduction to using policy gradient methods for CoAgents. We then provide an intuitive example for how CoAgent networks can be interpreted as MARL systems. Following this we discuss related work, with a roughly chronological framing.

In the following Chapter 3 we discuss our methodology. Here we define our research problem and hypotheses, as well as describing in detail our experimental setup.

[2] https://github.com/mojishoki/Coagent-Networks-Revisited

Chapter 3

Methodology

3.1 Problem Statement

Recently Gupta *et al.* [2021] provided an empirical study of CoANs used directly as a neural network to solve supervised learning classification and regression tasks. Their investigation focused on practical learning dynamics of the network and a benchmark against backpropagation. They demonstrated that, under their experimental conditions, CoANs performed significantly worse than backpropagation.

They additionally demonstrated that the bias from actor critic methods resulted in poor performance. However the reason for poor performance remains an open question.

It is clear that despite the promising characteristics of CoANs, little is known about the learning dynamics of CoANs, and thus further research is necessary.

3.2 Research Questions

This research moves towards a better understanding of the learning dynamics of CoANs, by answering the open question raised in Gupta *et al.* [2021]: Why is the bias introduced by a critic in an Actor-Critic CoAN extremely detrimental to learning?

To answer this question, we investigate two directions: (i) Sensitivity to Noise in Value Function Observations (ii) The Effect of Bootstrapping Bias.

3.2.1 Sensitivity to Noise in Value Function Observations

Schoenholz *et al.* [2017] suggest a theoretical maximum depth for stochastic neural networks. This theoretical maximum is hypothesised to exist due to the existence of noise which inhibits information propagation through the stochastic neural networks.

High variance of gradient estimates for each agent are a significant source of instability, as is for any MARL system using policy gradient updates [Canese *et al.* 2021b; Hernandez-Leal *et al.* 2017]. Thus, as any stochastic MARL system increases in size (such as a CoAN), the noise in the network increases.

Therefore it is important to examine the sources of noise within the network, as well as the impact of that noise on learning.

We investigate the sensitivity of biased value function observations to the increase in noise in CoANs as network depth and width increase. We vary depths and breadths of CoANs to understand how the network is effected by the increase in noise due to network size changes.

3.2.2 The Effect of Bootstrapping Bias

Bootstrapping is a notorious source of instability, being one of the three components of deadly triad [Sutton and Barto 2018b], the other two being function approximation and off-policy methods. As a secondary investigation, we look at the effect of bootstrapping bias on performance.

3.3 Hypotheses

Toward answering the questions above in Section 3.2, we formulate two hypotheses. It cannot be assumed that variance reduction will be beneficial in every CoAN configuration, and so we formulate our first hypothesis to test this as follows:

Hypothesis 1 *Variance reduction methods can help stabilise learning and increasing performance of CoANs at any depth and breadth.*

The causes of decreased performance for actor-critic methods in CoANs could be singular or multi-factorial. We choose noise as a singular possible component to test, because SNNs exhibit noise as discussed in Section 2.5. To include learned baselines, we generalise beyond Actor-Critic to any learned function approximator that could be used to reduce variance.

We formulate our hypothesis about noise in CoANs, as per Section 3.2.1, as follows:

Hypothesis 2 *Variance reduction methods using biased value estimates by a learned function approximator (value function) are more sensitive to additional noise in deeper CoANs than in shallower CoANs*

3.4 Problem Definition

Credit assignment in a feedforward neural network with backpropagation is carried out during optimisation of an objective function, with the error being mapped directly to neuron weight updates via recursively applying the chain rule [Goodfellow *et al.* 2015].

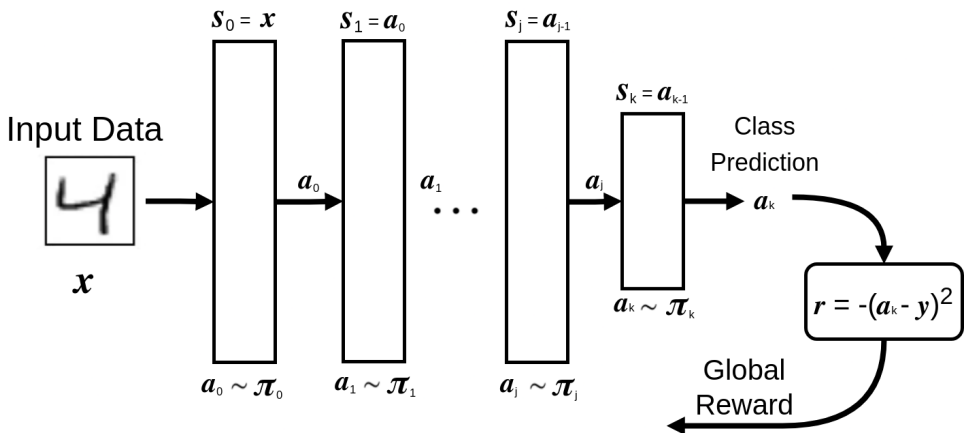
One of the main contributions from Gupta *et al.* [2021], was describing an equivalent MARL optimisation problem where error is communicated via a global reward signal, with credit assignment updates for neuronal weights occurring as each individual reinforcement learning agent neuron works toward optimising their own objective by following a policy gradient.

In the following we describe this problem definition from Gupta *et al.* [2021]. Fundamentally it is a Markov decision problem that has a finite time horizon or *Finite horizon MDP*. Said another way, it is a MDP that has a finite number of steps and is not ongoing. The problem exists at the

intersection of reinforcement learning (RL) and neural networks. In an effort to form a bridge between these two areas of research we try blend concepts and terminology from both. To start, we equate RL time steps and the layers of a neural network(NN). This is done by considering the input layer as time step zero and the first hidden layer as time step one. As the input data traverses the neural network from layer to layer, these are state transitions. One can see these state transitions across the NN depicted in figure 3.1.

To note, the problem was originally described generally for both the discrete and continuous case, however we choose to only consider the discrete action case as it has been shown to be more stable than the continuous action case.

More formally, given a fully connected feedforward neural network composed of k layers, with each layer containing n neurons, where each neuron's activation function is a probabilistic mapping from state to action described by a reinforcement learning agent's stochastic policy π. This policy is parameterised by the neuron's weight vector θ. We treat the input of the neuron as its state observation s. For an arbitrary neuron i in an arbitrary layer j, a binary discrete action $a_{i,j} \in [0, 1]$, is sampled from a Bernoulli distribution produced by neuronal weights. We go into further detail on the exact mechanism of it here 3.4 and depict it in figure 3.2.



(a)

Figure 3.1: Neural network forward pass as a finite horizon MDP adapted from Gupta *et al.* [2021]

We describe the sampling of the action $a_{i,j}$ from this distribution as

$$A_{i,j} \sim \pi(\cdot|S_{i,j}, \theta_{i,j}) \tag{3.1}$$

For the input layer of the CoAN $S_{j=0}$, there exist a vector input or or initial state for the CoAN X sampled from a supervised learning dataset (e.g. MNIST) where $X \in R^d$ and for each x, there is a labelled inference target y. We can express the action selection for the input layer of the CoAN as

$$A_{i,0} \sim \pi(\cdot|X, \theta_{i,0}) \tag{3.2}$$

For all downstream layers of the network that are not the input layer $j \neq 0$, a neuron's state observation are the actions of all the neurons from the previous layer $S_j = A_{j-1}$

$$A_{i,j} \sim \pi(\cdot|A_{j-1}, \theta_{i,j}) \tag{3.3}$$

Assuming a fully connected network, every neuron then in a layer j sees the same state observation s. For notional simplicity regarding equations 3.1, 3.2 and 3.3, we use the following to generally express action selection of a single CoAgent receiving an input vector S_j from the preceding layer

$$A_{i,j} \sim \pi^{\theta}(\cdot|S_j) \tag{3.4}$$

Each layer j in the network represents and is equivalent to a time-step t of a finite horizon MDP, where the finite time horizon or terminal time-step is equivalent to the terminal layer of the network k.

The only time step or layer to receive a reward is the terminal layer k as the negative of the prediction error from the action of this layer a_k and the labelled target y. We then let the return for any layer G be the undiscounted sum of the rewards to give us the following equations

$$\begin{aligned} R_k &:= -\operatorname{err}(A_k, Y) \\ G &= \sum_{j=0}^{k} R_j = R_k \\ G &= -\operatorname{err}(A_k, Y) \end{aligned} \tag{3.5}$$

Where $err(A_k, Y)$ is an arbitrary error function for a prediction A_k and labelled target Y. A transition between states is represented as a progression to downstream layers. Using a mean square error, we let a full episode trajectory τ take the form $s_0, a_0, r_0, s_1, a_1, r_1 ... s_k, a_k, r_k$ such that

$$\begin{aligned} j = 0: &\quad S_0 = x,\ A_0 = a_0, R_0 = 0 \\ j = 1: &\quad S_1 = a_0,\ A_1 = a_1,\ R_1 = 0 \\ &\quad ... \\ j = k: &\quad S_k = a_{k-1},\ A_k = a_k,\ R_k = -(a_k - y)^2 \end{aligned} \tag{3.6}$$

These transitions are Markov, because given a state s_1 and the corresponding action from that state a_1, the transition to the next state s_2 is independent of s_1.

The probability of a trajectory τ, is given by

$$p(\tau) = p(x)\pi^{\theta}(a_0|s_0)p(s_1|s_0, a_0)\pi^{\theta}(a_j|s_j)...p(s_k|s_{k-1,a_{k-1}})\pi^{\theta}(a_k|s_k)p(y|x) \tag{3.7}$$

Where $p(x)$ is the probability of the initial start state, $\pi^{\theta}(a_0, a_1, \ldots, a_k \mid x)$ is the probability of all actions taken from x and $p(y \mid x)$ describes the probability of reward on termination. However as our state transitions are deterministic, this can be simplified to

$$p(\tau) = p(x)\pi^{\theta}(a_0, a_1, \ldots, a_k \mid x)\, p(y \mid x) \tag{3.8}$$

Our expected return can then be expressed as

$$\begin{aligned} E[G] &= \int p(\tau)\ G\ d\tau \\ E[G] &= \int p(x)\pi^{\theta}\left(a_0, a_1, \ldots, a_k \mid x\right) p(y \mid x)\ G\ dx da_0 \ldots da_k dy \end{aligned} \tag{3.9}$$

We define an undiscounted objective function for our CoAgent network similarly as for a typical reinforcement learning agent with a parameterised policy using equation 2.11, however our expectation includes dependencies related to sampling our supervised training dataset

$$\begin{aligned} J(\boldsymbol{\theta}) &:= E[G \mid \theta, x, y] \\ J(\boldsymbol{\theta}) &= E_{\theta, p(x,y)}[G] \end{aligned} \tag{3.10}$$

so that by applying the policy gradient theorem in equation 2.12, the policy gradient of the CoAgent network for a layer j is

$$\nabla J(\boldsymbol{\theta_j}) = E_{\theta, p(x,y)}\left[Q^{\theta}\left(S_j, A_j\right) \nabla \ln \pi^{\theta}\left(A_j \mid S_j\right)\right] \tag{3.11}$$

Then by equations 2.23, 2.24, 2.25 we can describe the local policy gradient for the i^{th} CoAgent in the j^{th} layer as

$$\nabla J(\boldsymbol{\theta_{i,j}}) = E_{\theta, p(x,y)}\left[Q^{\theta}\left(S_{i,j}, A_{i,j}\right) \nabla \ln \pi^{\theta}\left(A_{i,j} \mid S_{i,j}\right)\right] \tag{3.12}$$

However as each CoAgent in a layer, receives the same input vector we can simplify the state component of this expression to

$$\nabla J(\boldsymbol{\theta_{i,j}}) = E_{\theta, p(x,y)}\left[Q^{\theta}\left(S_j, A_{i,j}\right) \nabla \ln \pi^{\theta}\left(A_{i,j} \mid S_j\right)\right] \tag{3.13}$$

Using stochastic gradient ascent

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla \widehat{J\left(\boldsymbol{\theta}_t\right)} \tag{3.14}$$

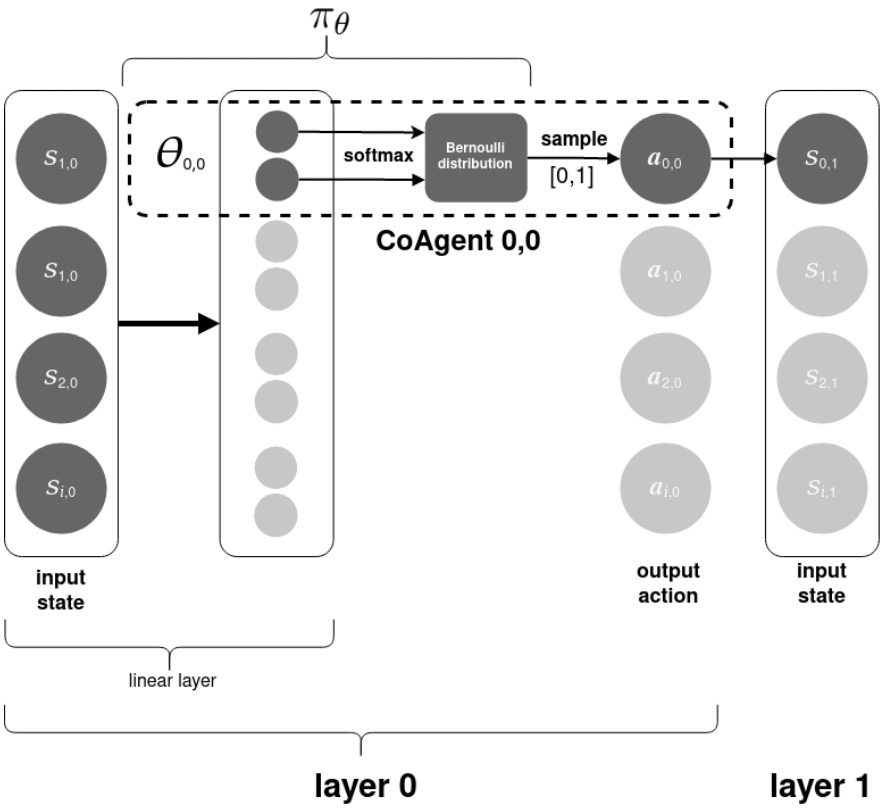
we can then describe a CoAN episodic weight update for a layer as

$$\boldsymbol{\theta}_{j,e} \leftarrow \boldsymbol{\theta}_{j,e} + \boldsymbol{\theta}_{j,e-1}\ \alpha\ Q^{\theta}(S_j, A_{i,j})\ \nabla_{\boldsymbol{\theta}} \ln \pi^{\boldsymbol{\theta}}\left(A_j \mid S_j\right) \tag{3.15}$$

or rather more simply as the equality

$$\Delta \boldsymbol{\theta}_{i,j} = \alpha\ Q^{\theta}(S_j, A_{i,j})\ \nabla_{\boldsymbol{\theta}} \ln \pi^{\theta}\left(A_{i,j} \mid S_j\right) \tag{3.16}$$

To get a better intuition of how the weight updates are carried out in our particular implementation, let us consider figure 3.2.



(a)

Figure 3.2: Single layer of CoAgent network, showing how weight vector is used with neurons and softmax function to carry out action distribution generation and sampling.

As seen in figure 3.2, in the implementations of our CoANs we use isolated backpropagation to update the weights for a CoAgent. The agents select an action from a Bernoulli distribution generated by a softmax function that takes as inputs, the outputs of two linear layer nodes. In this isolated manner we can use the policy gradient loss to backpropagate error signals from the action selected by the CoAgent to the weights of the CoAgent.

Said another way, a single CoAgent uses a linear layer parameterised by weight vector $\theta_{0,0}$ to deterministically map from an input state vector s_0 to a pair of output nodes. The values from these two nodes are then used as inputs for a softmax function to produce a Bernoulli distribution that provides probabilities for either an action of 1 or 0 being sampled. Once sampled, the CoAgent's action forms the input of the following layer $s_{0,1}$.

The equation 3.16 then describes the updates of the first CoAgent $i = 0$ in the first layer $j = 0$, for weight vector $\theta_{0,0}$, where $Q^\theta(S_{0,0}, A_{0,0})$ is a reward signal that can be global, local to the layer or local to the individual agent. For our experiments we use only global and layer-wise signals. $\ln \pi^\theta(A_{0,0} \mid S_{0,0})$ is a result of the CoAgent's selected action.

3.5 Datasets

3.5.1 MNIST

The most popular computer vision benchmark is the MNIST dataset [Deng 2012]. It consists of 28x28 grayscale images of handwritten digits from 0 to 9 with a training set of 60,000 examples, and a test set of 10,000 examples.

3.5.2 Fashion-MNIST

A common computer vision critique of the MNIST dataset is that it is too easy and overused [1]. The Fashion-MNIST was developed to be a more challenging drop-in replacement task for MNIST [Xiao *et al.* 2017]. It has the same 28x28 grayscale image format as well as 10 classes of images. The images are of various clothing or fashion items and accessories e.g. shirts, pants, shoes, bags. The dataset is from the Zalando group[2], an fashion e-commerce platform. There are 7000 images per class, a training set of 60000 images, a test set of 10000 images.

We use it in this work to see if there is a more challenging case where a REINFORCE CoAN fails and an AC CoAN succeeds, or if at least the performance gap is somewhat narrower.

3.6 Experiments

This section provides a brief overview of the experiments carried out. The results and discussion are dealt with in the following chapter.

We provide a description of experimental parameters in Section 3.6.1, while Section 3.6.2 describes the metrics measured both to ascertain performance but also to gain insight into learning dynamics.

In Section 3.6.3, we describe using Vanilla CoAN REINFORCE as a control cases, showing the bounds for successful learning in terms of breadth and depth.

In Section 3.6.4, we describe how we test **Hypothesis 1** and show how REINFORCE can use a sampled baseline to reduce noise at depth and increase performance in a CoAN. In Section 3.6.5, we explain how we test **Hypothesis 2**, first using REINFORCE with a learned baseline and then for an Actor-Critic setup.

3.6.1 Hyperparameters

The following section discusses details that apply to all of the experiments. All experiments are carried out on the finite horizon MDP described in Section 3.4. Additionally all experiments are run over at least 5 random seeds on both MNIST and F-MNIST datasets.

The behaviour in CoANs can differ significantly with respect to depth d, the number of layers in a CoAN and width w, the number of CoAgents per layer. For this reason for every test we

[1] `https://twitter.com/goodfellow_ian/status/852591106655043584`
[2] `https://jobs.zalando.com/en/tech/?gh_src=281f2ef41us`

run permutations that include at least 32, 64, 128, 256 and 512 CoAgents per layer for depths of up to 6 layers.

For our results, for visual clarity, we select results to display that we believe best represent a group's behaviour for a specific test. These groups are either narrower or wider CoANs, with 32 and 64 CoAgents per layer falling for most results into the *narrower* CoAN category and 128, 256 and 512 falling into the *wider* CoAN category. At times behaviour clusters into 3 groups where we see 128 CoAgents per layer separating into an additional *medium width* CoAN group. We point these groupings out when they occur in the results.

When running tests concerning the depth of a neural network, detecting a complete failure to learn may be difficult, as deeper neural, having potentially many more parameters to learn, may eventually reach the same asymptotic performance, albeit after many more epochs. We limit our training to 500 epochs and discuss our results in terms of relative performance within this range.

The values from Gupta *et al.* [2021] were used for additional hyperparameters. Basic hand tuning was carried out on each. For each, values were perturbed slightly higher and lower to determine the impact on training. In each of these cases the results of this hand tuning showed that values chosen by Gupta *et al.* [2021] performed as well or better than the perturbations. Hyperparameters included a batch size of 32, an actor learning rate (α) of 0.00048 and a ratio of actor to critic learning rates of 2. That is, for the critic a learning rate of 0.00024 is used. For both actor and critic networks, the learning rate used for each node is the same. Additionally, as we are trying to answer an open question from Gupta *et al.* [2021], in line with their experimental setup, we keep both these learning rates constant with respect to the CoAN breadth and depth.

The variations in the breadth and depth of the critic network showed the same trend as in Gupta *et al.* [2021], where increasing breadth and / or depth showed increases in performance. To limit computational overhead, a standard critic was chosen, using a single hidden layer of 128 nodes.

With the above in place the following optimisers were compared: RMSprop [Tieleman *et al.* 2012], Adam [Kingma and Ba 2014], stochastic gradient decent (SGD) [Amari 1993]. RMSprop consistently produced the best performance across configurations.

3.6.2 Metrics

We measure metrics to ascertain network performance as well as metrics to gain insight into the learning dynamics of the CoANs.

We use the following metrics to measure performance:

1. **Asymptotic Mean Classification Accuracy**: The highest classification reached.
2. **Convergence time**: This is the amount of time it takes to reach an asymptotic mean classification accuracy.
3. **Variance of Accuracy**: The variance of classification accuracy during training.

The behaviour of narrower and wider CoANs differ significantly. To help expand on these differences we use additional measures as suggested by Gupta *et al.* [2021], namely the the *entropy* and *sparsity* of the CoAN.

As the sum of surprise described by Shannon [1948], entropy is a useful to gauge the stochastic nature of our network.

As seen in figure 3.2a we use a softmax function to generate our CoAgent's action probability distribution from the outputs of two torch linear layer neurons. The entropy of a layer serves as an indicator for the mean stochasticity of the layer for that episode of training. That is, a high entropy for a single CoAgent arises when that CoAgent's Bernoulli distribution that is produced by that CoAgent's weight vector and is used for sampling actions, has approximately equal probabilities of providing a sampled 0 or 1, 50% for either.

By measuring the mean neuronal activation, whether a 0 or 1 is selected as an action, we can measure the mean sparsity of our learnt representations per layer.

However when the distribution is heavily skewed in either direction, for example a 99% chance for sampling a 0 and conversely a 1% chance of sampling a 1, then this entropy drops very low. One can consider a drop in mean entropy from 1 toward 0 of a layer, as a progression of that layer from stochastic toward being almost deterministic.

Additional metrics

We also record the mean and variance many other metrics for each layer during training and testing. These include but are not limited to the grad norm, critic value function loss and reward. All records of results and preliminary results are available [3].

3.6.3 Bounds and Control Case

We ran one experiment to establish a control case and to understand how varying the depth and breadth of a CoAN using REINFORCE to solve classification tasks on our datasets. The details are described in Experiment 1.

Experiment 1 Bounds and Control Case

We adapt the CoAN REINFORCE algorithm from Gupta *et al.* [2021] that uses a heterogeneous update rule for the output layer to use a homogeneous update rule throughout. We do this in order to mitigate additional potential complexities that may confound observations. The REINFORCE algorithm with homogenous update rule is shown in Algorithm 1. More on this design decision and a comparison between these two algorithms in the Appendix A.1.

The policy gradient for a CoAgent network where each neuron's policy is updated using REINFORCE can be expressed as

$$\Delta\boldsymbol{\theta}_{i,j} = \alpha\, G\, \nabla \ln \pi^{\theta}\left(A_{i,j} \mid S_j\right) \qquad \text{CoAN REINFORCE} \tag{3.17}$$

Where the undiscounted return G is the same for each CoAgent in every layer as shown in equation 3.5.

[3] https://wandb.ai/coans/coan_ac_results

Algorithm 1 CoAN REINFORCE — Homogeneous updates

Input: a dataset $\mathcal{D}$
Input: a policy parameterisation $\boldsymbol{\theta}_{i,j}$ for each CoAgent $\pi_{\boldsymbol{\theta}}(a|s)$
Algo param: step size $\alpha > 0$
Algo param: number of CoAgents per layer n, number of layers k
Initialise: each set of CoAgent policy parameters $\pi_{i,j} \in R^d$

while epoch is not terminal **do**
  for e episodes in dataset **do** ▷ where each episode is a dataset sample
    $x, y \sim \mathcal{D}$
    $S_0 \leftarrow x$

    for layer j in k layers **do**
      $a_j \sim \pi_\theta(S_j|A_j)$
      $s_{j+1} \leftarrow a_j$
    $G \leftarrow R_k \leftarrow -\operatorname{err}(a_k, y)$

    for layer j in k layers **do**
      $\boldsymbol{\theta}_{j,e} \leftarrow \boldsymbol{\theta}_{j,e} + \boldsymbol{\theta}_{j,e-1}\ \alpha\ G\ \nabla \ln \pi^{\boldsymbol{\theta}}\left(A_j \mid S_j\right)$

To roughly explore the functional bounds of a CoAN using REINFORCE updates, we test by training on our datasets. We vary the hyperparameters depth d of the layers, and width w number of CoAgents, in each layer.

Expectations

Our expectations for the results of Experiment 1 are as follows:

- *Bounds*: From existing published results, we expect the lower bounds for number of CoAgents in a single layer that can successfully learn to be about 8 CoAgents. Existing literature does not suggest an upper bound for either the number of agents in a single layer or for the number of layers at which failure to learn may occur.
- *Entropy*: The only layer that has been shown by Gupta *et al.* [2021] to exhibit very low mean entropy and become almost effectively deterministic is the first layer of a CoAN, while additional layers showing an initial drop in entropy that then rise back up, becoming once again more stochastic. We expect this behaviour for additional layers, however we have no expectation for the impact of additional CoAgents per layer.
- *Sparsity*: Based on existing research in Gupta *et al.* [2021], we expect sparsity to approximate a mean of 0.5.
- *Datasets*: Considering the experiments run on F-MNIST, the classification task is more challenging than MNIST, CoAN REINFORCE may fail but we at least expect some drop in an aspect of performance.

3.6.4 Testing Hypothesis 1 - Variance Reduction with Unbiased Sampled Baseline

We ran one experiment to determine if variance reduction is beneficial at all tested depths of CoANs. The details are described in Experiment 2.

Experiment 2 Variance Reduction with Unbiased Sampled Baseline

As CoANs lie at the intersection of Multi-agent reinforcement learning and neural networks making assumptions about behaviour is not always possible. Despite there being a good chance that less variance will improve performance, we choose to test and show this explicitly.

With this experiment we show using a sampled baseline as an unbiased variance reduction method, that variance is a component of this noise and that it can be somewhat mitigated with this method at all depths tested.

A perfect baseline would be the true value of a state. Without an oracle or the opportunity to take an infinite number of samples, we are only able to approximate the true value of a state. As we wish to better understand the cause of poor state-value estimation bias at deeper layers of our CoAN, we can compare baseline methods that do not use this form of estimation, such as an average of sampled returns, and those that do, such as learnt state-value function.

For information of why a baseline decreases variance without increasing bias please see section 2.2.2.

We use the same running average baseline as Gupta *et al.* [2021] that updates using the sampled return G for each episode trajectory e described by

$$\begin{aligned} \hat{G}_e &= \eta\, \hat{G}_{e-1} + (1-\eta) G_e \\ \hat{G}_e &= b \end{aligned} \tag{3.18}$$

Where η is a decay parameter. For this experiment, as in previous work we use $\eta = 0.99$. As shown in the algorithm 1 there is a single dataset sample per episode e. For example, $e-1$ refers to the previous episode or dataset sample.

The running average baseline is used with Algorithm 1 modifying the REINFORCE update as follows:

$$\Delta \boldsymbol{\theta}_{i,j} = \alpha\, (G - b)\, \nabla \ln \pi^{\theta}\left(A_{i,j} \mid S_j\right) \qquad \text{CoAN REINFORCE with baseline} \tag{3.19}$$

where the baseline b is calculated using Equation 3.18.

Expectations

Policy gradient methods exhibit high variance in practice as discussed in section 2.2. Variance reduction has been shown to be a key intervention for these methods [Sutton and Barto 2018b]. CoANs can be considered a multi agent reinforcement learning (MARL) systems [Thomas and Barto 2011] and variance reduction has been shown to improve performance of MARL systems

[Buşoniu *et al.* 2010; Canese *et al.* 2021a]. We expect that there will be some increase in performance for a CoAN of any depth and any number of CoAgents per layer.

3.6.5 Testing Hypothesis 2 - Variance Reduction with Biased Learned Value Functions

For learned base lines and Actor-Critic CoANs there are a number of various ways one could produce a state-action value function or critic in our multi-agent setting where CoAgents can share a value functions and their estimates. We could (1) train a value function to produce a value estimate for each CoAgent $q(S_j, A_{i,j})$ similar to the critic used in Foerster *et al.* [2018]. Alternatively (2) a layer-wise value function $q(S_j, A_j)$ or perhaps even (3) a value function for the entire network over an episode $q(S, A)$. Here we choose (2) a layer-wise value function, such that there is a separate value function and therefore a state or state-action value estimate for each layer. This setup produced the best results between (2) and (3) and avoided the implementation complexity of (1) which is beyond the scope of our current investigation. We do believe that (1) is an opportunity for further investigation.

Here we run four experiments. In experiment 3 we run an experiment to test the impact of bootstrapping bias on learning for CoANs using Actor Critic methods, as well as the impact that depth has. Experiment 4 tests the impact of depth on performance of learned baselines. Experiments 5 and 6 test the impact of adding noise to the value functions of learned baselines and Critic.

Experiment 3 Critic Bootstrapping and Actor-Critic at Depth

For a background on Actor-Critic and temporal difference methods please see section 2.2.3.

The simplest possible Actor-Critic algorithm that we can apply to a CoAN is a Q value Actor-Critic, CoAN Q-AC 2, where a state-action value function is used for the critic to give the policy gradient

$$\Delta\boldsymbol{\theta}_{i,j} = Q_j^w\left(S_j, A_j\right)\nabla \ln \pi_{i,j}^{\theta}\left(A_{i,j} \mid S_j\right) \qquad \text{CoAN Q-AC} \tag{3.20}$$

We can also use an Advantage Actor Critic to give us a CoAN A2C 2, where the update becomes

$$\Delta\boldsymbol{\theta}_{i,j} = \left(Q_{w,j}\left(S_j, A_j\right) - b\right)\nabla \ln \pi^{\theta}\left(A_{i,j} \mid S_j\right) \qquad \text{CoAN A2C} \tag{3.21}$$

Algorithm 1 is modified to include the critic updates for CoAN Q-AC and CoAN A2C such that

Algorithm 2 CoAN Q-AC and A2C — Temporal Difference Critic Updates

Input: a dataset $\mathcal{D}$
Input: a policy parameterisation $\boldsymbol{\theta}_{i,j}$ for each CoAgent $\pi_{\boldsymbol{\theta}}(a|s)$
Algo param: step size $\alpha > 0$
Algo param: number of CoAgents per layer n, number of layers k
Initialise: each set of CoAgent policy parameters $\pi_{i,j} \in R^d$

while epoch is not terminal **do**
  for e episodes in dataset **do** ▷ where each episode is a dataset sample
    $x, y \sim \mathcal{D}$
    $S_0 \leftarrow x$

    for layer j in k layers **do**
      $\boldsymbol{w}_j \leftarrow \boldsymbol{w}_{j-1} + \alpha \left(Q_{w,j}(s_j, a_j) - Q_{w,j-1}\left(s_{j-1}, a_{j-1}\right)\right) \nabla \boldsymbol{w}_{j-1} Q_{w,j-1}\left(s_{j-1}, a_{j-1}\right)$
      $s_{j+1} \leftarrow a_j$
    $G \leftarrow R_k \leftarrow -\operatorname{err}(a_k, y)$

    for layer j in k layers **do**
      $\boldsymbol{\theta}_{j,e} \leftarrow \boldsymbol{\theta}_{j,e} + \boldsymbol{\theta}_{j,e-1}\, \alpha\, Q_{w,j}(s_j, a_j)\, \nabla \ln \pi^{\boldsymbol{\theta}}\left(A_j \mid S_j\right)$ ▷ actor update

To note for the algorithms 3 and 2 of CoAN Q-AC and CoAN A2C, for brevity we have combined their algorithmic depiction. This shows only CoAN Q-AC, to modify them for CoAN A2C, simply add a baseline term to the actor updated, as such $(Q_{w,j}(s_j, a_j) - b)$.

The update for the critics using MC methods is described as

Algorithm 3 CoAN Q-AC and A2C — Monte Carlo Critic Updates

Input: a dataset $\mathcal{D}$
Input: a policy parameterisation $\boldsymbol{\theta}_{i,j}$ for each CoAgent $\pi_{\boldsymbol{\theta}}(a|s)$
Algo param: step size $\alpha > 0$
Algo param: number of CoAgents per layer n, number of layers k
Initialise: each set of CoAgent policy parameters $\pi_{i,j} \in R^d$

while epoch is not terminal **do**
  for e episodes in dataset **do** ▷ where each episode is a dataset sample
    $x, y \sim \mathcal{D}$
    $S_0 \leftarrow x$

    for layer j in k layers **do**
      $a_j \sim \pi_\theta(S_j|A_j)$
      $s_{j+1} \leftarrow a_j$
    $G \leftarrow R_k \leftarrow -\operatorname{err}(a_k, y)$

    for layer j in k layers **do**
      $\boldsymbol{w}_{j,e} \leftarrow \boldsymbol{w}_{j,e-1} + \alpha \left(G_{e-1} - Q_{w,j}\left(s_j, a_j\right)\right) \nabla \boldsymbol{w}_{j,e-1} Q_{w,j}\left(s_j, a_j\right)$ ▷ critic update
      $\boldsymbol{\theta}_{j,e} \leftarrow \boldsymbol{\theta}_{j,e} + \boldsymbol{\theta}_{j,e-1}\, \alpha\, Q_{w,j}(s_j, a_j)\, \nabla \ln \pi^{\boldsymbol{\theta}}\left(A_j \mid S_j\right)$ ▷ actor update

To note, using a Monte Carlo update for the critic remove the potential benefit from temporal difference variance reduction. However we do this here as a means to show the impact that bootstrapping bias has in a CoAN.

Expectations

From existing work on CoAN Actor-Critic methods Gupta *et al.* [2021], we expect the performance of these methods to be significantly poorer than our control case of a CoAN using REINFORCE updates.

As no results have been published on the affect of increased depth to CoAN Actor-Critic methods and the cause for poor performance of these methods is not understood, we do not have an expectation for the performance of these methods in deeper CoANs.

As bootstrapping is a notorious source of instability, being one of the three components of deadly triad [Sutton and Barto 2018b], the other two being function approximation and off-policy methods. For a CoAN system that is noisy and unstable, one could expect that removing it would provide an increase in performance, however the CoAN is also a high variance system, such that it is possible that updates over shorter trajectories could result in increased performance. For these reason we did not have a set expectation for the affect of the bootstrapping for critic updates.

Experiment 4 Learned Baseline at Depth

Here we test the impact of depth on learned baselines. To apply a learned baseline to REINFORCE we use the same CoAN REINFORCE with baseline update equation 3.19 as in sampled baselines. Where b is a learned value function instead of a sampled return. We use the same value estimation methods as Gupta *et al.* [2021]. A parameterised value function V_w for each layer j of the CoAN is learned using Monte Carlo updates.

$$\boldsymbol{w}_{j,e} \leftarrow \boldsymbol{w}_{j,e-1} + \alpha \left(G_e - V_{w,j}\left(\boldsymbol{s}_j\right)\right) \nabla \boldsymbol{w}_{j,e-1} V_{w,j}\left(\boldsymbol{s}_j\right) \quad \text{state value function} \tag{3.22}$$

We compare the performance to our vanilla REINFORCE control case in experiment 1 and to REINFORCE with sampled baseline in experiment 2

Expectations

From existing work [Gupta *et al.* 2021] we expect the performance to be good in shallow CoANs. We do not have expectations for performance in deeper CoANs.

Experiment 5 Adding Noise to Critic Observations

To have adequate performance to be able to show drops from introducing additional noise, we carry out noise tests using Monte Carlo critic updates using CoAN A2C from algorithm 3. We compare the performance between a shallower and deeper CoAN with noise injected into the last hidden layers value function.

When injecting noise into a layer's value function, we consider the nature of state observations differ for the input layer and for the hidden layers. The input layer state observation vector

s_0, an image, is $\in R^n$ whereas the hidden layers state observation vector s_h is a binary output from the preceding layer is $\in 2^n$. This has significant consequences when adding noise to a signal. Consider that one can apply an arbitrary measured amount of Gaussian noise to the entire input image vector, however a similar operation to an entire hidden layer state observation vector would scramble or make completely noisy the entire vector. For this reason, for a fair and consistent comparison between layers, we only compare the impact of noise to hidden layers value estimators.

To inject noise into a hidden layer, instead of applying noise to the entire vector we instead select an upper bound percentage p of the vector. For each noise transform we then randomly vary actual percentage of the vector that receives this noise up to that threshold. This is done so that there is also randomness in the percentage of noise added with each transform.[4]

State observation vector s of the deepest hidden layer's state observation $j = k - 1$ to create a noisy state observation vector s'. We choose this value function at this layer and not the output layer $j = k$ as the output layer will always have only the number of CoAgents as the number of prediction classes.

As mentioned in section 3.6.1 we run all tests on a permutation of 32, 64, 128, 256 and 512 CoAgents per layer and select result that are a good representative of a group's behaviour. Additionally all tests were carried out with permutations for amounts of 0%, 10%, 20%, and 30% noise being injected into the critic state observation. The graded impact of this is demonstrated in figure 4.9.

Experiment 6 *Adding Noise to Baseline Value Function Observations*

To investigate what might be the cause of this decrease in performance that only occurs for deeper CoANs using REINFORCE with a learned baseline, we test the sensitivity to additional noise in the value function's state observation. Compare this sensitivity in shallower and deeper CoANs.

Noise is generated and added to value function state observation in the same manner as experiment 5. We use a noisy state observation s' for the value function such that the weight updates are then

$$\boldsymbol{w}_{j,e} \leftarrow \boldsymbol{w}_{j,e-1} + \alpha \left(G_e - V_{w,j}\left(\boldsymbol{s'}_j\right)\right) \nabla \boldsymbol{w}_{j,e-1} V_{w,j}\left(\boldsymbol{s'}_j\right) \quad \text{noisy state value function} \tag{3.23}$$

This gives us the following policy gradient for a CoAN using REINFORCE with a learned baseline, such that the baseline is trained and tested with noisy observations

$$\Delta \boldsymbol{\theta}_{i,k-1} = \alpha \left(G - V_w(s'_{i,k-1}\right) \nabla \ln \pi^{\theta}\left(a_{i,k-1} \mid s_{i,k-1}\right) \quad \text{CoAN REINFORCE with noisy baseline} \tag{3.24}$$

As a control however, when $p = 100\%$, that is 100% noise, we use a completely artificial noise signal for all the value functions of all layers. This is to demonstrate an observed behaviour

[4]However at the time of writing is has become apparent that this adds an extra layer of complexity that is unnecessary, however it is sufficient as a source of noise and we do not expect it to negatively impact the results of our experiments. A simpler method that we did use in preliminary experiments simply flipped up to 50% of the bits and then randomly permuted those flipped bits.

of the learned baseline in this context. That is, when the learned baseline is trained on only noise, it approximates a rolling average baseline. We expand on these observations in the results, showing as well incremental changes in behaviour with the addition of noise.

Expectations

From existing work Gupta *et al.* [2021] we expect to see good performance when used with shallow CoANs. We then expect to see progressively worse performance when used with deeper CoANs. As Schoenholz *et al.* [2017]; Pretorius *et al.* [2020] suggests, for stochastic neural networks, due to the stochastic noise in each successive layer as the CoAN gets deeper, the signal gets nosier, making information propagation harder, which we expect will make it harder for the value function to produce a good estimate.

3.7 Summary

In this chapter we provide a problem statement and research questions extending the work of Gupta *et al.* [2021]. In these we directly address the open question from their work: Why is the bias introduced by a critic in an Actor-Critic CoAN extremely detrimental to learning?

We introduce two hypotheses directed at variance reduction methods in CoANs in order to specifically probe these questions. We utilise the same problem definition as Gupta *et al.* [2021], but provide our own interpretation. This problem definition serves as the general foundation for all of our experiments. We provide a detailed description and set of expectations for each of our six experiments.

In the following Chapter 4 we provide the results of these six experiments with a discussion for each.

Chapter 4

Results and Discussion

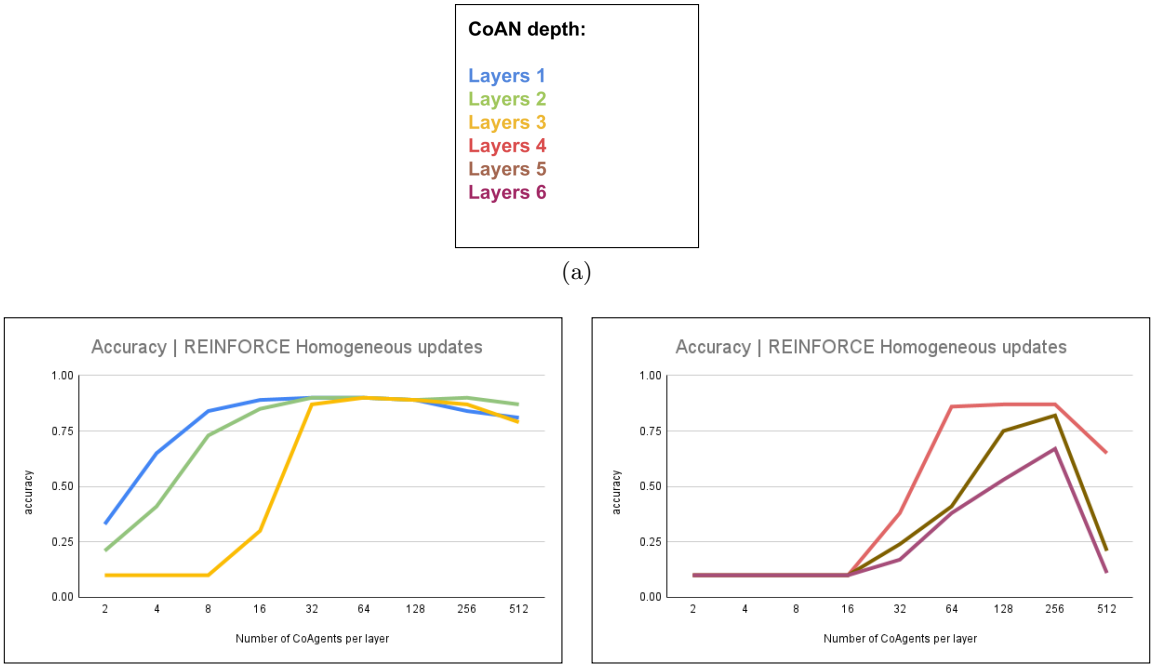
The analysis of the results is organised with respect to the experiments outlined in Chapter 3. Each section of this chapter provides results and discussion for each of these experiments.

In all results, when referring to *CoAN depth*, the output layer is omitted. The output layer will always have as many CoAgents as prediction classes where as we vary the number of CoAgents for the input layer and all hidden layers. This is the number indicated for instance on the x-axis of the Figure 4.1b, and Figure 4.1c.

Regarding additional metrics to our main performance metrics such as entropy and sparsity of CoANs, as described in section 3.6.2, we specify if they deviate significantly from what is depicted in experiment 1, otherwise it can be assumed we did not see a significant discernible trend from them during training.

For all experiments we saw a general trend of training with less variance on the F-MNIST dataset as compared with MNIST. This is a surprising result as F-MNIST was made to be a more difficult dataset as we mentioned in Section 3.5.2. The asymptotic performance was however consistently lower on F-MNIST. We show results for both MNIST and F-MNIST in different cases. One can assume trends were consistent on both datasets unless we specify that they were not.

4.1 Results for Experiment 1 - Bounds and Control Case



(a)

(b)

(c)

Figure 4.1: Mean accuracy overview for CoANs of various depths and widths after 500 epochs using REINFORCE algorithm 1

Figure 4.1, shows a summary of runs such as those depicted in figure 4.2. It shows us that at the extremes of depths and widths we see different behaviours in performance. Figure 4.1b shows us the accuracy for shallower CoANs. For CoANs of 1 and 2 layers, widths of 2-8 CoAgents show the ability to learn, albeit at a much lower asymptotic accuracy as compared to all wider CoANs. In layer 3 we see that widths of 2-8 CoAgents do not learn. Widths of 16 CoAgents can learn but only to a very low accuracy of approximately 26%. Widths of 32 to 512 CoAgents all learn to over 75% for shallow CoANs.

In Figure 4.1c, we see deeper CoANs. Widths of 2-16 CoAgents fail to learn at these depths. A width of 32 CoAgents shows some learning but only to an accuracy of 40% for four layer CoANs, 25% for 5 Layer CoANs and 20% for 6 Layer CoANs. Widths of 64, 128 and 256 CoAgents per layer reach an accuracy of over 75% for layer 4. We see that only a width of 128 and 256 CoAgents get an accuracy over 75% for depths of 6 layers. Only a width of 256 get over 75% accuracy at a depth of 6 layers. We see a sharp drop off of accuracy for 512 CoAgents per layer for CoANs of depth 5 and 6 layers.

To summarise the above, we see that very narrow CoANs do not learn well at depths and very wide CoANs do not learn well at depth

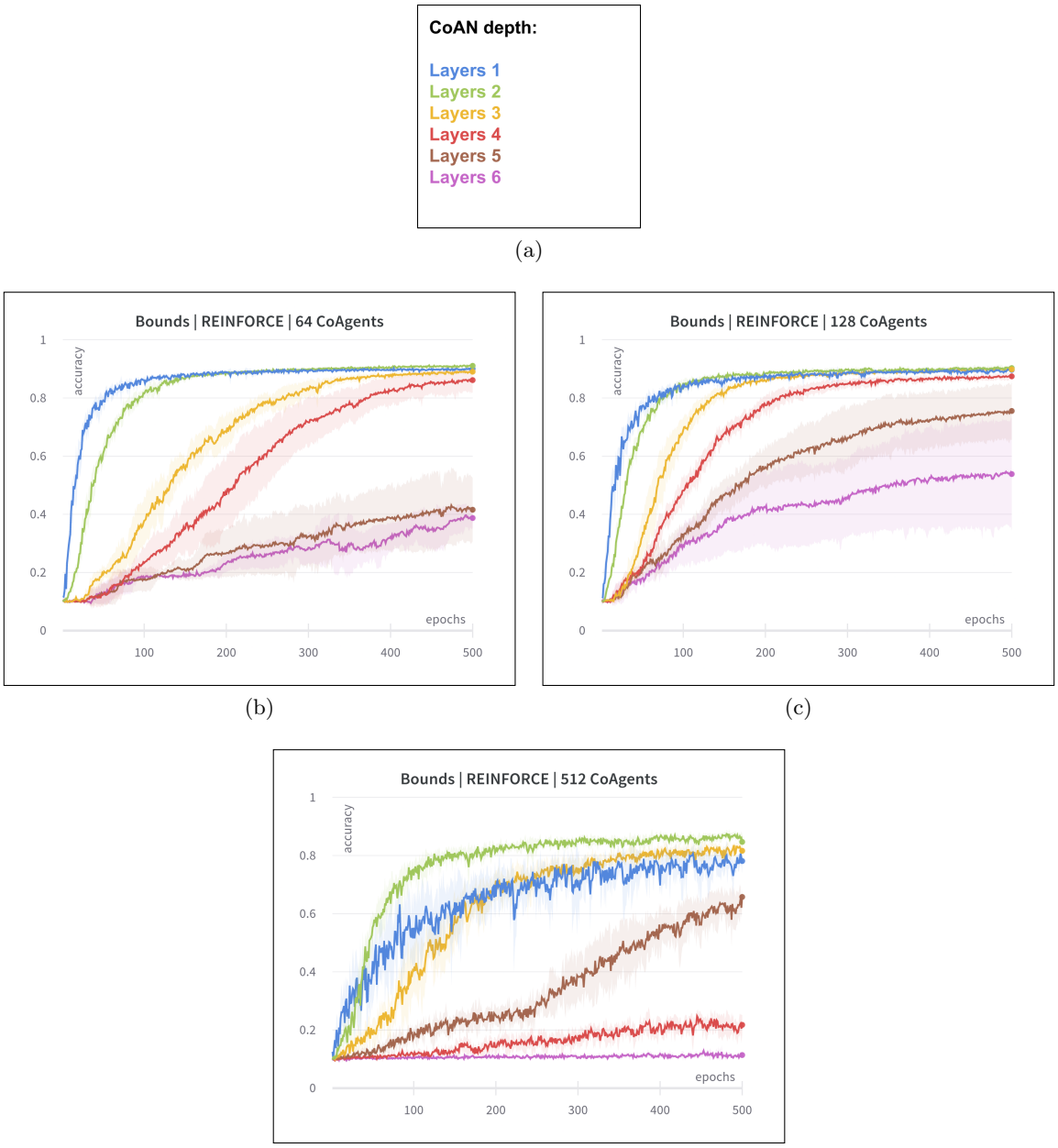


Figure 4.2: Mean accuracy for CoANs various depths and widths after 500 epochs using REINFORCE algorithm 1

In figure 4.2 we see the same general pattern of learning as in figure 4.1. Here we have information on the variance of accuracy during training and the trajectory of mean accuracy per epoch. Figure 4.2d for a CoAN of 512 CoAgents per layer is notable as it indicates that for this width, it is the only case tested where a single layer CoAN is outperformed by deeper CoANs. In figures 4.2b and 4.2c we see an notable increase in accuracy variance in addition to a drop in asymptotic accuracy.

Both figures 4.2 and 4.1 show that the performance CoANs is significantly impacted by an increased depth.

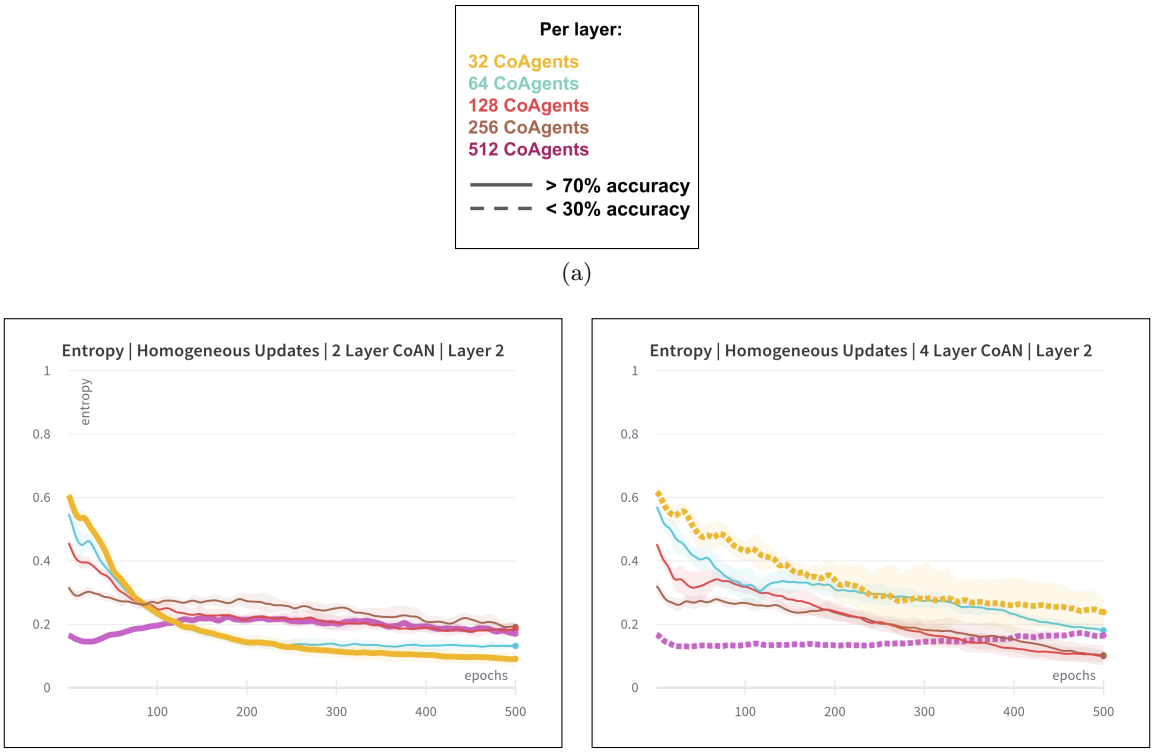


Figure 4.3: Entropy from layer 2 for CoANs of depth 2 and 4 layers

We recorded metrics for each layer's entropy during training. We show figure 4.3 as it both depicts the general behaviour of all hidden layers' entropy well and it shows two training trajectories for a narrow and wide CoAN that learn successfully at a shallow depths but fail in deeper CoANs. The dashed lines show cases of very poor performance. This gives us some potential insight into the behaviour of the networks in cases of good and poor performance. For 32 CoAgents per layer, for poor performance we see that the entropy does not drop as low as in the case of higher performance and that the variance of the entropy is noticeably higher. For 512 CoAgents per layer, we see that, while subtle, the curve in the case of higher performance dips and then rises, where as in the case of lower performance the curve is flat.

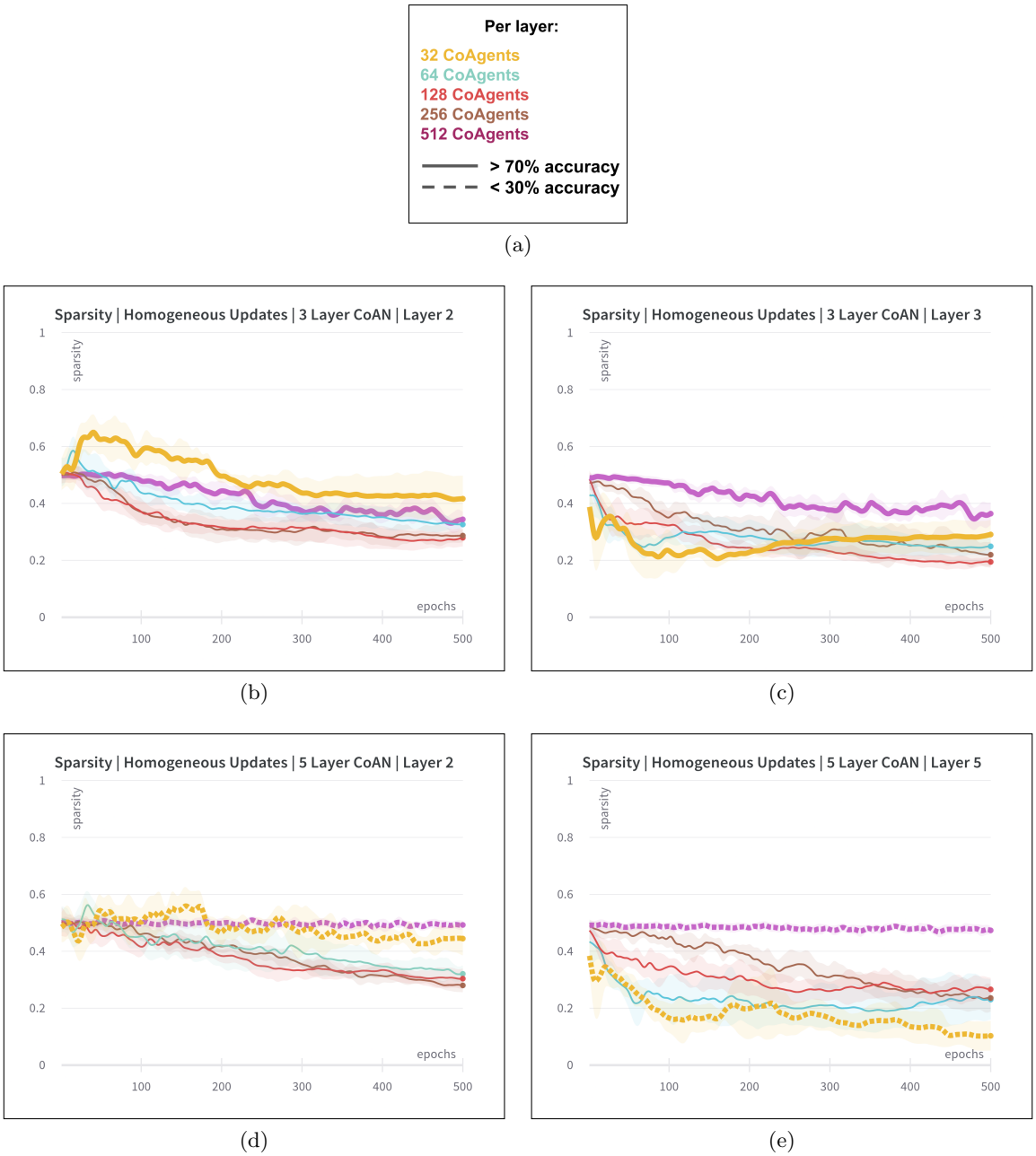


Figure 4.4: Sparsity from layer 2 and 3 for CoANs of depth 3 and 5 layers

In figure 4.4 we have selected cases to show the difference in sparsity for cases of high performance and cases of poor performance.

Figures 4.4b and 4.4c show the sparsity of narrow and wide CoANs performing well in a 3 layer CoAN. The bold lines can then be contrasted with their counterparts of poor performance in a 5 layer CoAN.

We noted a general trend towards moderately sparse representations for good performance.

4.2 Results for Experiment 2 - Variance Reduction with Unbiased Sampled Baseline

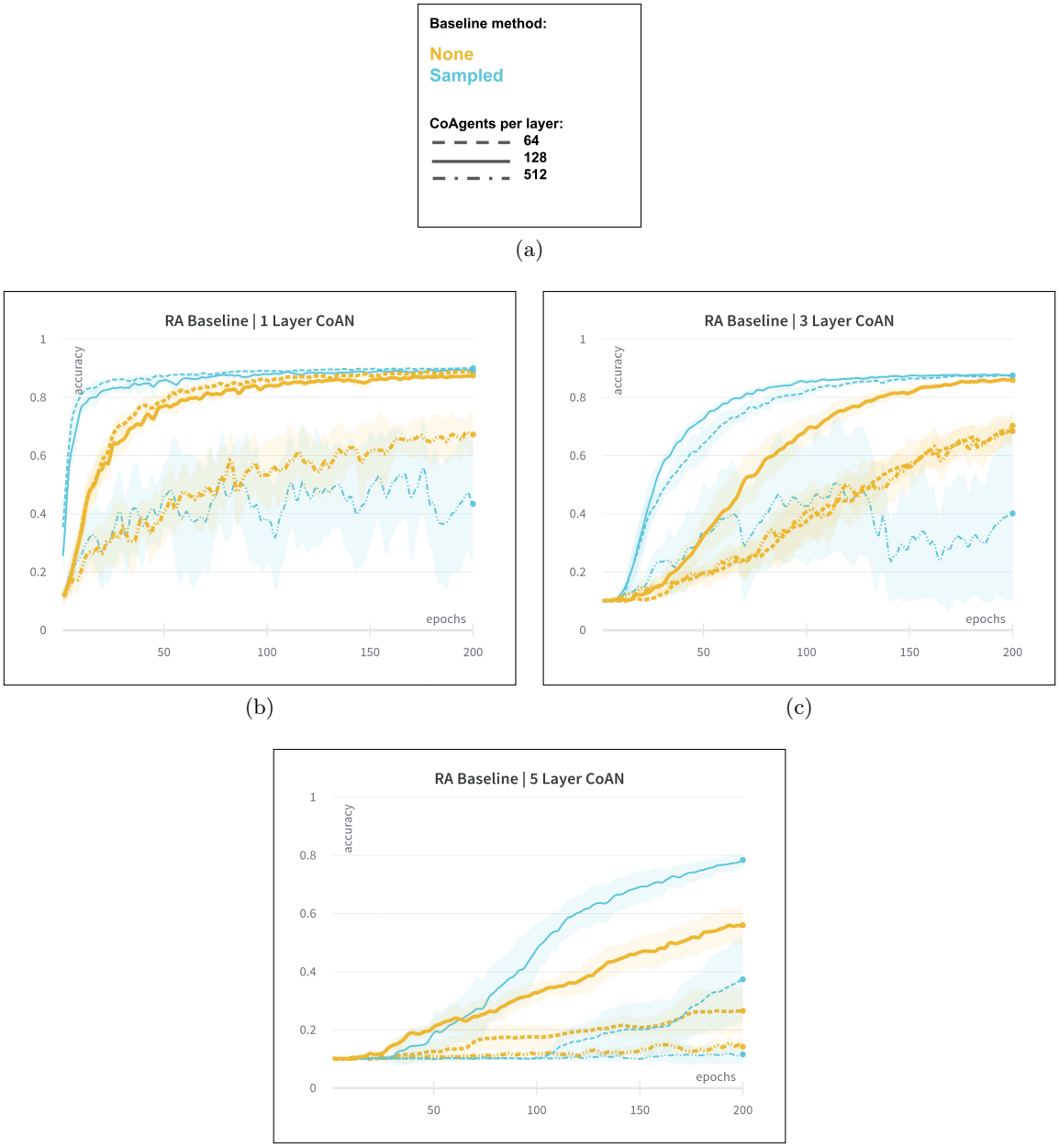
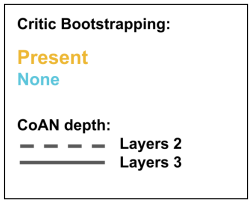


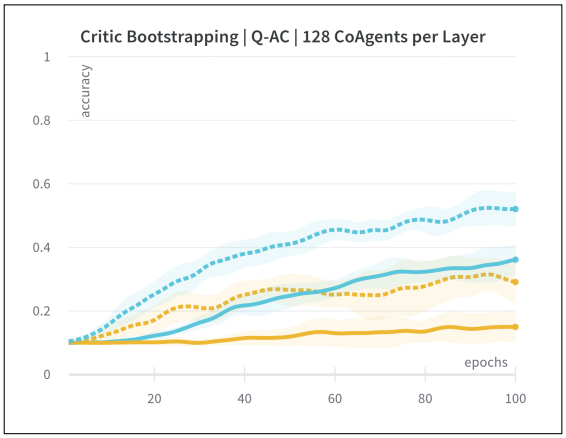
Figure 4.5: Accuracy for REINFORCE with sampled baseline across various CoAN depths with various numbers of CoAgents per layer.

Interestingly our expectations and hypothesis 1 hold only for a sufficiently small widths of CoAgents. As we can see for all figures in 4.5 the performance of CoANs with widths of 64 and 128 CoAgents per layers benefiting from a sampled baseline, whereas the performance of CoANs with widths of 512 CoAgents becoming highly varied. It appears that for CoANs of any depth, when there are sufficiently many CoAgents, a sampled running average baseline impedes performance. This is an area for possible further exploration.

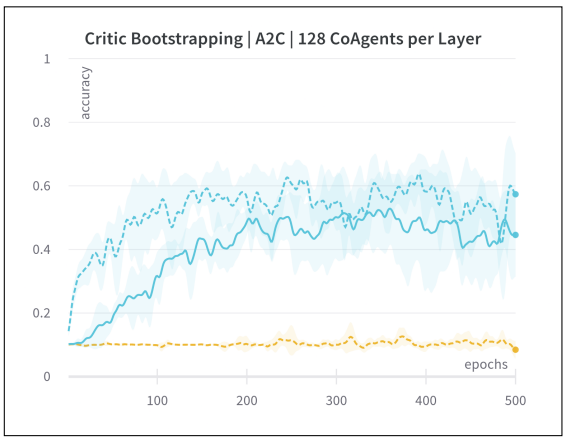
4.3 Results for Experiment 3 - Critic Bootstrapping and Actor-Critic at Depth



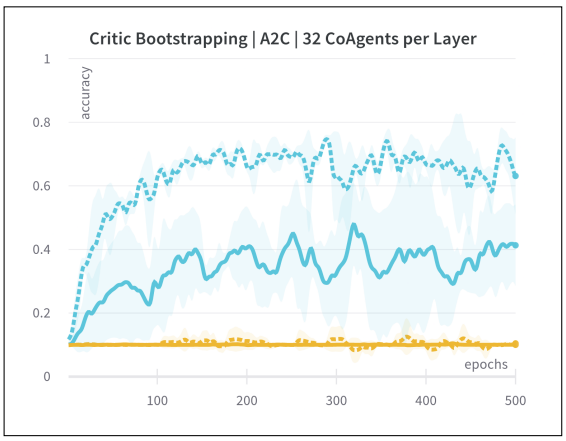
(a)



(b)



(c)



(d)

Figure 4.6: Critic Bootstrapping, F-MNIST

In figure 4.6b we can see that Q-AC generally has a low performance but is able to learn using bootstrapping. Interestingly as seen in figures 4.6c and 4.6d even though A2C has higher performance for Monte Carlo updates, the bootstrapping bias has a much larger impact, impeding learning completely. The reason for this is not clear. In both cases though we can see that bootstrapping bias has a very significant impact on learning. We believe this bootstrapping bias is a major contributor to poor performance of Actor-Critic as noted by Gupta *et al.* [2021]. This does show however that understanding how bootstrapping for CoANs is a potential area of future research, though this may necessitate reformulation of the problem.

4.4 Results for Experiment 4 - Learned Baselines at Depth

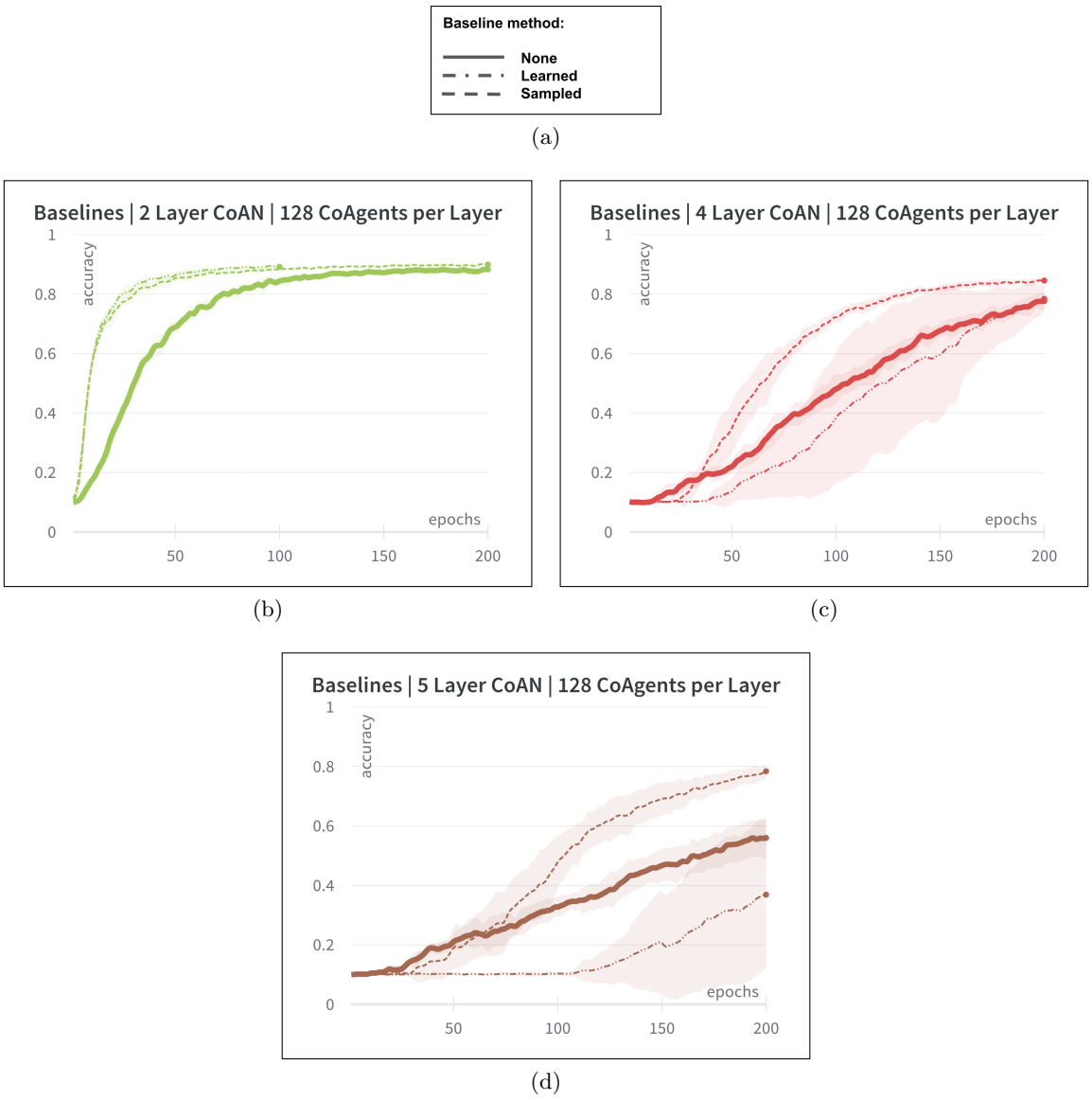
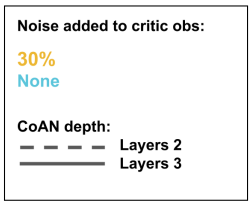


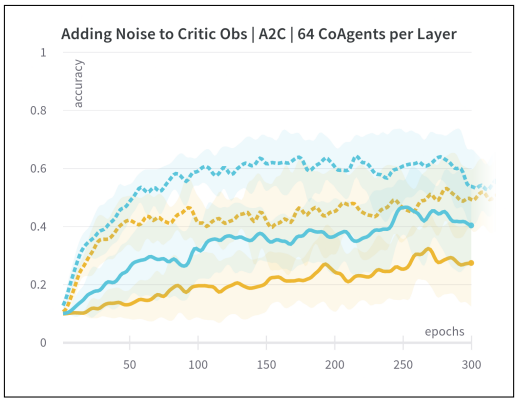
Figure 4.7: Learned baseline applied to every layer of the CoAN. As depth increases, learned baseline negatively affects learning.

In figure 4.7 we can see the progressive decrease in performance of a learned baseline for CoANs of all widths tested with an increase in CoAN depth. While the asymptotic performance does not deteriorate necessarily, we do see a significant increase in variance. The contrast in performance between the learned baseline and the sampled baseline as depth increases is especially notable. We can see in figure 4.7b that the REINFORCE with a learned baseline out performs a sampled baseline in a two layer CoAN. The deteriorating performance is then evident as the depth increases to four layer CoAN in figure 4.7c and then to a five layer CoAN in figure 4.7d

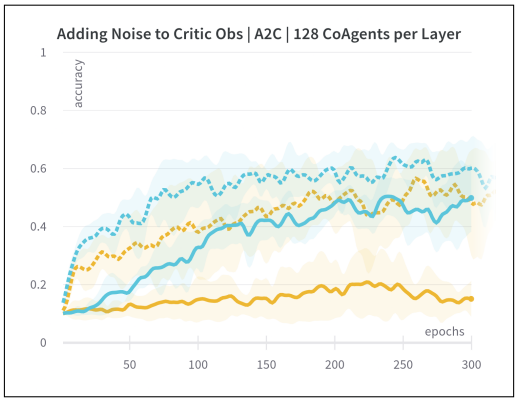
4.5 Results for Experiment 5 - Noisy Value Function Observations - Actor-Critic



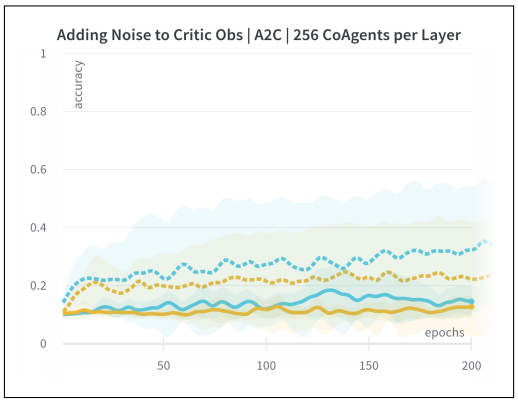
(a)



(b)



(c)



(d)

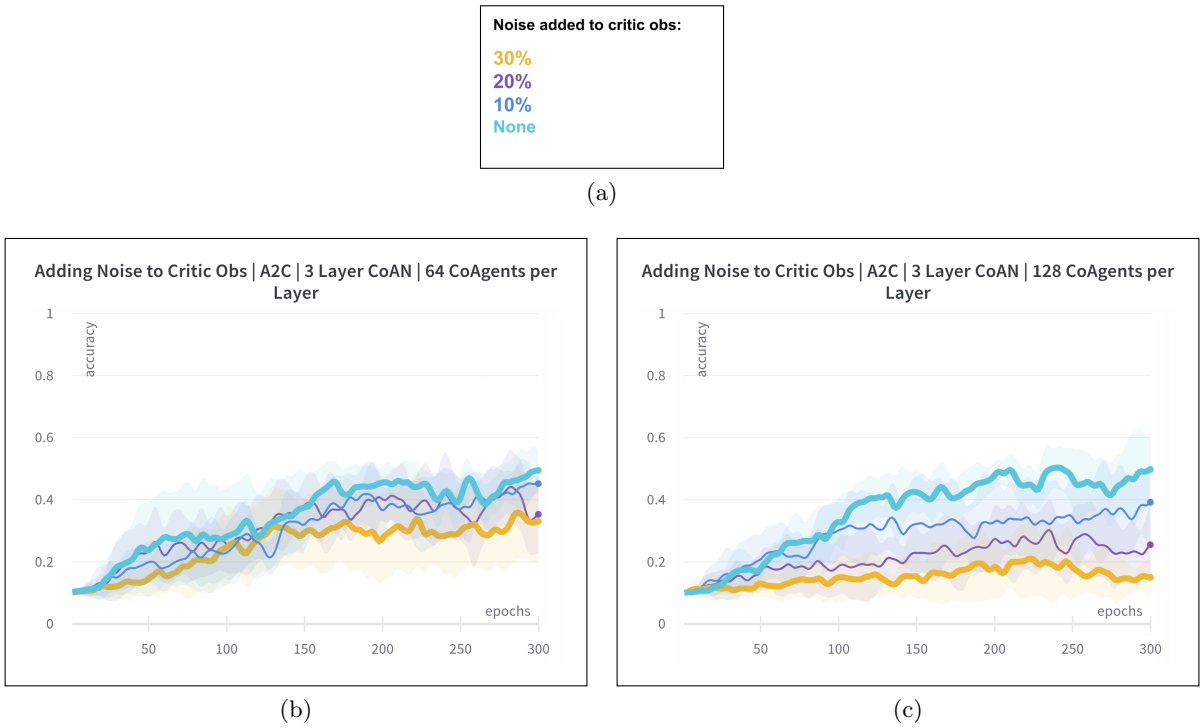
Figure 4.8: Adding noise to critic state observations. Accuracy on F-MNIST

We see an example of narrower CoAN group behaviour in figure 4.8b of 64 CoAgents per layer and an example of wider CoAN group in figure 4.8c of 128 CoAgents per layer. Figure 4.8d[1] is an example of wider CoAN group behaviour.

As the 32 CoAgent per layer had similar performance to the 64 CoAgents per layer, albeit slightly less stable is not shown. The 512 CoAgents per layer test did not learn, however we can see already that the 256 CoAgent per layer in figure 4.8d result as seen in figure demonstrates a signification drop in performance. As the accuracy of the wider CoANs groups is so low and the variance of the accuracy so high for both the case of no noise in blue and where 30% noise is injected into the last hidden layer critic state observation, shown in yellow, it is an unreliable indication of the impact of noise that is being tested. Thus for this experiment we consider the 128 CoAgent per layer result to be representative of the group behaviour for wider CoANs.

We can see that the narrower CoAN of 64 CoAgents per layer in figure 4.8b is not sensitive to noise added to the last hidden layer critic state observation and the accuracy does not differ significantly between a two layer and three layer CoAN for a 30% injection of noise. That is, there is a consistent difference between the change in performance between the blue and yellow dashed lines of a two layer CoAN and the solid lines of the 3 layer CoAN.

As seen in figure 4.8c there is a significant difference present for a wider CoAN of 128 CoAgents per layer. This confirms our hypothesis 2 for a subset of CoAN widths.



(a)

(b)

(c)

Figure 4.9: Adding noise to critic state observations. Accuracy on F-MNIST. Graded noise injection.

Figure 4.9 serves to demonstrate the behaviours of narrower and wider CoANs as noise is incrementally added in 10% intervals from no noise up to 30% noise. This figure repeats some of the

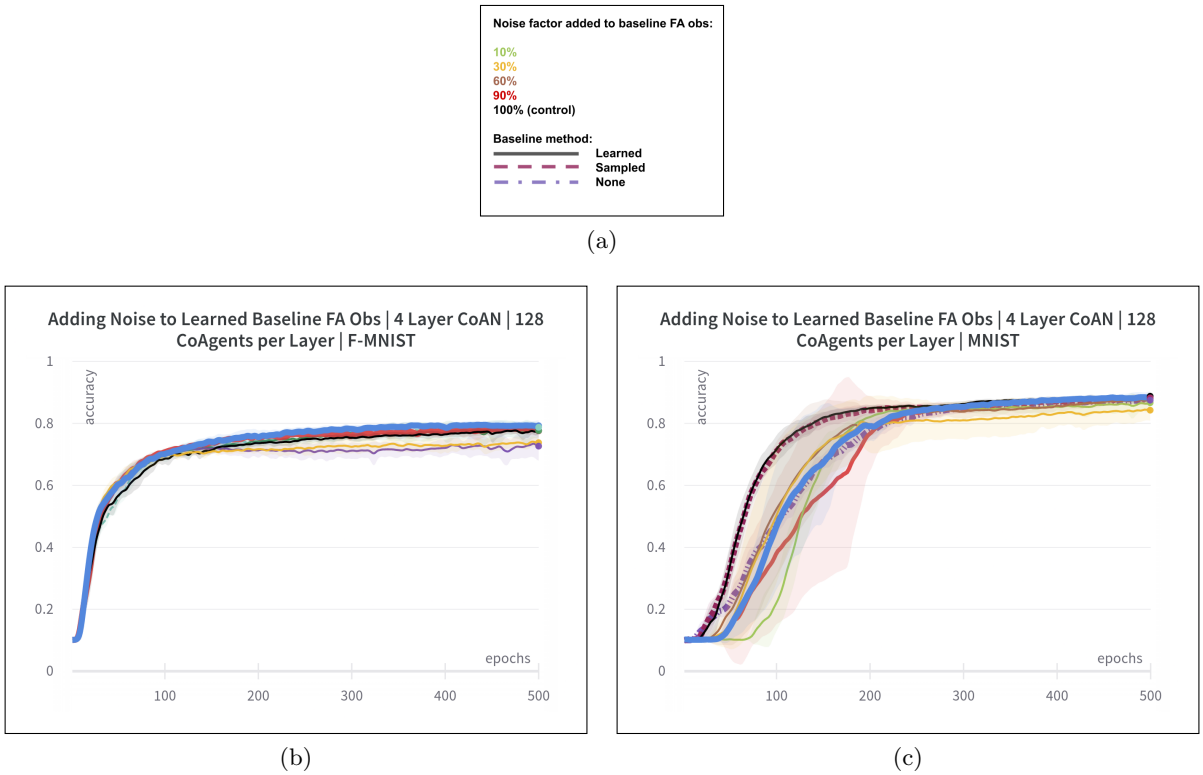
[1]The number of epochs for this run is truncated at 200. Permutations involving deeper and wider CoANs take much longer to run making them more susceptible to compute cluster outages, which is the cause of the truncation here

results from figure 4.9 but adds new plots for 10% and 20% amounts of noise. Repeated results are shown in bold.

Figure 4.9b shows the case of the narrower CoAN, where we can see that with each incremental addition of noise, the impact is not significant as each plot is deeply embedded in the variance of one another. In figure 4.9c we see the wider CoAN of 128 CoAgents per layer. We can clearly see the incremental impact of noise with each addition 10% of noise injection.

Our results do not show why a narrower CoAN is not impacted with increased depth in the same manner a wider CoAN is. This is an area for possible future investigation. One may consider an investigation on the impact of higher dimensional inputs for value functions in CoANs, as higher dimensional inputs have been shown to be impactful in neural network function approximators in other contexts [András 2018].

4.6 Results for Experiment 6 - Noisy Value Function Observations - Learned Baselines



(a)

(b)

(c)

Figure 4.10: Adding noise to baseline state value function state observation.

Adding noise to a learned baselines value function did not produce a fruitful result. There was no immediate pattern or trend that we observed. We show the figure 4.10 as it does contrast the behaviour of learned based lines when used on different datasets MNIST and F-MNIST.

With the incremental addition of noise to the learned baseline state observation irrespective of the amount of noise, relative to the learned baseline without the addition of noise, at different widths and depths one sees no consistent increase or decrease in performance in terms

of asymptotic accuracy or the variance of accuracy during training. The asymptotic accuracy could be relatively increased or decreased in no specific pattern with respect to the amount of noise added. The same can be said for the accuracy variance.

Notable there was no accuracy higher than that of REINFORCE with a sampled baseline observed and the control case of 100% noise followed a trend that approximated the sampled baseline.

4.7 Summary

We summarize the results of our experimentation by highlighting the key findings from each experiment, as follows:

- *Experiment 1*: We demonstrated that for CoANs using Vanilla REINFORCE, performance deteriorates with depth.
- *Experiment 2*: We demonstrated that variance reduction is beneficial at all depths that were tested, but only for a subset of CoAN widths. This confirms Hypothesis 1 for narrower networks, but not for wider networks, in which performance was hindered by sampled baselines.
- *Experiment 3*: We showed that bootstrapping bias is significant for CoANs using Actor-Critic methods.
- *Experiment 4*: We showed that learned baselines performance deteriorates extensively with depth.
- *Experiment 5*: We confirm Hypothesis 2 for Actor-Critic on wider CoANs, but not on narrower CoANs i.e. For wider, deeper CoANs, the impact of noise on a Critic value function's state observation is detrimental.
- *Experiment 6*: We found no conclusive evidence.

Chapter 5

Conclusion

The aim of this work was to investigate how learning dynamics of CoAgent Networks (CoANs) impacts learned value functions in deeper CoANs. In order to investigate this, we extended the work of Gupta *et al.* [2021] by incorporating investigations where we varied the width and depth of the CoANs, and injecting noise into the state observations of the learned function values, comparing impact on performance in various configurations.

Our experiments showed that CoAgent networks (CoANs) using REINFORCE policy gradient methods perform significantly worse as the depth of the CoAN increases. We showed that variance reduction with a sampled baseline is a means of increasing performance for narrower CoANs at depths. However for CoANs that are sufficiently wide, this variance reduction method decreases performance not only at depth by also in shallower CoANs. Variance reduction is however not sufficient to completely stabilise learning in deeper CoANs and despite variance reduction there is a sharp drop off in performance with increasing depth.

Toward answering the open question on the poor performance of Actor-Critic methods in CoANs from Gupta *et al.* [2021], we showed that the variance reduction methods that use learnt value functions are both negatively impacted by increasing depth in CoANs. In Actor-Critic methods, bootstrapping bias demonstrated a large negative impact on performance. Additionally for Actor-Critic methods, we showed that value functions in wider CoANs are more sensitive to additional noise in their state observations, making them more susceptible to noise with increasing depth of the CoAN.

The learning dynamics of CoANs differ significantly from traditional neural networks and have not yet been thoroughly explored. CoANs offer a rich set of opportunities for further investigation. A significant source of instability, as is for any MARL system using policy gradient updates, is the high variance of gradient estimates for each agent [Canese *et al.* 2021b; Hernandez-Leal *et al.* 2017]. Better understanding of how this property affects the addition to CoAgents to the network in width as compared to depth is a potential area for future research. Exploring behaviour in settings where the network architecture is varied such as bottle necks or recurrent networks is also an area for future work. As CoANs require only their local observations and a global reward signal, networks with heterogeneous CoAgents are possible. This opens the door to configurations of mixing agent types or interleaving layer types. Other formulations of the problem are additional avenues that could be fruitful.

Bibliography

[Aenugu *et al.* 2019] Sneha Aenugu, Abhishek Sharma, Sasikiran Yelamarthi, Hananel Hazan, Philip S. Thomas, and Robert Thijs Kozma. Reinforcement learning with spiking coagents. *ArXiv*, abs/1910.06489, 2019.

[Amari 1993] Shun-ichi Amari. Backpropagation and stochastic gradient descent method. *Neurocomputing*, 5(4-5):185–196, 1993.

[András 2018] Péter András. High-dimensional function approximation with neural networks for large volumes of data. *IEEE Transactions on Neural Networks and Learning Systems*, 29:500–508, 2018.

[Bacon *et al.* 2017] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[Baxter and Bartlett 2000] Jonathan Baxter and Peter L. Bartlett. Reinforcement Learning in POMDP's via Direct Gradient Ascent. In *ICML*, 2000.

[Bernstein *et al.* 2002] Daniel S. Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. The complexity of decentralized control of Markov decision processes. *Mathematics of Operations Research*, 27(4):819–840, Nov 2002. 01374.

[Buşoniu *et al.* 2010] Lucian Buşoniu, Robert Babuška, and Bart De Schutter. Multi-agent reinforcement learning: An overview. In *Innovations in multi-agent systems and applications-1*, pages 183–221. Springer, 2010.

[Canese *et al.* 2021a] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11), 2021.

[Canese *et al.* 2021b] Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11:4948, 2021.

[Claus and Boutilier 1998] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[Deng 2012] Li Deng. The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[Foerster *et al.* 2018] Jakob N. Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *AAAI*, 2018.

[Goodfellow *et al.* 2015] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep learning. *Nature*, 521:436–444, 2015.

[Gupta *et al.* 2021] Dhawal Gupta, Gabor Mihucz, Matthew Schlegel, James Kostas, Philip S Thomas, and Martha White. Structural credit assignment in neural networks using reinforcement learning. *Advances in Neural Information Processing Systems*, 34, 2021.

[Hernandez-Leal *et al.* 2017] Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz de Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

[Hu *et al.* 2021] Zijian Hu, Chengxiang Zhuge, and Wei Ma. Towards a very large scale traffic simulator for multi-agent reinforcement learning testbeds. *ArXiv*, abs/2105.13907, 2021.

[Iqbal and Sha 2019] Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 2961–2970. PMLR, 2019.

[Kaelbling *et al.* 1996] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996.

[Kingma and Ba 2014] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[Kostas *et al.* 2019] James Kostas, Chris Nota, and Philip S. Thomas. Asynchronous coagent networks: Stochastic networks for reinforcement learning without backpropagation or a clock. *arXiv:1902.05650 [cs, stat]*, Feb 2019. arXiv: 1902.05650.

[Lillicrap and Santoro 2019] Timothy P Lillicrap and Adam Santoro. Backpropagation through time and the brain. *Current opinion in neurobiology*, 55:82–89, 2019.

[Lowe *et al.* 2017] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Neural Information Processing Systems (NIPS)*, 2017.

[Mahadevan 1996] Sridhar Mahadevan. Average reward reinforcement learning: Foundations, algorithms, and empirical results. *Machine learning*, 22(1):159–195, 1996.

[Merkh and Montúfar 2019] Thomas Merkh and Guido Montúfar. Stochastic feedforward neural networks: Universal approximation. *ArXiv*, abs/1910.09763, 2019.

[Nguyen *et al.* 2018] Duc Thien Nguyen, Akshat Kumar, and Hoong Chuin Lau. Credit assignment for collective multiagent RL with global rewards. *Advances in neural information processing systems*, 31, 2018.

[Nota and Thomas 2019] Chris Nota and Philip S Thomas. Is the policy gradient a gradient? *arXiv preprint arXiv:1906.07073*, 2019.

[Pretorius *et al.* 2020] Arnu Pretorius, Elan Van Biljon, Benjamin van Niekerk, Ryan Eloff, Matthew Reynard, Steve James, Benjamin Rosman, Herman Kamper, and Steve Kroon. If dropout limits trainable depth, does critical initialisation still matter? a large-scale statistical analysis on ReLU networks. *Pattern Recognition Letters*, 138:95–105, 2020.

[Schoenholz *et al.* 2017] Samuel S. Schoenholz, Justin Gilmer, Surya Ganguli, and Jascha Sohl-Dickstein. Deep information propagation. *arXiv:1611.01232 [cs, stat]*, Apr 2017. arXiv: 1611.01232.

[Schulman *et al.* 2016] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. *arXiv:1506.05254 [cs]*, Jan 2016. arXiv: 1506.05254.

[Schulman *et al.* 2018] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv:1506.02438 [cs]*, Oct 2018. arXiv: 1506.02438.

[Shannon 1948] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[Sutton and Barto 2018a] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction*. Adaptive computation and machine learning series. The MIT Press, second edition edition, 2018.

[Sutton and Barto 2018b] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[Sutton *et al.* 1999a] Richard S. Sutton, David A. McAllester, Satinder Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, 1999.

[Sutton *et al.* 1999b] Richard S Sutton, Doina Precup, and Satinder Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

[Sutton *et al.* 2000] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Advances in neural information processing systems*, pages 1057–1063, 2000.

[Takeshi 2017] Daniel Takeshi. *Going Deeper Into Reinforcement Learning: Fundamentals of Policy Gradients*. https://danieltakeshi.github.io/2017/03/28/going-deeper-into-reinforcement-learning-fundamentals-of-policy-gradients/, 2017. Accessed: 2022-02-06.

[Tan 1997] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative learning. *Readings in Agents*, pages 487–494, 1997.

[Thomas and Barto 2011] Philip S Thomas and Andrew G Barto. Conjugate Markov decision processes. In *ICML*, 2011.

[Thomas and Barto 2012] Philip S Thomas and Andrew G Barto. Motor primitive discovery. In *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)*, pages 1–8. IEEE, 2012.

[Thomas 2011] Philip S. Thomas. *Policy Gradient Coagent Networks*, page 1944–1952. Curran Associates, Inc., 2011.

[Tieleman *et al.* 2012] Tijmen Tieleman, Geoffrey Hinton, et al. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[Tuyls and Weiss 2012] Karl Tuyls and Gerhard Weiss. Multiagent learning: Basics, challenges, and prospects. *AI Magazine*, 33(3):41–41, 2012.

[Weber *et al.* 2019] Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit assignment techniques in stochastic computation graphs. *arXiv:1901.01761 [cs, stat]*, Jan 2019. arXiv: 1901.01761.

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

[Williams 2004] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004.

[Xiao *et al.* 2017] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*, 2017.

[Zhang *et al.* 2019a] Huichu Zhang, Siyuan Feng, Chang Liu, Yaoyao Ding, Yichen Zhu, Zihan Zhou, Weinan Zhang, Yong Yu, Haiming Jin, and Zhenhui Jessie Li. Cityflow: A multi-agent reinforcement learning environment for large scale city traffic scenario. *The World Wide Web Conference*, 2019.

[Zhang *et al.* 2019b] Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *arXiv preprint arXiv:1911.10635*, 2019.

[Zhao *et al.* 2012] Tingting Zhao, Hirotaka Hachiya, Gang Niu, and Masashi Sugiyama. Analysis and improvement of policy gradient estimation. *Neural Networks*, 26:118–129, Feb 2012. 00098.

[Zini *et al.* 2020] Modjtaba Shokrian Zini, Mohammad Pedramfar, Matthew Riemer, and Miao Liu. Coagent networks revisited. *arXiv preprint arXiv:2001.10474*, 2020.

Appendices

Appendix A

Design decisions

A.1 Heterogeneous vs Homogeneous updates

Here we provide further details for using a homogeneous update for REINFORCE, algorithm 1, instead of a heterogeneous update, algorithm 4.

To note the major difference is in the update of the output layer. Algorithm 4 shows that this update is not a reinforcement learning update, but rather, effectively a backpropagation update.

For algorithm 1, as seen in figure 3.2 in the implementations of our CoANs we use isolated backpropagation to update the weights for a CoAgent. The agents select and action from a Bernoulli distribution generate by a softmax function that takes as inputs, the outputs of two linear layer nodes. In this isolated manner we can use the policy gradient loss to backpropagate error signals from the action selected by the CoAgent to the weights of the CoAgent.

For algorithm 4 the loss being used is not a policy gradient loss, but rather a traditional backpropagation loss. This is likely why it can solve classification tasks in isolation whereas the output layer of algorithm 1 cannot.

Algorithm 4 reproduction from Gupta *et al.* [2021].

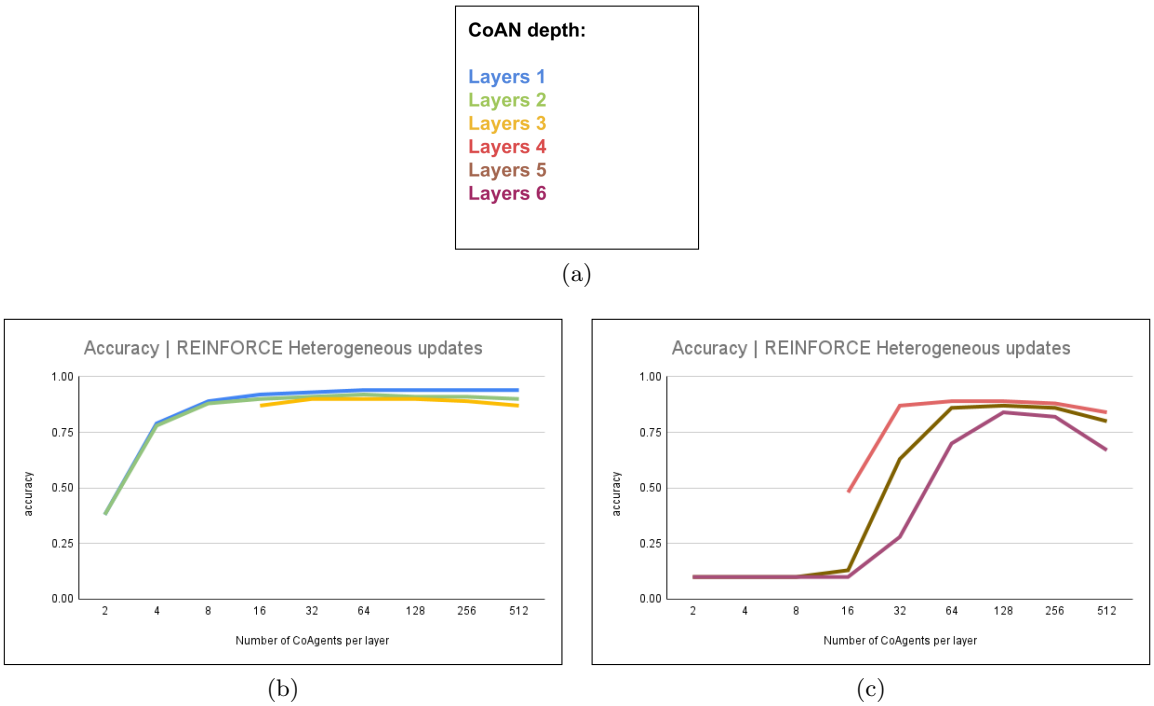
Algorithm 4 CoAN REINFORCE: Heterogeneous updates

Input: a dataset $\mathcal{D}$
Input: a policy parameterisation $\boldsymbol{\theta}_{i,j}$ for each CoAgent $\pi_{\boldsymbol{\theta}}(a|s)$
Input: a parameterisation w for an output layer $f_w(s)$
Algo param: step size $\alpha > 0$
Algo param: number of CoAgents per layer n, number of layers k
Initialise: each set of CoAgent policy parameters $\pi_{i,j} \in R^d$ and output layer parameters $w \in R^d$

while epoch is not terminal **do**
  for e episodes / samples in dataset **do**
    $x, y \sim \mathcal{D}$
    $S_0 \leftarrow x$

    for j in k **do**
      $a_j \sim \pi_\theta(S_j|A_j)$
      $s_{j+1} \leftarrow a_j$
    $a_{k+1} \leftarrow f_w(s_k)$ ▷ output layer
    $G \leftarrow R_{k+1} \leftarrow -\operatorname{err}(a_{k+1}, y)$

    for j in k **do**
      $\boldsymbol{\theta}_{j,e} \leftarrow \boldsymbol{\theta}_{j,e} + \boldsymbol{\theta}_{j,e}\, \alpha\, G\, \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}\left(A_j \mid S_j\right)$
    $w_e \leftarrow w_e + w_{e-1}\, \alpha\, \nabla_w f_w(s_k)$ ▷ update output layer



(a)

(b)

(c)

Figure A.1: Mean accuracy overview for CoANs of various depths and breadths after 500 epochs using REINFORCE algorithm 4

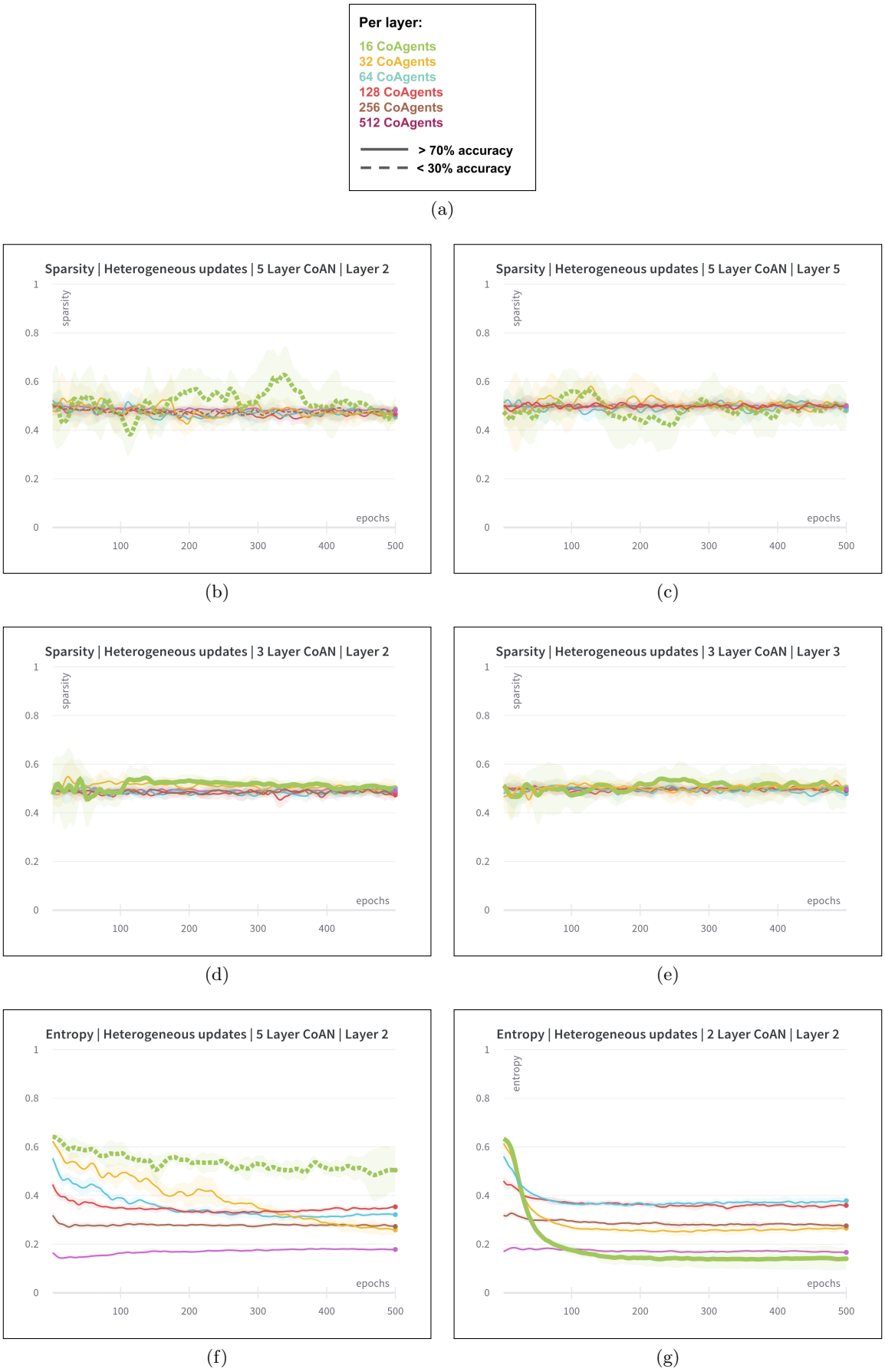


Figure A.2: Sparsity and entropy metrics for CoANs using REINFORCE algorithm using REINFORCE algorithm 4