Structural Credit Assignment in Neural Networks using Reinforcement Learning

by

Dhawal Gupta

A thesis submitted in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

University of Alberta

Abstract

Structural credit assignment in neural networks is a long-standing problem, with a variety of alternatives to backpropagation proposed to allow for local training of nodes. One of the early strategies was to treat each node as an agent and use a reinforcement learning method called REINFORCE to update each node locally with only a global reward signal. In this work, we revisit this approach and investigate if we can leverage other reinforcement learning approaches to improve learning. We first formalize training a neural network as a finite-horizon reinforcement learning problem and discuss how this facilitates using ideas from reinforcement learning like off-policy learning, exploration and planning. We first show that the standard REINFORCE approach can learn but is suboptimal due to on-policy training: each agent learns to output an activation under suboptimal action selection from the other agents. We show that we can overcome this suboptimality with an off-policy approach, that it is particularly effective with discretized actions. We provide several additional experiments, highlighting the utility of exploration, robustness to correlated samples when learning online and a study into the policy parameterization of each agent.

Preface

This thesis is a compilation and extension of a conference paper under submission to NeurIPs 2021, co-authored with Gabor Mihucz, Matthew Schlegel, James Kostas, Philip Thomas and Martha White. Gabor implemented the continual learning experiments (Section 5.2) and Monte Carlo (MC) agent in Figure 3.4. Martha contributed to parts of introduction (Chapter 1) and formalism (Chapter 2), she provided with the proof of proposition 1 and suggestions throughout. Philip and Martha helped design the local minima experiment (Section 5.1). Matthew, James and Philip helped develop ideas based off of coagent networks and variance reduction strategies in the same. Rest of the work is original contribution of Dhawal Gupta.

To Ayush, Mummy and Papa

Acknowledgements

I would like to thank my supervisor, Martha White, under whose guidance and acumen I have learned how to think like a researcher and value collaboration. She has supported my ideas throughout the journey and given me the flexibility to pursue unconventional directions. Martha has the ability and insight to declutter noise around research and focus on things that hold value and appreciate its possible impact. She has been a fantastic mentor.

The master's program has been one of the most fulfilling experiences in my life; the environment provided by RLAI, AMII, and in general, Canada is conducive for research and makes the whole process much more joyful.

The experience would not have been possible without my friends throughout these two years; I am especially indebted to Anmol, Alex, Katyani, Mayank, Tanisha, Ashish, Alan, Sina, Andy, Khurram, Abhishek, Michael, Matt, Shivam, Navya & Maithrreye for being part of this journey. Special thanks to Csaba for being a great mentor, friend, runner and fellow rock climber. Also, thanks to my undergrad collegiate (Infinity), who have stuck with me till now.

Lastly, everything I have accomplished and will accomplish would not have been possible without the unconditional love and support that my parents and younger brother have shown me; I am blessed to have such an amazing family.

Contents

List of Figures

Chapter 1

Introduction

Training neural networks involves *structural credit assignment*: attributing credit (or blame) to nodes in the network for correct (or incorrect) predictions. The output from a node early in the network impacts all the outputs downstream and, finally, the prediction outputted at the end of the network. Our goal is to adjust the weights that produced the output for this node, so that the prediction would have been more accurate. The most widely used solution for the structural credit assignment problem is backpropagation (Rumelhart et al., 1986), namely gradient descent on the loss for the outputs.

Moving beyond backprop provides more flexibility in training neural networks. Backprop requires differentiability of activations and losses for the network, as well as synchronicity for computing the gradient and updating the weights. To update a node internal to the network, a full feedforward and backward pass needs to be computed, with global gradient information sweeping backwards from the output. Ideally, for online agents operating real-time, with computational constraints, we would have nodes that update each step, locally and asynchronously.

To make progress towards this lofty goal, we revisit an old idea: treating each node as an agent. Work in reinforcement learning (RL), including ideas like eligibility traces, were in fact inspired by Klopf (1982) and the hedonistic neuron. It is not surprising that the idea of using an RL agent for each node is found in early work, including the original REINFORCE algorithm (Williams, 1992), which is a policy gradient approach using sampled returns.

Most work treating each node as an agent uses the REINFORCE update, often with baselines for variance reduction, including work on learning with spiking neurons (Fiete and Seung, 2006) and CoAgent Networks (CoANs) (Kostas et al., 2020; Thomas, 2011; Thomas and Barto, 2011) which extended prior work by Barto (1985) in the area of *learning automata* (Narendra and Thathachar, 1989). The work in CoANs 1) nicely formalizes the idea of a collection of agents—each agent corresponding to a node or subset of nodes—cooperating to maximize return and 2) provides a general theorem on the validity of using the REINFORCE update. For this reason, we adopt their terminology and use CoANs to refer to networks composed of agents.

More recently, other algorithmic ideas from reinforcement learning, beyond REINFORCE, have begun to affect training of (stochastic) neural networks. The ideas of critics and baselines, which reduce the variance of policy gradient updates, have been well-developed for stochastic computation graphs (Weber et al., 2019). This work provides a unification of gradient derivations, but as yet not an investigation into practical algorithms for structural credit assignment in neural networks. Other work on learning under stochastic neurons has typically used REINFORCE as a basic method, and explored other heuristics to improve learning, such as straight-through estimators (Y. Bengio et al., 2013), rather than improved RL approaches. Work on credit assignment is loosely inspired by the idea of bootstrapping in RL, including synthetic gradients (Jaderberg et al., 2017; Lansdell et al., 2020) and fixed-point propagation (Nath et al., 2020). Works that were not motivated by bootstrapping used similar ideas of explicitly optimizing activation vectors, such as target propagation (Y. Bengio, 2014; Y. Bengio et al., 2016; Lee et al., 2015) and auxiliary variable methods (Carreira-Perpinan and Wang, 2014)

Overall, however, the broader space of RL algorithms has not been leveraged to learn CoANs. These include advances in policy gradient methods, ideas from off-policy learning, exploration approaches and planning approaches. One reason for this omission could be that the structural credit assignment problem within the neural network has not been clearly defined as an RL problem; rather, it was simply intuitive to use REINFORCE approaches for each

node. Even the theory from the original CoANs work focused on the return in the environment—since CoANs were used to solve a reinforcement learning problem—and did not explicitly formalize the structural credit assignment problem within the network. Another reason could be that many straightforward ideas are not effective, as we show in this work.

To facilitate the use of RL algorithms, we first formalize the structural credit assignment problem as a finite horizon RL problem. We show local policy gradient updates provide an unbiased estimate of the joint gradient for structural credit assignment, ensuring REINFORCE is a sound approach. We then discuss key ideas from RL—namely exploration, off-policy learning and planning—that can be leveraged to improve learning in CoANs. In particular, we discuss how these methods can leverage some structural information—as might be the case if the agent has some model information or prior information about the environment—while still using local updates. We show that REINFORCE can train multi-layered networks, but faces issues with suboptimality due to coagents learning under nondeterminism of fellow coagents. We provide an in-depth study highlighting this problem and measuring the entropy of different parts of the network. This in-depth study motivates the difficulties in using the common on-policy approaches, and we discuss and show how off-policy learning is a more promising direction.

Finally, we discuss the advantages of CoANs which include active exploration and learning in non-standard settings. When moving away from the standard iid learning setting, we show CoAN's can perform better than backprop on a continual learning problem with a highly correlated dataset.

1.1 Related Work

The literature on approaches to structural credit assignment is vast, with much of it using ideas different from reinforcement learning. One category of approaches uses local updates to make activations similar to a target vector of activations, such as target propagation (Y. Bengio, 2014; Lee et al., 2015), the method of auxiliary variables (Carreira-Perpinan and Wang, 2014) and fixed

point propagation (Nath et al., 2020). Kickback approximates the backprop update for ReLU networks, using an approximation to the gradient that allows for local updates (Balduzzi et al., 2015). Feedback alignment (Lillicrap et al., 2016; Nøkland, 2016) involves using random weights, instead of the actual weights in the next layer, that avoids symmetric propagation that is thought to be biologically implausible.

Weight perturbation and node perturbation approaches have been used to estimate gradients, with node perturbations emerging as the preferred approach (Saito et al., 2011). Fiete and Seung (2006) showed that their node perturbation algorithm actually includes REINFORCE as a special case, linking these two classes of methods. However, the connection only exists for Gaussian noise perturbations and REINFORCE; for other perturbations, the connection is lost, as well as for other RL algorithms. Another form of perturbation methods involve representation search for systematic generation and testing of features (Mahmood and Sutton, 2013). This approach takes advantage of stochasticity in selecting features with a backprop like fine-tuning algorithm which is beneficial in non-stationary settings.

Other work has focused on changes in weights across time. Spiking neural networks and the associated spike-timing-dependent plasticity (STDP) learning rule (Markram et al., 2012) adjusts weights based on the relative timing of activations for nearby nodes. Equilibrium propagation (Scellier and Bengio, 2017) involves a phase of propagation in the network until reaching a low energy state, followed by a learning update. This work showed similarities to STDP, Contrastive Hebbian Learning (Movellan, 1991) and Contrastive Divergence (Hinton, 2002). Reinforcement learning updates have been used for spiking neural networks, called Reward-modulated STDP. These updates use a global reward but with local update rules (Legenstein et al., 2010; Legenstein et al., 2008), with some interesting insights that perturbations can be beneficial to induce exploration (Legenstein et al., 2010). This area has focused on delayed reward, namely assigning credit from node changes across time and across multiple updates. The local updates use node perturbation with eligibility traces to link perturbations on this step, to accuracy (rewards)

at later time steps (Legenstein et al., 2010; Miconi, 2017). A more recent algorithm, called cross propagation (Veeriah et al., 2017), explicitly adjusts weights back-in-time, to account for accuracy on this step, similarly to some meta-learning strategies but completely online.

There has also been some work using REINFORCE to learn activation paths through a network (Denoyer and Gallinari, 2014) and learning when to activate parts of the network (E. Bengio et al., 2015; Y. Bengio et al., 2013). Other work has connected structural and temporal credit assignment, but in the opposite direction from this work: specifying temporal credit assignment as a structural credit assignment problem (Agogino and Tumer, 2004).

Once we use RL agents as nodes, which have stochastic policies, there is a clear connection to the work on stochastic neural networks. Much of this work has looked at networks with stochastic binary activations (Y. Bengio et al., 2013; Merkh and Montúfar, 2019; Neal, 1990; Raiko et al., 2015), though the wider literature on stochastic computation graphs encompasses a broad range of stochastic neural networks (Schulman et al., 2015; Tang and Salakhutdinov, 2013; Weber et al., 2019). Early work considered an EM-style algorithm (Tang and Salakhutdinov, 2013) and a simple alternative, called the straight-through estimator (Y. Bengio et al., 2013), that directly passes the gradient back through the node to the weights that created the activation. The straight-through estimator has recently been shown to be a valid estimator for stochastic binary networks (Shekhovtsov and Yanush, 2021), and a lower-variance update has been proposed (Gu et al., 2016).

Multi-agent reinforcement learning approaches tackle a similar problem, in the cooperative setting. The coagents in the neural network can be seen as a group of agents cooperating to produce accurate predictions, though with the notable difference that there is an ordering to the actions taken by coagents. A common approach in this area has been to use reinforcement learning agents, and consider mechanisms for coordination without centralization. Some approaches have been to carefully define rewards for each agent (Wolpert and Field, 2002; Wolpert et al., 1999); to use a single global reward plus some noise (Chang et al., 2003); to use independent Q-learners (Tan, 1993); or to estimate

a global critic (Oliehoek et al., 2008) potentially with local policy updates (Foerster et al., 2018; Rashid et al., 2018). When using independent Q-learners for each agent, the other agents are treated as part of the environment; consequently, the environment appears non-stationary. Hyper Q-learning (Tesauro, 2003) reduces the impact of this non-stationarity by estimating other agent's policies. These strategies do not directly extend to CoANs, but the connection to the cooperative multi-agent reinforcement learning problem could provide fruitful avenues for improved algorithms.

Finally, there have been several empirical studies on alternative update strategies and architectures. Spiking neural networks generally have lower accuracy than standard deep neural networks, but a recent study has shown that with advances in hardware and algorithms to train spiking neural networks, this gap has become smaller (Tavanaei et al., 2019). Stochastic neural networks, particularly with binary activations, are challenging to train. Still, with some improvements to the gradient estimator, they can have significant advantages, including providing a level of regularization (Raiko et al., 2015). In this work, we aim to provide a more comprehensive empirical investigation into using reinforcement learning approaches to train CoANs.

1.2 Contributions and Outline

The thesis is structured as follows with the following contributions:

- Formalizing the credit assignment in a neural network as finite horizon reinforcement learning problem, allowing for easy application of RL algorithms (Chapter 2).

- Providing evidence of suboptimality of the current methods, including REINFORCE for such networks particularly due to nondeterminism in fellow agents of the networks and failure of on-policy learning to solve the same. (Chapter 3)

- Introducing the use of critics and off-policy learning algorithms by leveraging the finite horizon formalism. Followed by providing an algorithm

and experiments to present the potential benefits of off-policy learning in providing a way to deal with nondeterminism of surrounding coagents. (Chapter 4)

- Demonstrating that CoAN's can offer benefit over backpropagation methods in non standard problem settings. These include the possibility of active exploration as well as dealing with continual learning problems. (Chapter 5)

Chapter 2

Structural Credit Assignment as a Finite Horizon Reinforcement Learning Problem

This chapter describes how to formalize the credit assignment problem in a feedforward neural network as a finite-horizon RL problem. The basic idea for the finite horizon formulation is that the horizon is the number of layers. On each step, the input state for the agent is the activations from the previous step, and the output is the activations for this layer or the final output prediction.

2.1 Formalism

Here we introduce the finite horizon formalism for structural credit assignment in neural networks. We start in the simplest setting, where we have a fully connected feedforward neural network composed of k layers of size n hidden units per layer. We assume we have inputs $\boldsymbol{x} \in \mathcal{X}$, prediction targets $y \in \mathcal{Y}$, hidden layer activations $\boldsymbol{h}_j = f(\boldsymbol{z}_j)$ for pre-activations $\boldsymbol{z}_j$ with activation function f. Here, err(.) refers to the error function used for supervised learning problems, for example, mean squared loss for regression or cross-entropy loss for classification.

Consider the following agent-environment interaction. On the first step, given the sampled input $\boldsymbol{x} \in \mathcal{X}$, the initial observation $\boldsymbol{o}_0 = [\boldsymbol{x}, 0]$ where the 0 indicates the step-index in the finite horizon problem. The agent observes input $\boldsymbol{o}_0$, takes action $\boldsymbol{a}_0 \in \mathcal{A}$ that is a vector of activations $\boldsymbol{h}_1$ (or pre-

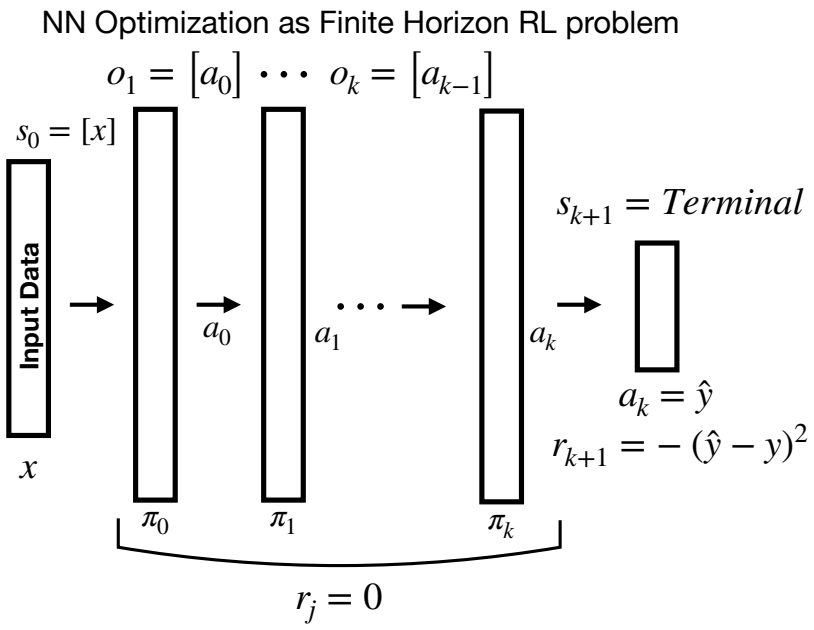

Figure 2.1: The problem described as a finite horizon RL problem, where each intermediate step produces a reward of 0, and the end step produces a reward equal to negative of the loss function.

activations $\boldsymbol{z}_1$), and the next observation is deterministically $\boldsymbol{o}_1 = [\boldsymbol{a}_0, 1] = [\boldsymbol{h}_1, 1]$ (or $\boldsymbol{o}_1 = [f(\boldsymbol{a}_0), 1] = [f(\boldsymbol{z}_1), 1]$). Then the agent inputs $\boldsymbol{o}_1$ and outputs the activations for the next layer $\boldsymbol{a}_1$ and obtains next observation $\boldsymbol{o}_2 = [\boldsymbol{a}_1, 2]$ (or $\boldsymbol{o}_2 = [f(\boldsymbol{a}_1), 2]$). This transition is Markov, because given $\boldsymbol{o}_1$ and $\boldsymbol{a}_1$, the next observation $\boldsymbol{o}_2$ is independent of $\boldsymbol{o}_0$. This interaction continues for k steps, terminating at the last layer when the final action is to output the prediction $\hat{y}$.

The majority of this process is Markov, and only becomes partially observable due to the reward given at the end of the episode. The rewards received during the episode are zero, with a reward given upon termination that is a function of the error between prediction and target, such as $\text{err}(\hat{y}, y) = (\hat{y} - y)^2$ with reward $-\text{err}(\hat{y}, y)$. The probability of y depends on $\boldsymbol{x}$, and so the reward depends on $\boldsymbol{x}$ on this last step. We can obtain Markov states by simply including $\boldsymbol{x}$ in each observation. But, because the policies are not given the ability to use $\boldsymbol{x}$, it is more accurate to model this problem as a partially observable one. Formally, the partially observable finite-horizon undiscounted MDP consists

of state space $\mathcal{S} = \mathcal{X} \times \{0\} \cup \mathcal{X} \times \mathbb{R}^n \times \{1, \ldots, k-1\}$, actions $\mathcal{A} = \mathbb{R}^n$ and observation function $\boldsymbol{o}(s) = s$ if s has index 0 and otherwise $\boldsymbol{o}(s) = [\boldsymbol{h}, j]$ for $s = [\boldsymbol{x}, \boldsymbol{h}, j]$. Remember that the integer values are included to indicate the layer number of the agents processing the input and hence we include $\{0\}$ and other integers in the state space.

A typical RL agent learns a separate policy for each horizon. This corresponds to learning one stationary policy because the step-index is included in the state. Practically, however, these (stochastic) policies $\pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j)$ for $j \in \{0, \ldots, k\}$ are updated separately, without considering the single stationary policy. If the agent uses policy gradients, it uses parameterized distributions for the policies, such as Gaussians with means parameterized by the activations from the last layer. These policies can be updated using a REINFORCE update, using the gradient of the CoAN objective.

The CoAN objective corresponds to a policy gradient objective. An episode trajectory, with index information implicit and assuming actions are activations, looks like:

$$\boldsymbol{o}_0 = \boldsymbol{x}, \boldsymbol{a}_0, r_1 = 0, \boldsymbol{o}_1 = \boldsymbol{a}_0, \ldots, \boldsymbol{o}_k = \boldsymbol{a}_{k-1}, \boldsymbol{a}_k = \hat{y}, r_{k+1} = -\text{err}(\hat{y}, y), \text{termination}$$

Because $\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k|\boldsymbol{x}) = \pi(\boldsymbol{a}_0|\boldsymbol{x}) \prod_{j=1}^{k} \pi_j(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})$, the probability of this trajectory is $p(\boldsymbol{x})\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k|\boldsymbol{x})p(y|\boldsymbol{x})$. This is because $p(\boldsymbol{x})$ gives the probability of the start state; $\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k|\boldsymbol{x})$ fully defines the probability of the trajectory because the state outcomes are deterministic given the action (activation or pre-activation); and $p(y|\boldsymbol{x})$ defines the probability of the reward on termination, since the target is stochastic. This provides a straightforward policy gradient objective, for undiscounted return $G \stackrel{\text{def}}{=} -\text{err}(a_k, y)$.

The policy gradient objective is defined as

$$\begin{aligned} J(\boldsymbol{\theta}) &\stackrel{\text{def}}{=} \int p(\boldsymbol{x})\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_k|\boldsymbol{x})p(y|\boldsymbol{x})G d\boldsymbol{x} d\boldsymbol{a}_0 \ldots d\boldsymbol{a}_k dy \\ &= \mathbb{E}_{\pi_{\boldsymbol{\theta}}, p(\boldsymbol{x}, y)}[G] \end{aligned}$$

where each policy π_j has parameters $\boldsymbol{\theta}_j$ and $\boldsymbol{\theta} = [\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_k]$. The stochastic gradient of this objective separates out into the following stochastic gradi-

ents for each policy separately: $G\nabla_{\boldsymbol{\theta}_j} \log \pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j)$. We show this formally in Proposition 1.

Proposition 1 (Policy Gradient Theorem for Structural Credit Assignment in CoANs)**.**

$$\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}, p(\boldsymbol{x},y)}\big[G\nabla_{\boldsymbol{\theta}_j} \ln \pi_j(\boldsymbol{A}_j|\,\boldsymbol{O}_j)\big]$$

Proof. For a given trajectory and its return G, we can notice that them sampled gradient of the policy objective can be written as:

$$G\nabla\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1}, \hat{y}|\boldsymbol{x}) =$$
$$G\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1}, \hat{y}|\boldsymbol{x})\nabla \ln \pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1}, \hat{y}|\boldsymbol{x})]$$

Further we can write the derivative log term as:

$$\begin{aligned}
\nabla_{\boldsymbol{\theta}} \ln(\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{k-1}, \hat{y}|\boldsymbol{x})) &= \nabla_{\boldsymbol{\theta}} \ln(\pi(\boldsymbol{a}_0|x)\pi(\boldsymbol{a}_1|\boldsymbol{a}_0) \ldots \pi(\hat{y}|\boldsymbol{a}_{k-1})) \\
&= \nabla_{\boldsymbol{\theta}}(\ln \pi(\boldsymbol{a}_0|x) + \ln \pi(\boldsymbol{a}_1|\boldsymbol{a}_0) + \ldots + \ln \pi(\hat{y}|\boldsymbol{a}_{k-1})) \\
&= \sum_{j=0}^{k} \nabla_{\boldsymbol{\theta}} \ln \pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})
\end{aligned}$$

Note: $\boldsymbol{a}_{-1} = \boldsymbol{x}$ is the input and the output is $\boldsymbol{a}_k = \hat{y}$. The final expression is a summation over the gradient of log probability of activations from each layer. The activations of a layer only depends on the policy of the coagents in that layer of the network. The policy for agents in each layer don't share any parameters for coagents in other layer (in fact for linear coagents, each coagent in a layer also doesn't share any parameter with any other coagent in that layer.) Hence we can separate out the gradients for the policy in each

layer locally to depend only on the parameters of that layer i.e.:

$$\begin{aligned}\sum_{j=0}^{k} \nabla_{\boldsymbol{\theta}} \ln \pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1}) &= \sum_{j=0}^{k} \nabla_{\boldsymbol{\theta}} \ln \pi_j(\boldsymbol{a}_j|\boldsymbol{a}_{j-1}) \\ &= \sum_{j=0}^{k} \nabla_{\boldsymbol{\theta}_j} \ln \pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1}) \\ &= \begin{bmatrix} \nabla_{\boldsymbol{\theta}_0} \ln \pi_0(\boldsymbol{a}_0|\boldsymbol{x}) \\ \nabla_{\boldsymbol{\theta}_1} \ln \pi_1(\boldsymbol{a}_1|\boldsymbol{a}_0) \\ \vdots \\ \nabla_{\boldsymbol{\theta}_k} \ln \pi_k(\hat{y}|\boldsymbol{a}_{k-1}) \end{bmatrix}\end{aligned}$$

Note : $\nabla_{\boldsymbol{\theta}_j} \ln \pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})$ corresponds to vector of gradients for $\nabla_{\boldsymbol{\theta}_j} \ln \pi_j(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})$ but padded with 0's for other coagents ($\neq j$).

The last part comes from the summation over different groups of parameters and hence the total vector of gradients looks like a vector of separate gradients.

Hence taking the gradient of the PG objective w.r.t. to the parameters of specific coagent separate outs into the gradient of the parameters of that specific coagent only.

□

A similar result has been shown for CoANs used in the RL setting (Kostas et al., 2020, Theorem 1) and for stochastic computation graphs (Weber et al., 2019, Theorem 2); we include the result specifically for this case because it avoids much of the complications from those other works.

The locality of the policy gradient means policies can be updated locally with their own gradients, with a shared global return signal. We can also easily incorporate different control variates to reduce the variance of the gradient, called baselines. The update is $(G - V(\boldsymbol{o}_j))\nabla_{\boldsymbol{\theta}_j} \log \pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j)$, where the learned baseline $V(\boldsymbol{o}_j)$ estimates expected value for a given input across all actions, meaning $G - V(\boldsymbol{o}_j)$ corresponds to the advantage for the actions selected. Subtracting a baseline as above which doesn't depend on the current action doesn't introduce any bias in our gradient estimate. Several different critics have been proposed for stochastic computation graphs (Weber et al., 2019);

we discuss how to make similar baselines for this finite horizon problem in section 3.1.2.

However, the local policy gradient update is limited in that it can make it difficult to learn nearly deterministic coagents. The variance in the policy gradient update is composed of two components: the variance of the other coagents and environmental. In the next chapter, we show that the culprit for this issue is the non-determinism in other coagents contrary to the standard issue in RL algorithms, where the variance arises from the environment. This motivates the use of off-policy methods, that allow each agent to reason about the greedy policy of other coagents.

Remark: We developed the formalism for a simple setting, to facilitate understanding. However, all the ideas extend to more generic acyclic networks because every acyclic network has a topological ordering on nodes. The state input for each coagent still consists of the input nodes. At each time step, whichever coagents have all their nodes evaluated—namely, have their state input available—can produce their action. This propagates forward until a prediction is produced. As before, this process is only partially observable because the final reward depends on the input $\boldsymbol{x}$.

Also the above formalism talks about linear coagents specifically as that settings offers a direct comparison to the backprop setting. But in general the coagents can be composed of any function approximator. For example each node can be a neural network in itself.

Chapter 3

Issues with Nondeterminism in CoANs

This chapter explores that though CoANs with REINFORCE can learn, they are hindered by nondeterminism in other coagents. We investigate a variety of variance reduction approaches, and find that learning plateaus at a suboptimal point regardless. Next, we investigate more deeply and find that the coagents learn suboptimal action selection due to other coagents outputting stochastic actions. We switch to the discrete coagents, which allows a more principled approach to observe the effect of reducing nondeterminism in the on-policy setting.

3.1 Continuous Nodes

We investigate CoANs on problems where backprop is known to perform well, define baselines, and facilitate understanding the behavior and potential issues when learning in CoANs. We expect backprop to outperform CoANs, and ask: how much worse are CoANs compared to backprop, and can we close the gap with simple variance reduction techniques? To investigate this question we use two well-studied datasets: MNIST (LeCun and Cortes, 2010) for classifying handwritten digits, and the Boston Housing Dataset from UCI [1] for determining the price of a house.

Where possible, we matched the architecture and optimization choices for

[1] https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

backprop and the CoAN learner. We test both strategies using a single and double-layer neural network, with 64 hidden nodes and ReLU activations.

3.1.1 Experimental Details

Each node in the CoAN is a single coagent using a Gaussian distribution with parameterized mean and a fixed standard deviation, set system-wide through a systematic sweep over $\sigma \in \{0.1, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0\}$. Each coagent is a linear agent in its input, where the weights of the linear function are initialized using default settings. The feedforward procedure samples actions from these policies and then applies the layer's activation function (i.e., ReLU for hidden layers or the identity/softmax function for the output layer). Both use RMSProp (Tieleman and Hinton, 2012), with fixed $\beta = 0.99$ and stepsizes swept for $\alpha \in \{2^{-7}, 2^{-9}, 2^{-11} \ldots, 2^{-15}\}$. We use mini-batch gradient descent with batch size 32 for MNIST with 50 epochs, and full gradient descent for the Boston Housing Dataset with 10K epochs.

Hyperparameters are chosen from performance on a validation set held out from the training set: 10K for MNIST and 51 samples for Boston Housing. Results are averaged over ten independent runs and compared using area under curve (AUC). Agents are evaluated in terms of the accuracy achieved in classification and squared error loss for regression on the greedy policy learned by each coagent. Greedy policy implies that, at the time of evaluation the forward inferencing becomes a deterministic function, where the forward pass comprises of using the greedy policy, which corresponds to using the mean of the Gaussian as action of the coagent.

3.1.2 Using Baselines to Lower Variance

In this section we introduce three baselines for CoAN's and test their effectiveness alongside the REINFORCE update.

There has been quite a lot of development on variance reduction approaches for stochastic computational graphs (SCGs) (Weber et al., 2019). Specifically, the work around SCGs has been in optimizing NNs with stochastic nodes,

which precludes a straightforward application of backpropagation. The algorithms developed for SCGs often take advantage of global updates which it is possible to reparameterize, or by computing baselines from global information.

We can build on this to more explicitly take advantage of the fact that we are tackling a particular finite horizon problem. The typical strategies to reduce variance in the return are to (1) incorporate a baseline and (2) to estimate the expected returns with a critic. The role of the baseline is to reduce the variance due to the action selected. Typically, the baseline is chosen to be the value from the state, averaged across all possible actions. If $G - V(s) > 0$, then the selected action was better than the average, and the update increases the probability of that action; otherwise, it decreases it.

The baseline for the finite horizon setting is slightly different, as a different policy is used for each step. Let $\boldsymbol{a}$ consist of all actions outputted by the agents, and $Q(\boldsymbol{x}, \boldsymbol{a}) = \mathbb{E}[-\text{err}(\hat{y}, Y)|\mathbf{X} = \boldsymbol{x}, \mathbf{A} = \boldsymbol{a}]$ the expected return for that episode under those actions for the given input $\boldsymbol{x}$. Here we define $\boldsymbol{a}_{-j}$ as the set of all actions from all the layers excluding that of layer j. Ideally the baseline would consist of $B^j(\boldsymbol{x}, \boldsymbol{a}_{-j}) = \sum_{\boldsymbol{a}_j} \pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1}) Q(\boldsymbol{x}, [\boldsymbol{a}_{-j}, \boldsymbol{a}_j])$, the expected value for policy j, assuming the other layers chose the same actions. policies had fixed values. Then we can use $G - B^j(\boldsymbol{x}, \boldsymbol{a}_{-j})$ for the update to π_j, where if this quantity is greater than zero the action probability is increased and otherwise its decreased.

Once we have $Q(\boldsymbol{x}, \boldsymbol{a})$, we can also consider using it more directly. Notice that these action-values model the loss function for the optimization, but now more directly in the activations of the neural network. The policies are effectively searching this high-dimensional space, that likely has many local maxima. This is much lower variance than only using the return, as the agent can directly reason about all actions' outcomes jointly and remove variance due to variance in the targets. However, as with all action-value critic methods, this approach can be high bias if Q is inaccurate.

However, there is a more fundamental problem for us, as in decentralized learning, we cannot use $Q(\boldsymbol{x}, \boldsymbol{a})$ or the baseline $B^j(\boldsymbol{x}, \boldsymbol{a}_{-j})$. Instead, each agent can only use local information or at most a scalar global value like the

return that can be easily broadcast to all nodes. Going the other extreme from full joint action-values, we could use only a single global baseline $V(\boldsymbol{x})$ that reflects the average error for this input. All policies then use the update $G-V(\boldsymbol{x})$, without specifically considering the role of their action. This baseline essentially just centers the updates.

Such a baseline, however, is a minimal optimization improvement. Instead, we can also learn local baselines. For each policy π_j, we can learn a simple linear baseline $V_j(\boldsymbol{o}_j) = \langle \boldsymbol{o}_j, {\boldsymbol{\theta}^{(v)}}_j \rangle$ using Monte Carlo.

$$ {\boldsymbol{\theta}^{(v)}}_j \leftarrow {\boldsymbol{\theta}^{(v)}}_j + \alpha(G - \langle \boldsymbol{o}_j, {\boldsymbol{\theta}^{(v)}}_j \rangle)\boldsymbol{o}_j $$

We can also allow local communication, where nearby layers communicate small amounts of information, like their predicted values. In this case, we use temporal difference (TD) updates bootstrapping on values in the next layer.

In our experiments, we test using a baseline learned with Monte Carlo, namely with the above updates. When only using a baseline, the updates are of the form

$$ (G - V_j(\boldsymbol{o}_j))\nabla_{\boldsymbol{\theta}_j} \ln \pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j) $$

We use the following three simple baselines.

- **Global baseline** (Co-G): maintains a scalar running average $\hat{G}$ of the global loss function parameterized by η the rate of decay (fixed at 0.99).

$$ \hat{G}_t = \eta\hat{G}_{t-1} + (1-\eta)G_t $$

- **State Global baseline** (Co-SG): learns a parameterized value function associated with each input / state to the complete network and tries to predict the loss i.e. $V(\boldsymbol{x})$.

$$ {\boldsymbol{\theta}^{(v)}}_t \leftarrow {\boldsymbol{\theta}^{(v)}}_{t-1} + \alpha(G_t - V(\boldsymbol{x}))\nabla_{{\boldsymbol{\theta}^{(v)}}_{t-1}} V(\boldsymbol{x}) $$

- **State Layer baseline** (Co-SL): is a per layer baseline, where we learn a parameterized value function for each layer based on its input learned using Monte Carlo updates from the global loss, i.e. $V_j(\boldsymbol{o}_j)$.

$$ {\boldsymbol{\theta}^{(v)}}_{j,t} \leftarrow {\boldsymbol{\theta}^{(v)}}_{j,t-1} + \alpha(G_t - V_j(\boldsymbol{o}_j))\nabla_{{\boldsymbol{\theta}^{(v)}}_{j,t-1}} V_j(\boldsymbol{o}_j) $$

Algorithm 1 provides the pseudo code for REINFORCE based method with a layer wise state based baseline as an example. The other variants can be easily derived depending on the locality of baseline, i.e. global baseline would only need the running average of the global loss, whereas global state baseline would require to estimate a value function for the global input.

Algorithm 1 Coagent with layer state value function for classification

Input: Number of coagents per layer n, number of layers k, feature size d and $\mathcal{D}$ be the dataset distribution. Number of epochs e, batch size b, step size α.

Initialize: Gaussian policy for all $n \times k$ coagents, with linear function approximation for $\mu_{i,j}$ and a fixed σ, parameterized by $\boldsymbol{\theta}_{i,j}$.

Initlialize: Final layer (of size $n \times c$) with c outputs corresponding to number of classes for task, called as f.

Initlialize: $k+1$ Value functions $(v_0, \dots v_k)$, one for each layer, mostly a linear function, parameterized by ${\boldsymbol{\theta}^{(\boldsymbol{v})}}_i$

repeat

  $\boldsymbol{x}, y \sim \mathcal{D}$ batch of size b

  $\boldsymbol{o}_0 = \boldsymbol{x}$

  `// Forward Pass`

  for $i = 0$ **to** $k-1$ **do**

    for $j = 0$ **to** $n-1$ **do**

      `// Gaussian policy for each coagent`

      $\pi_{i,j} = \mathcal{N}(\mu_{i,j}(.), \sigma)$

      $a_{i,j} \sim \pi_{i,j}(\boldsymbol{o}_i)$

      $u_{i,j} = \text{ReLU}(a_{i,j})$

    end for

    `// Build next coagent layer state`

    $\boldsymbol{o}_{i+1} = (u_{i,0}, \dots, u_{i,n-1})$

  end for

  $\pi_k = \mathcal{N}(., \sigma)$

  $\hat{y} \sim \pi_k(f(\boldsymbol{o}_k)))$

  $\hat{y} = \text{softmax}(\hat{y})$

  $G = -\text{CrossEntropy}(y, \hat{y})$

  `// Backward Pass`

  for $i = k$ **to** 0 **do**

    for $j = 0$ **to** $n-1$ **do**

      `// Update coagents`

      $g_{i,j} = (G - v_i(\boldsymbol{o}_i)) \nabla_{\boldsymbol{\theta}_{i,j}} \log(\pi_{i,j}(a_{i,j} | \boldsymbol{o}_i, \sigma))$

      $\boldsymbol{\theta}_{i,j} = \boldsymbol{\theta}_{i,j} + \alpha g_{i,j}$

    end for

    `// Update baselines`

    $g_i^v = (G - v_i(\boldsymbol{o}_i)) \nabla_{{\boldsymbol{\theta}^{(\boldsymbol{v})}}_i} v_i(\boldsymbol{o}_i)$

    ${\boldsymbol{\theta}^{(\boldsymbol{v})}}_i = {\boldsymbol{\theta}^{(\boldsymbol{v})}}_i + \alpha g_i^v$

  end for

until Number of e epochs achieved

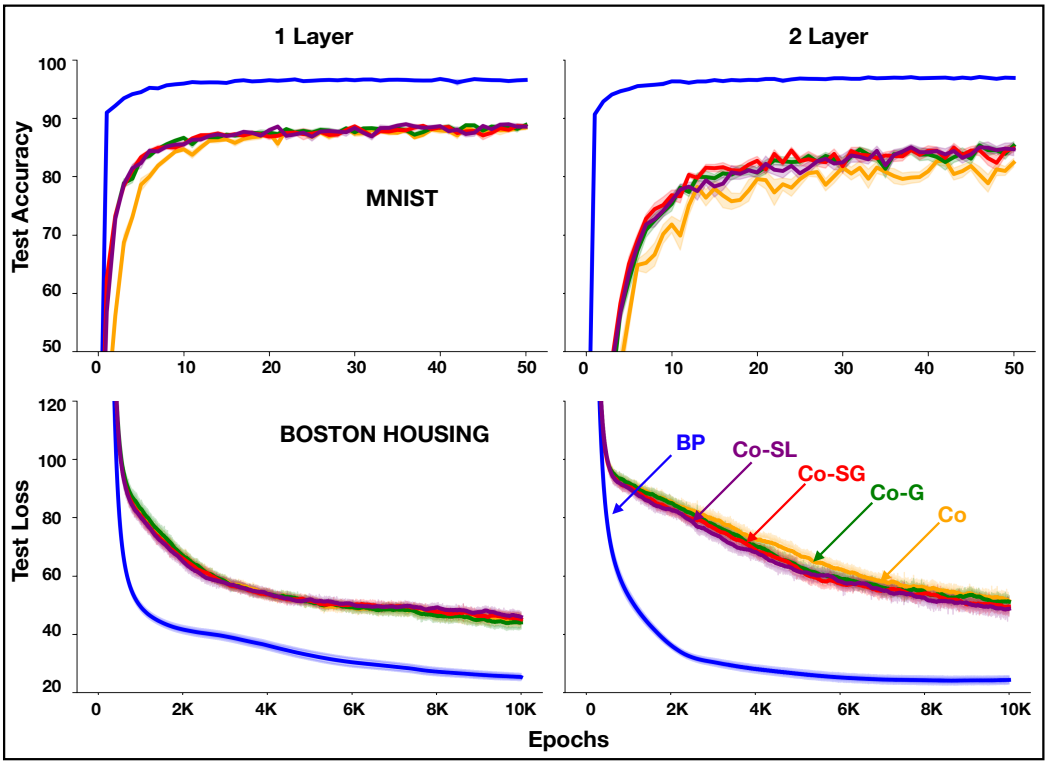

Figure 3.1: All curves are averaged for 10 runs with standard error bars. Training different variants of CoAN's and backprop on MNIST and Boston Housing with one and two layer NNs.

3.1.3 Results with Variance Reduction

Figure 3.1 reports the performance of the best performing parameters on a held-out test set with the same size as the validation set. At a glance, it is clear that backprop outperforms all the variants of CoANs. We can also observe that having a baseline can help in making learning faster, particularly in the case of MNIST. The baseline seems to have greater effect in two layer networks, where the one without a baseline seems to have really high variance. However, surprisingly, there is no improvement using the local baselines as compared to the global baseline, as can be observed from similar performance of Co-G, Co-SG and Co-SL. This experiment also shows that subtracting a baseline doesn't introduce a bias, as we see all the coagents level off at similar performance. A natural questions which arises from these results is, why does there exist a performance gap between CoAN's and backprop. Is the gap due to the optimization process or a poor stationary point of the CoAN itself. Experiments running the CoAN for longer duration showed only slight improvement, but never seem to catch on to backprop. Before this experiment, we hypothesized that high variance updates would negatively impact the CoAN optimization, but the variance reduction schemes used above did not improve performance at all, and instead levels of at similar performance.

3.2 Controlled Nondeterminism Experiments

To investigate further, we initialized a CoAN at the backprop solution, to examine if it moves away from it. If so, it would suggest the joint solution among the coagents is suboptimal, and causes this gap. We pretrain a two-layer neural network using backprop (up to 5K epochs) and then switch to the CoAN algorithm. We use the best hyperparameters obtained in earlier experiments for backprop and tune in the same range of parameters for σ and α scaled down by a factor of $(2^0, 2^{-1}, 2^{-2}, 2^{-3})$ as used in the past for the coagent training phase. We allow a subset of nodes in the first layer to be stochastic and hence learn. As shown in Figure 3.2 (left), performance starts to degrade rapidly after more than half the nodes are stochastic, though for

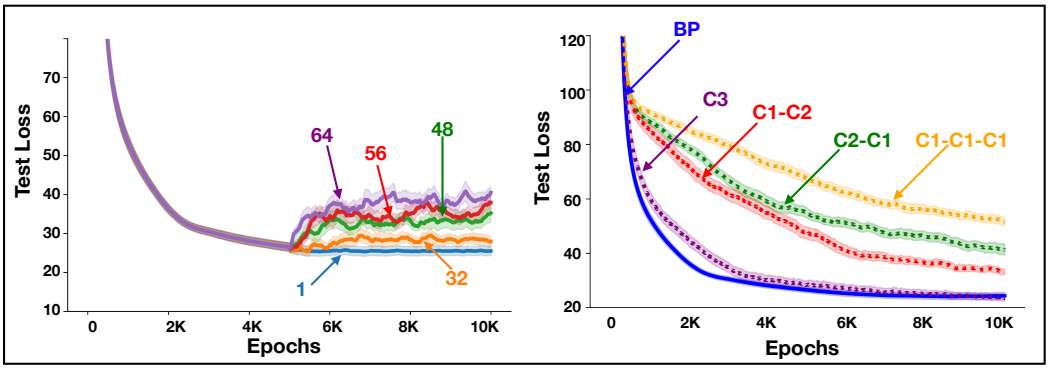

Figure 3.2: Experiments highlighting issues with nondeterminism in coagents. The left graph shows the effect of training with backprop followed by CoAN training for subsets of nodes. The right graph depicts performance with different coagent partitioning of the network.

nodes less than half, the network remains close to the initialized solution.

To further test the effect of stochastic coagents, we test different partitioning schemes of the two-layer coagent network. We can treat the whole network as one coagent, and use backprop within that agent (called C3). We can use two coagents: one that outputs the activations for the final layer and one that learns the weights to produce the prediction (called C2-C1). In general, we have four partitioning schemes, where we always have a prediction coagent to keep the objective consistent, labeled: C1-C1-C1, C2-C1, C1-C2, and C3. C1-C1-C1 has a coagent for each layer, and C1-C2 uses a linear coagent for the first layer and a single layer neural net coagent for the next.

In Figure 3.2 (right), we see the deterministic network C3 performs similarly to the backprop network. And as we add stochastic (non deterministic) agents back into the intermediary layers the performance degrades significantly. This, along with the previous experiment, further supports the claim that the performance issues for CoANs are primarily due to learning with stochastic coagents, rather than problems with variance in the optimization problem.

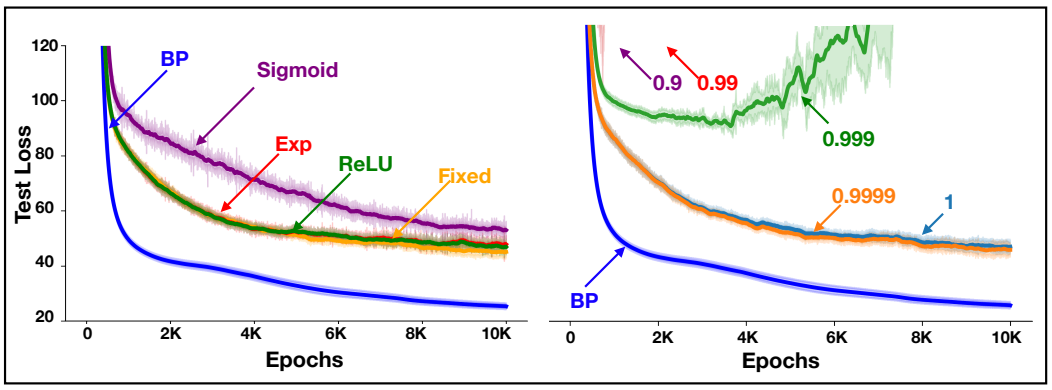

Figure 3.3: Failures of different strategies to gradually reduce nondeterminism of coagents overtime, on Boston Housing. (Left) Learning a bias unit for variance. (Right) Decayed global variance for the whole

3.2.1 Failures of Simple On-policy Approaches to Reduce Nondeterminism

A natural next step is to consider simple strategies to reduce the non determinism of coagents gradually. For example, it is common in RL to have a decaying schedule for the exploration parameter. Similarly here, instead of using a fixed variance parameter for coagents, we can gradually decay this parameter and so allow coagents to gradually learn good actions under nearly greedy actions from other coagents.

We tested two strategies to enable the variance parameter in the Gaussian distributed actions to decrease with time. The first strategy is to learn a σ per node, using a bias unit per node, passed through either a sigmoid, ReLU, or exponential activation to obtain a non-negative number. The second is to use a decay rate for the σ of the whole network. In both cases we initialized σ, with the best value found in the original experiments (i.e. $\sigma = 4.0$).

Unfortunately, these natural strategies to remove stochasticity in coagents do not improve performance, as can be seen in Figure 3.3. One possible reasoning can be explosion in gradients of policy parameterization, mainly because for a smaller σ, the gradient of the $\nabla_{\boldsymbol{\theta}} \ln \pi(a|\boldsymbol{o})$ term can explode significantly for a slightly deviated sample, as can be observed in the faster decay rates in Figure 3.3 (right), i.e., $0.9, 0.99$.

3.3 Discrete Actions Helps Control Stochasticity, But Not Enough

Here we study discrete coagents with Bernoulli discrete actions and their stochasticity over time. Discrete networks provide an easier way to handle stochasticity in their policy. Softmax parameterization adapts stochasticity when learning, and can become fully deterministic if required. For a baseline estimator, we do gradient backpropagation via straight-through (ST) estimators (Y. Bengio et al., 2013; Shekhovtsov et al., 2020), because standard backprop cannot be used for discrete nodes. In this set of experiments, we again use REINFORCE coagents, and test on MNIST. We measure the entropy of the coagents over time and the sparsity of the representation to ensure reasonable levels of activation.

3.3.1 Experimental Details

To keep comparisons similar and avoid overfitting to hyperparameters, we adopt all the hyper-parameters from the continuous coagent case (where it was possible). For discrete nodes, the main difference came from the probability distribution parameterization. The continuous agents used a mean parameterized Gaussian, with a fixed standard deviation for all the nodes in the network. In the discrete case, each coagent has $|\mathcal{A}|$ number of heads. In this experiment we consider a coagent with two actions. The outputs of these heads are fed through a softmax activation and the actions are sampled using a Bernoulli distribution. Each action corresponds to 0 or 1 appropriately.

As the outputs are real-valued in the case of continuous agents, we can treat each layer of the network as an agent and hence be a Gaussian distribution. Whereas in discrete coagents, as the outputs of agents are 0 and 1 values, we can't do classification and regression with just those nodes. Hence, we apply a final linear layer on top of the learned representation of binary nodes. In essence, the π_k agent in discrete agents is a deterministic agent that transforms the binary input $\boldsymbol{o}_k$ to real-valued actions and uses backprop to update this linear layer and use the CoAN update for the agents in all the other layers.

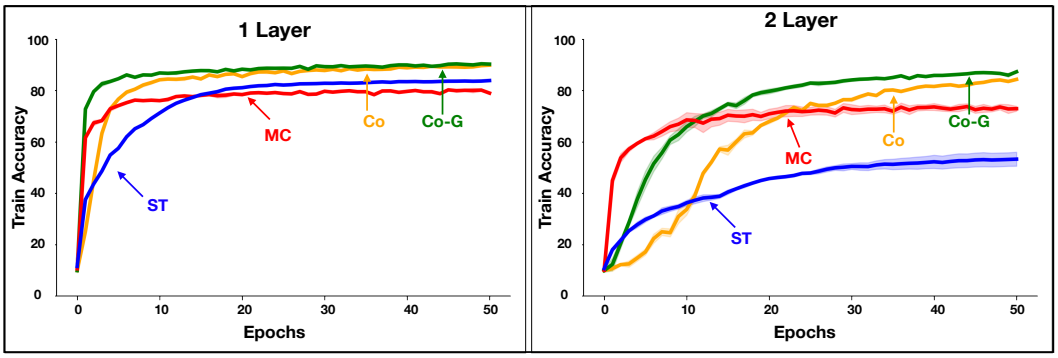

Figure 3.4: Learning curves fro discrete agents (ST, MC, Co, Co-G) with single and double-layers, averaged for 10 runs, for 50 epochs with 64 nodes on MNIST dataset.

Similar to the continuous case, the evaluation is performed greedily, where the action with higher probability is selected.

3.3.2 Results

In Figure 3.4 we can see that CoANs actually perform better than the ST estimator, and here the baseline has a bigger positive effect (Co-G versus Co). ST estimators have difficulties when learning in more than a single layer (Y. Bengio et al., 2013), probably due to misalignment (Shekhovtsov et al., 2020). We also include an on-policy Monte Carlo algorithm that learns action values.

For Co-G, the ability to better control the entropy does seem to be helping, wherein Figure 3.5(a) the entropy drops —earlier for layer one and then at 200 epochs for layer two and three—cause a sudden rise in accuracy. However, the entropy does not fully decrease, and there is some unintuitive behavior. For example, the first layer becomes deterministic faster, which is surprising as it relies on stochastic actions of downstream agents. The entropy for layer two and layer three also decreases, but eventually, the entropy starts to creep back up while maintaining or improving accuracy. Finally, we provide a more in-depth investigation into the stochasticity under discrete actions, including using more discrete actions per coagent and training with (randomized) subsets of coagents fixed.

3.3.3 Investigating Nondeterminism of Discrete Agents

As we saw in section 3.3.2, stochasticity in intermediate layers doesn't decrease, even though the coagents match backprop in terms of accuracy if trained for long enough (i.e., $> 99\%$ training data; figure 3.5(a)). This raises the question of why this happens. There can be multiple reasons, including (1) stochasticity in earlier layers not giving a stationary representation to the later layers, (2) too many interacting nodes in a single layer, (3) representation in intermediate layers being relatively weak to support deterministic policies. We study these properties in our subsequent experiments.

Figure 3.5(b,c,d) tests the above 3 hypothesis. To counter inter layer nondeterminism, we adopt the strategy of freezing earlier layers in the network with a fixed training schedule, in the hope of later layers learning deterministic policies on a stationary representation, which also doesn't seem to be the case. This training regime has an issue that the earlier layers still have confounding stochasticity because of randomness in the downstream layers. For the second set of experiments, we train a subset of randomly selected nodes in a given layer for each epoch while keeping the other nodes fixed. We observe that except for offering some help in early learning speeds, there is no clear benefit to have this strategy.Thirdly, counter to our intuition of the earlier layer being more stochastic, we observe that layer 1 has a sharp decrease in its entropy, which might be because of a strong representation it gets in the form of images. We experiment with different number of nodes ($\in [2^1, \ldots, 2^9]$) in each layer to test this. We also allow agents to have more than 2 actions ($\in [2, 5, 9, 17]$), and actions correspond to a float value distributed evenly in the interval $[-1, 1]$. As we can observe in Figure 3.5(d), more number of nodes in the intermediate layer does seem to help with the entropy of those layers. In contrast, an increased number of actions might not be that helpful for the case of policy gradient algorithms. So rather than having a more fine-grained control, it is better to have a bigger and maybe sparse representation. Overall, we find that the former significantly increases performance for REINFORCE coagents; however, the latter approach does not substantially reduce the entropy,

nor improve performance.

3.4 Summary

This chapter studied the suboptimality of the REINFORCE update for coagents, mainly due to nondeterminism of other coagents, and how different on-policy updates might fail to solve this. We notice some benefits for the case of the discrete node, where they can reduce nondeterminism autonomously based on their gradient update, and even achieve backprop level performance with enough samples.

In the next chapter, we introduce the notion of critics and off-policy learning as a possibility to deal with nondeterminism in the downstream nodes.

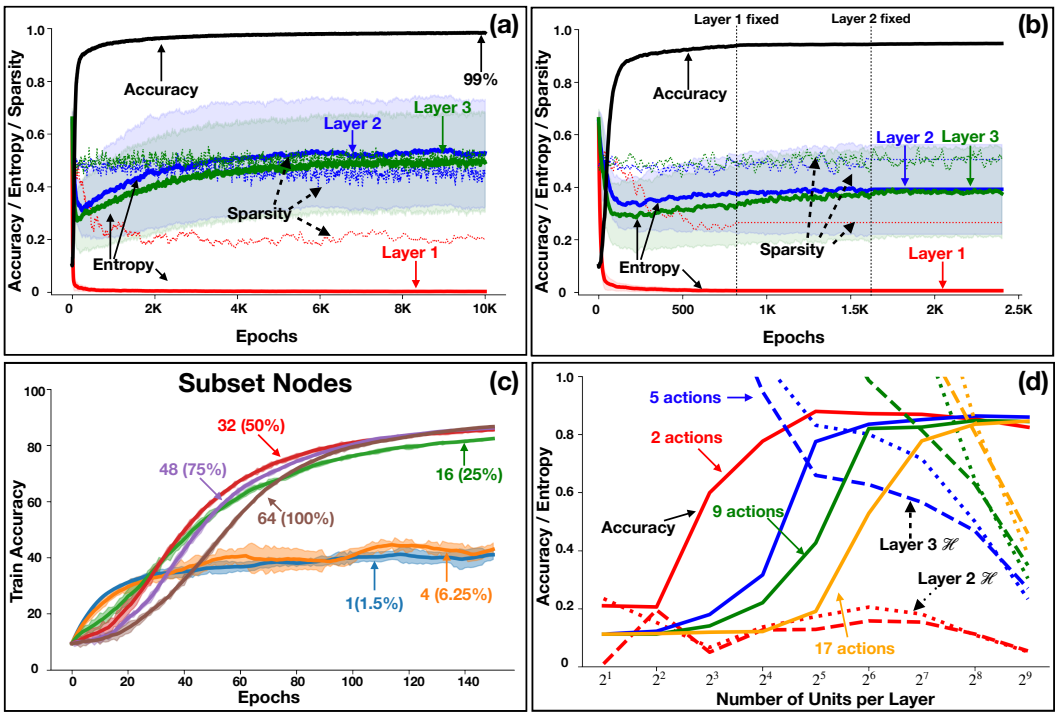

Figure 3.5: All experiments in this figure are on MNIST dataset, with experiment (a,b,c) use 64 nodes per layer. **(a)** Learning progress of three layer network for 10K epochs, with entropy *(solid line)* and sparsity *(dotted line)* of each layer, along with accuracy (scaled to $[0, 1]$). **(b)** Inter layer experiment: fixing earlier layers at regular intervals for a three layer network for 2.4K epochs (fixing at 800 and 1.6K epochs). Shows sparsity and entropy similar to (a). **(c)** Intra layer experiment: training a random fraction of subset of nodes in each layer. Shown for 150 epochs, averaged for 10 runs. **(d)** Representation experiment: sensitivity curves for different number of nodes per layer $\in [2^1, \ldots, 2^9]$ and different number of actions per node $\in [2, 5, 9, 17]$. *Solid* line shows accuracy (scaled to $[0, 1]$), *dotted* line for entropy ($\mathcal{H}$) layer 2 and *dashed* line for entropy of layer 3. All agents were trained for 100 epochs and values represent average of last 20 epochs.

Chapter 4

Off-Policy Learning to Learn CoAgents

The chief difference between using RL and typical optimization approaches, both for SCGs and standard NNs, is that we can learn off-policy. When training a neural network, it is rarely the case that the accuracy of predictions matters when doing an update. Rather, this setting matches the fully offline learning setting—the pure exploration setting—instead of the online setting where the agent needs to maximize reward while learning. This highlights that we can also use many different exploration approaches to gather useful data about how to adjust the policies for more accurate predictions. For example, the behavior could choose to make a poor prediction, to gather experience that is more useful for improving accuracy than if the on-policy best prediction was used. This separation is in stark contrast to methods like backprop.

In this section, we show how to learn critics off-policy, and that this improves on using on-policy critics. We start by describing the algorithm, and then provide results on MNIST.

4.1 An Off-Policy Algorithm for CoANs

The first step is to modify the REINFORCE update, to use an action-value critic. Instead of using a sampled return G, we can estimate the expected return $Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j)$, for the coagent taking action $\boldsymbol{a}_j$ given input $\boldsymbol{o}_j$, namely the

previous hidden layer $\boldsymbol{h}_{j-1}$. The update then uses:

$$(Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j) - V_j(\boldsymbol{o}_j))\nabla_{\boldsymbol{\theta}_j} \ln \pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j)$$

Where for the final layer we do not learn a critic and simply using the immediate reward (i.e., the negative of the error). These action-values can be updated on-policy, each time the network is queried, using a SARSA update

$$\boldsymbol{\theta^{(q)}}_j \leftarrow \boldsymbol{\theta^{(q)}}_j + \alpha(0 + Q_{j+1}(\boldsymbol{o}_{j+1}, \boldsymbol{a}_{j+1}) - Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j))\nabla Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j)$$

where $Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j)$ can simply be a linear function of $\boldsymbol{o}_j, \boldsymbol{a}_j$—namely a linear function of $[\boldsymbol{h}_{j-1}, \boldsymbol{h}_j]$—or could itself be a small neural network.

This strategy, however, introduces bias for two reasons. First, estimating action-values means we have some error in our expected return estimate due both to estimate error and approximation. Second, the local action values are actually tracking a non-stationary target. They estimate the expected return for an action, where the expectation is taken over the input and output as well as the actions of the other coagents. Further, the coagent is attempting to learn how to select actions, given stochastic action selection by the other coagents rather than the best action (greedy action) for each coagent. This nonstationarity and difficulties in credit assignment is well-recognized as an issue in multi-agent reinforcement learning (Foerster et al., 2018; Rashid et al., 2018; Tesauro, 2003). However, in our setting, the known structure between agents means we can more easily obtain a solution, than an unstructured collection of cooperating agents.

The key is to reason about greedy actions of downstream coagents, rather than the action they actually took. The update has a small modification, to instead use a maximum over values in the next layer

$$\boldsymbol{\theta^{(q)}}_j \leftarrow \boldsymbol{\theta^{(q)}}_j + \alpha(0 + \max_{\boldsymbol{a}'} Q_{j+1}(\boldsymbol{o}_{j+1}, \boldsymbol{a}') - Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j))\nabla Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j)$$

Given the input, the agent asks: what is the value of each action, given the maximal actions that will be taken for downstream layers? This update bootstraps only on the action-value in the next layer, but the update for that Q_{j+1} also bootstraps off of the max in the next layer. Therefore, each action-value

starting from the end of the network is learning about maximal action-values for downstream coagents, and propagating that information backwards. This approach directly exploits the known Markov structure of the credit assignment problem, and so should learn more efficiently than using structureless algorithms like REINFORCE. Note the coagents need not reason about greedy actions for upstream coagents, because action-values are *conditioned* on inputs produced by those coagents.

The purpose of these experiments is twofold. Firstly, we wish to study whether off-policy learning can indeed offer advantages in our case. Secondly, as shown in previous experiments in section 3.3, intermediate layers have a hard time settling to deterministic policies. By introducing critics, we can tradeoff some variance for bias; hence we wish to observe whether coagents in intermediate layers are able to approach a deterministic policy.

4.2 Experimental Details

We perform experiments where we estimate the $Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j)$ function for each layer in the CoAN, using the on-policy and off-policy approaches.

The on-policy case only requires a single sampling operation which is included in the forward inferencing part. In the off-policy case, bootstrapping from the greedy action in the next layer requires access to the joint greedy actions of all coagents in that layer with respect to its critic. This is a hard optimization problem, and rather than finding the joint greedy action, we just query each coagent in the next layer for its greedy action. As these agents learn in a stochastic environment and ideally should account for stochasticity when learning their respective policies.

We allowed our agents to select from different forms of critics, which include linear and single-layered neural nets with (64, 128 & 1024) nodes. We found that NN critics worked best with 1024 nodes and a learning rate twice the value of the CoAN. We keep the other hyperparameters the same from Section 3.3. Note that here our goal is to understand if critics can help, and so we allow for larger critics per coagent; practically, depending on the setting, there may

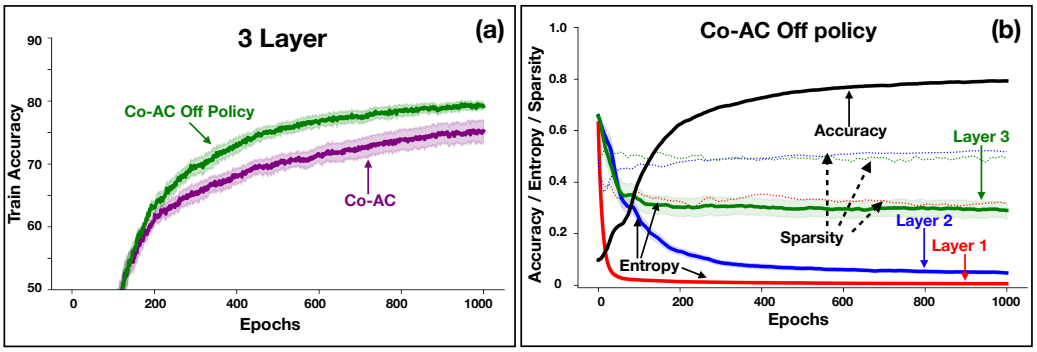

Figure 4.1: **(a)** Learning curves comparing on policy and off policy actor critic. **(b)** Entropy curves for off policy actor critic coagents networks. Both the graphs are for a three layered network, averaged over 10 runs, for 64 nodes and 1K epochs.

be stronger limitations on the critics.

Algorithm 2 provides the pseudo code for critics and off-policy learning in CoAN's.

4.3 Results with Off-Policy CoANs

Figure 4.1 (a) presents the results for on-policy (**Co-AC**) and off-policy (**Co-AC Off Policy**) critics. We find that off-policy critics help improve performance, particularly later in learning, whereas the on-policy critic is plateauing at a lower point. In the next set of graphs (Figure 4.1 (b)), we can observe that the critics can indeed help reduce variance for intermediate layers (layer 2). We don't observe the decay for the last layer because the last layer still uses the noisy reward signal directly. This can be attributed to bootstrapping from targets that average out the variance in targets because of the nondeterminism in downstream nodes, hence allowing for faster convergence. Still, at the same time, biasness causes the point to be sub-optimal.

As a note, the performance with critics is actually worse than that of REINFORCE, likely due to bias in the critics. Better approaches to learning the critic off-policy, including using methods like replay, should help us close this gap. Our goal in this experiment was primarily to contrast the on-policy and off-policy critics, highlighting that off-policy learning is a promising direction

towards addressing the nondeterminism issue in CoANs.

4.4 Summary

This chapter introduced critics for CoAN's learning and presented an off-policy variant of the learning algorithm to deal with the nondeterminism of downstream nodes. We saw that off-policy learning improvements over on-policy critic algorithms but doesn't achieve the performance we observe with the REINFORCE update because of the bias introduced by critics. Also, in its current form, the layer-wise critics handle nondeterminism for nodes in downstream layers, ignoring the same for nodes in the same layer. This would demand learning a critic separately for each coagent, and it is not clear how bootstrapping will occur in that setting.

In the next chapter, we look at settings where active exploration from CoAN's can offer benefits for learning in nonstandard settings and even improve backpropagation.

Algorithm 2 Binary discrete coagent with critic and off policy learning for classification

Input: Number of coagents per layer n, Number of layers k, feature size d and $\mathcal{D}$ be the dataset distribution. Number of epochs e, Batch Size: b.
Initialize: 2 action softmax policy for all $n \times k$ coagents, with linear function approximation parameterized by $\boldsymbol{\theta}_{i,j}$.
Initlialize: Final layer (of size $n \times c$) with c outputs corresponding to number of classes for task, called as f.
Initlialize: Critics $(q_0, \dots q_k)$, one for each layer, mostly a linear function, parameterized by ${\boldsymbol{\theta}^{(q)}}_i$
Input Param: α - stepsize for gradient descent, for all components
repeat
  $\boldsymbol{x}, y \sim \mathcal{D}$ batch of size b
  $\boldsymbol{o}_0 = \boldsymbol{x}$
  `// Forward Pass`
  for $i = 0$ **to** $k - 1$ **do**
    for $j = 0$ **to** $n - 1$ **do**
      $a_{i,j} \sim \pi_{i,j}(\boldsymbol{o}_i)$
    end for
    $\boldsymbol{a}_i = (a_{i,0}, \dots a_{i,n-1})$
    $\boldsymbol{o}_{i+1} = \boldsymbol{a}_i$
  end for
  $\hat{y} = f(\boldsymbol{o}_k)$
  $G = -\text{CrossEntropy}(y, \hat{y})$
  $q_{k+1}(.) = G$
  `// Backward Pass`
  for $i = k$ **to** 0 **do**
    `// Update critics with target policy`
    $\boldsymbol{a}_{i+1} \leftarrow \pi_i^{\text{greedy}}(\boldsymbol{o}_{i+1})$
    $Q_{tar} = q_{i+1}(\boldsymbol{o}_{i+1}, \boldsymbol{a}_{i+1})$
    $g_i^q = (Q_{tar} - q_i(\boldsymbol{o}_i, \boldsymbol{a}_i)) \nabla_{{\boldsymbol{\theta}^{(q)}}_i} q_i(\boldsymbol{o}_i, \boldsymbol{a}_i)$
    ${\boldsymbol{\theta}^{(q)}}_i = {\boldsymbol{\theta}^{(q)}}_i + \alpha g_i^q$
    for $j = 0$ **to** $n - 1$ **do**
      $g_{i,j} = (Q_{tar}) \nabla_{\boldsymbol{\theta}_{i,j}} \ln(\pi_{i,j}(a_{i,j} | \boldsymbol{o}_i))$
      $\boldsymbol{\theta}_{i,j} = \boldsymbol{\theta}_{i,j} + \alpha g_{i,j}$
    end for
  end for
until Number of e epochs achieved

Chapter 5

Potential Advantages of CoAN's

In the previous sections, we chose settings where backprop is effective, to make the results more interpretable and start from the standard learning setting. Backprop is designed to decimate these problems having access to the full model of the agents, and the problem. Whereas the motivation of the CoAN's goes much beyond the standard setting of supervised learning. In this chapter we investigate two simple settings where CoAN's can offer potential advantages over plain gradient descent methods. First setting gives us some insight into possibility of avoiding local minima and saddle points by using stochastic nodes in general. Second setting provides experiments on the performance of CoAN and gradient methods on the online learning setting.

5.1 Avoiding Local Minima

We perform a set of simple experiments to assess the performance of stochastic nodes stacked against deterministic nodes for minimizing functions which might have local minima and saddle points. The goal is to understand if the exploration from stochastic nodes trained using policy gradient methods can escape local minima and saddle points on simple loss surfaces.

5.1.1 Problem Setup and Experimental Details

To keep things simple, we define our loss functions to be parameterized with a scalar weight w. For the first loss function $\mathcal{L}_1(w) = -w + \sin w$, in which case the optimal value lies towards $w^* = +\infty$, with the $\mathcal{L}_1(w^*) = -\infty$. The

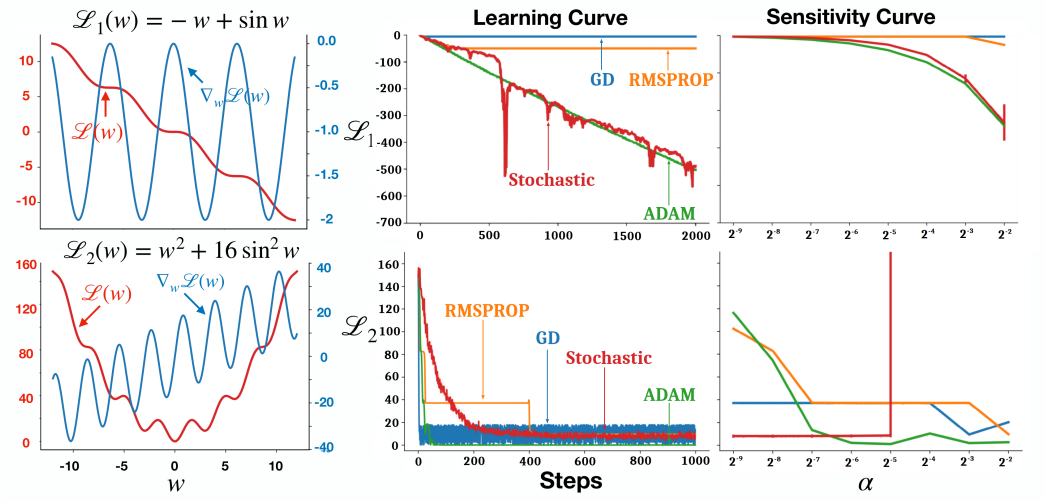

Figure 5.1: First column depicts the loss curve and corresponding derivative with respect to the scalar weight of both problems. Second column corresponds to the learning curve followed by the sensitivity of α for all the methods. The stochastic node is averaged over 50 runs.

loss surface has several saddle points. This problem can be solved with large step sizes in a single direction. We also look at a problem with a real valued global minima and several local minima with loss $\mathcal{L}_2(w) = w^2 + 16\sin^2 w$, where $w* = 0$ and $\mathcal{L}_2(w^*) = 0$.

The experiment is aimed at comparing learning methods, hence we start the optimization process from $w_0 = 2$ for $\mathcal{L}_1$ and $w_0 = -12$ for $\mathcal{L}_2$. We optimize for a single weight vector, using different gradient strategies, i.e. gradient descent (GD), RMSprop (swept with $\beta \in \{0.9, 0.99, 0.999\}$) and ADAM (Kingma and Ba, 2015) (swept with β_1 and $\beta_2 \in \{[0.9, 0.99, 0.999\}$). We also train a stochastic node using REINFORCE and a baseline. The stochastic node outputs a weight in the learning process, the weight is sampled from a Gaussian distribution where $\mu_0 = w_0$ with standard deviation σ swept in $\{2^{-2}, 2^{-1}, 2^0, 2^1, 2^2, 2^3\}$.

For the first problem each algorithm runs for 2000 update steps, for the second problem, each algorithm runs for 1000 update steps. In both cases we sweep $\alpha \in \{2^{-2}, \ldots, 2^{-9}\}$ and the stochastic node is averaged over 50 runs.

5.1.2 Results

We report the performance of the different learning methods in Figure 5.1. For the first loss function $\mathcal{L}_1$, GD and RMSprop both get stuck at the saddle points, while ADAM is able to escape saddle points likely due to the momentum term, and generally performs similar to the stochastic node. For the second loss function $\mathcal{L}_2$, ADAM outperforms the stochastic node with larger step sizes (and likely with smaller stepsizes too given enough steps to run), even without added noise in the loss. RMSprop performs with $\alpha = 0.25$ about as well as the stochastic node with smaller step sizes. Gradient Descent quickly reaches the basin of global minimum, however it oscillates around it due to the large stepsize, whereas with smaller stepsize it gets stuck in a local minimum and overall ends up performing worst from all four methods. As the sensitivity curve in Figure 5.1 demonstrates, the stochastic node is insensitive to and performs best with smaller step sizes, but at $\alpha = 2^{-5}$ and above it is unable to learn. This experiment hence demonstrates that stochasticity in updates is an effective strategy to optimize (although its tends to be high variance), requires little stepsize-tuning as long as the stepsize is small enough. Whereas for backprop ADAM proves to be an effective strategy to solve this sort of problems with appropriate hyperparameter tuning.

One might argue that the stochastic variants of gradient descent methods can introduce the relevant stochasticity to avoid local minima and saddle points when comparing to coagents. It is important to note, that in those settings the stochasticity is not under the control of the agents, i.e. the stochasticity is the feature of the environment, whereas with stochastic nodes, the responsibility of exploration lies with the agent itself, hence giving it control in the direction of optimization with clever exploration techniques.

5.2 Continual Learning on Correlated Data

The goal of this section is to understand the learning in online learning settings with correlated data and the need for real-time learning and decision-making. CoANs also naturally facilitate asynchronous inputs and updating (Kostas et

al., 2020), as well as learning with recurrence.

In this section, we investigate the utility of CoANs for prediction when learning online with highly temporally correlated inputs. We expect CoANs to be less prone to failure than backprop, which can completely overwrite previous learning when learning on correlated data. Furthermore, because coagents are RL agents, the policies should better track and adapt to changes in the environment, which again is a feature of active exploration.

5.2.1 The Continual Learning Problem

To measure CoAN's ability to learn online with various degrees of temporal correlation, we examine the performance of Co-G on the PieceWise Random Walk Problem introduced in Pan et al. (2021), with the same parameter settings and target function.

Piecewise Random Walk problem

Pan et al. (2021) introduced the following problem where we have a fixed Gaussian $X_t \sim \mathcal{N}(S_t, \beta^2)$, with fixed variance β^2 and mean S_t, that drifts every T time steps. The mean S_t, stays fixed for T time steps, then drifts according to a random walk.

In this dataset, the correlation difficulty parameter $d \in [0, 1]$ controls the amount of temporal correlation within the data: 0 stands for iid data points, while 1 indicates a fully temporally correlated dataset, while maintaining a stationary distribution. The dataset generated $\{(X_t, Y_t)\}_{t \in N}$ is from the class of regression problem, where $x_t \sim X_t$ is generated as described above, and regression label y_t is defined as $y_t = \sin 2\pi x_t^2$. Figure 5.2 shows examples of the data points produced for different level of difficulties starting from iid (d = 0), to very difficult i.e. completely correlated (d=1).

5.2.2 Experiments

We first train Co-G and backprop for 180,000 training steps, with d $\in \{0, 0.85, 0.95, 1\}$, and test it every 900^{th} step on a test set of 1,800 iid samples, with the best

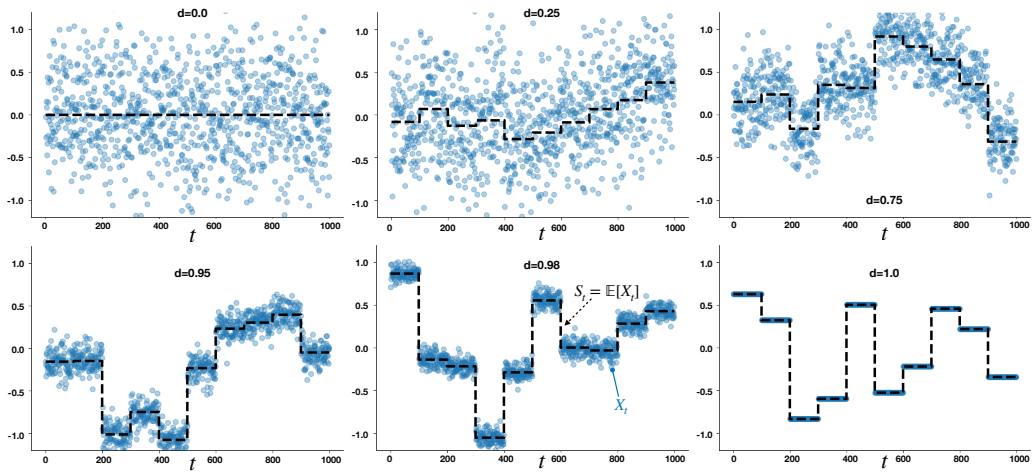

Figure 5.2: Piecewise random walk problem, with sample trajectories of $\{X_t\}_{t\in\mathbb{N}}$ (blue) and $\{S_t\}_{t\in\mathbb{N}}$ (black), with different level of difficulties.

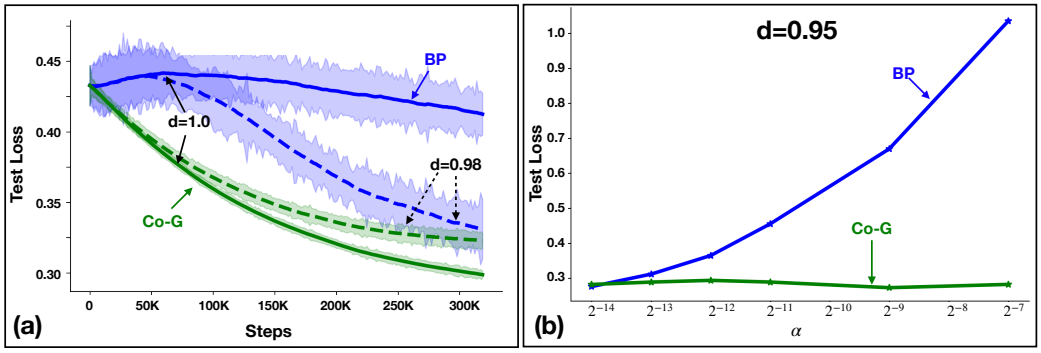

Figure 5.3: **(a)** CL experiments with problem difficulties at $d = 0.98$ *dashed* lines and *solid* lines for $d = 1.0$ comparing backprop and CoAN's. **(b)** Sensitivity plots in CL problem for learning rate (α), for problem difficulty of $d = 0.95$.

hyperparameters picked on the validation set of equal size. Standard deviation for Co-G is swept $\sigma \in \{2^{-4}, 2^{-2}, 2^{0}, 2^{1}, 2^{2}, 2^{3}, 2^{4}\}$. For both algorithms, stepsize values are swept $\alpha \in \{2^{-14}, 2^{-13}, 2^{-12}, 2^{-11}2^{-9}, 2^{-7}\}$, the batch size is fixed at 32, number of units at 50, and number of hidden layers at 1. Co-G is trained using the RMSProp optimizer, while backprop is trained using the Adam optimizer, both with standard β values. Then, using the chosen hyperparameters, we allow the agent to continue learning up to 320,000 training steps on d $\in \{0.98, 1\}$, to gauge the long run performance on extremely correlated data. Results are averaged over 200 runs.

5.2.3 Results

As expected, backprop outperforms Co-G when trained on iid samples, and it also achieves lower error on d=0.85. Co-G begins to display an edge at d=0.95, where it achieves similar performance to backprop, and at d=1, Co-G performs better and continues to learn steadily, while backprop is incapable of learning. In addition, the sensitivity plot for this experiment in Figure 5.3 (b) depicts that while backprop's performance is largely dependent on the correct alpha parameter, Co-G's performance remains approximately the same across the swept alpha values.

Looking at long run performance after 180,000 steps, we see two interesting phenomena. Co-G is able to continue to steadily learn on the highly correlated dataset, unlike backprop. But, for d=0.98, backprop actually starts to match the performance of Co-G. Again, Co-G may be plateauing early at a suboptimal solution, as we found in previous results. As well, for backprop improving eventually on this highly correlated datastream, these results are for very carefully swept hyperparameters, and we do see that backprop is quite sensitive to the stepsize. Under this heavy correlation, therefore, we find that CoANs provide more robust performance, in that they provide steady progress from the very beginning of learning and are much less sensitive to the stepsize.

Chapter 6

Conclusion

In this text, we investigated the use of reinforcement learning (RL) for the structural credit assignment problem in neural networks. We formalized this problem as a finite-horizon MDP, and showed that local policy gradient updates for each node (coagent) provide an unbiased estimate of the joint policy gradient for all nodes. We show that the basic local policy gradient update for this coagent network (CoAN) can learn—even under difficult learning settings like highly correlated data—but that it plateaus at suboptimal solutions. Through a set of targeted experiments, we identify nondeterminism amongst coagents resulting in each node outputting suboptimal activations, to account for the suboptimal sampled activations of other coagents. We highlight that off-policy learning can naturally be applied to this problem, through the use of off-policy critics. We show that this strategy is promising, that much more work needs to be done to improve the learned critics and mitigate bias.

An important point highlighted in this work is that RL approaches to credit assignment have not been systematically explored for structural credit assignment. The work formalized CoANs focused primarily on credit assignment across time (Kostas et al., 2020), without exploiting the internal structure of the network to improve coagent learning. Some of the work in stochastic computation graphs has looked at generic uses of baselines and critics (Weber et al., 2019), but even there when discussing RL, did not consider the relationship to structural credit assignment for a standard feedforward NN and the ability to exploit the finite horizon structure. The Markov structure fa-

cilitates bootstrapping and learning about the dynamics more efficiently just using short transitions. There are more insights, from planning, replay and exploration, that can be leveraged to improve structural credit assignment.

In their current form, the methods presented are still limited in their application and performance over standard methods. This raises two points to work on in the future. The first is to obtain better off-policy methods to close the gap between standard REINFORCE methods and actor-critic while getting the potential benefits of off-policy learning. Having individual critics for each coagent will allow them to also deal with non-determinism from the coagents in the same layer and perhaps offer a more sound way to perform greedification operation in the off-policy setting.

Secondly, studying more deeply the settings where CoAN's have clear benefits, just like we showed how CoAN's might offer benefits in continual learning. It demands a more rigorous investigation as to why this happens and if there are other problem settings / non-standard network structures that can leverage this form of credit assignment.

References

Agogino, A. K., & Tumer, K. (2004). Unifying Temporal and Structural Credit Assignment Problems, In *Proceedings of the Annual Conference on Autonomous Agents, AAMAS*. 5

Balduzzi, D., Vanchinathan, H., & Buhmann, J. (2015). Kickback Cuts Backprop's Red-Tape: Biologically Plausible Credit Assignment in Neural Networks. *Proceedings of the AAAI Conference on Artificial Intelligence*. 4

Barto, A. G. (1985). Learning by statistical cooperation of self interested neuron-like computing elements. *Human Neurobiology*. 2

Bengio, E., Bacon, P.-L., Pineau, J., & Precup, D. (2015). Conditional Computation in Neural Networks for Faster Models. *arXiv:1511.06297*. 5

Bengio, Y. (2014). How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation. *arXiv:1407.7906*. 2, 3

Bengio, Y., Lee, D.-H., Bornschein, J., Mesnard, T., & Lin, Z. (2016). Towards Biologically Plausible Deep Learning. *arXiv:1502.04156*. 2

Bengio, Y., Léonard, N., & Courville, A. (2013). Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432*. 2, 5, 24, 25

Carreira-Perpinan, M., & Wang, W. (2014). Distributed Optimization of Deeply Nested Systems, In *Artificial Intelligence and Statistics, AISTATS*. 2, 3

Chang, Y.-H., Ho, T., & Kaelbling, L. P. (2003). All Learning is Local: Multi-Agent Learning in Global Reward Games, In *Advances in Neural Information Processing Systems, NeurIPs*. 5

Denoyer, L., & Gallinari, P. (2014). Deep Sequential Neural Network. *arXiv:1410.0510*. 5

Fiete, I. R., & Seung, H. S. (2006). Gradient Learning in Spiking Neural Networks by Dynamic Perturbation of Conductances. *Physical Review Letters*. 2, 4

Foerster, J., Farquhar, G., Afouras, T., Nardelli, N., & Whiteson, S. (2018). Counterfactual Multi-Agent Policy Gradients, In *Proceedings of the AAAI Conference on Artificial Intelligence*. 6, 30

Gu, S., Levine, S., Sutskever, I., & Mnih, A. (2016). Muprop: Unbiased Backpropagation for Stochastic Neural Networks, In *International Conference on Learning Representations, ICLR*. 5

Hinton, G. E. (2002). Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*. 4

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., & Kavukcuoglu, K. (2017). Decoupled Neural Interfaces Using Synthetic Gradients, In *International Conference on Machine Learning, ICML*. 2

Kingma, D. P., & Ba, J. (2015). Adam: A Method for Stochastic Optimization, In *3rd International Conference on Learning Representations, ICLR*. 36

Klopf, A. H. (1982). *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*. 1

Kostas, J., Nota, C., & Thomas, P. (2020). Asynchronous Coagent Networks, In *International Conference on Machine Learning, ICML*. 2, 12, 37, 41

Lansdell, B. J., Prakash, P. R., & Körding, K. P. (2020). Learning to Solve the Credit Assignment Problem, In *International Conference on Learning Representations, ICLR*. 2

LeCun, Y., & Cortes, C. (2010). MNIST Handwritten Digit Database. 14

Lee, D.-H., Zhang, S., Fischer, A., & Bengio, Y. (2015). Difference Target Propagation, In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*. 2, 3

Legenstein, R., Chase, S. M., Schwartz, A. B., & Maass, W. (2010). A Reward-Modulated Hebbian Learning Rule Can Explain Experimentally Observed Network Reorganization in a Brain Control Task. *Journal of Neuroscience*. 4, 5

Legenstein, R., Pecevski, D., & Maass, W. (2008). A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback. *PLOS Computational Biology*. 4

Lillicrap, T. P., Cownden, D., Tweed, D. B., & Akerman, C. J. (2016). Random Synaptic Feedback Weights Support Error Backpropagation for Deep Learning. *Nature Communications*. 4

Mahmood, A. R., & Sutton, R. S. (2013). Representation Search through Generate and Test, In *Proceedings of the AAAI Conference on Artificial Intelligence*. 4

Markram, H., Gerstner, W., & Sjöström, P. J. (2012). Spike-Timing-Dependent Plasticity: A Comprehensive Overview. *Frontiers in Synaptic Neuroscience*. 4

Merkh, T., & Montúfar, G. (2019). Stochastic Feedforward Neural Networks: Universal Approximation. *arXiv:1910.09763*. 5

Miconi, T. (2017). Biologically Plausible Learning in Recurrent Neural Networks Reproduces Neural Dynamics Observed during Cognitive Tasks. *eLife*. 5

Movellan, J. (1991). Contrastive Hebbian Learning in the Continuous Hopfield Model. *Connectionist Models*. 4

Narendra, K. S., & Thathachar, M. A. L. (1989). *Learning automata - an introduction*. 2

Nath, S., Liu, V., Chan, A., Li, X., White, A., & White, M. (2020). Training Recurrent Neural Networks Online by Learning Explicit State Variables, In *International Conference on Learning Representations, ICLR*. 2, 4

Neal, R. M. (1990). *Learning Stochastic Feedforward Networks* (tech. rep.). 5

Nøkland, A. (2016). Direct Feedback Alignment Provides Learning in Deep Neural Networks, In *Advances in Neural Information Processing Systems, NeurIPs*. 4

Oliehoek, F. A., Spaan, M. T. J., & Vlassis, N. (2008). Optimal and Approximate Q-Value Functions for Decentralized POMDPs. *Journal of Artificial Intelligence Research*. 6

Pan, Y., Banman, K., & White, M. (2021). Fuzzy Tiling Activations: A Simple Approach to Learning Sparse Representations Online, In *International Conference on Learning Representations, ICLR*. 38

Raiko, T., Berglund, M., Alain, G., & Dinh, L. (2015). Techniques for Learning Binary Stochastic Feedforward Neural Networks, In *International Conference on Learning Representations, ICLR*. 5, 6

Rashid, T., Samvelyan, M., de Witt, C. S., Farquhar, G., Foerster, J. N., & Whiteson, S. (2018). QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning, In *International Conference on Machine Learning, ICML*. 6, 30

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning Representations by Back-propagating Errors. *Nature*. 1

Saito, H., Katahira, K., Okanoya, K., & Okada, M. (2011). Statistical Mechanics of Structural and Temporal Credit Assignment Effects on Learning in Neural Networks. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*. 4

Scellier, B., & Bengio, Y. (2017). Equilibrium Propagation: Bridging the Gap between Energy-Based Models and Backpropagation. *Frontiers in Computational Neuroscience*. 4

Schulman, J., Heess, N., Weber, T., & Abbeel, P. (2015). Gradient Estimation Using Stochastic Computation Graphs, In *Advances in Neural Information Processing Systems, NeurIPs*. 5

Shekhovtsov, A., & Yanush, V. (2021). Reintroducing Straight-Through Estimators as Principled Methods for Stochastic Binary Networks. *arXiv:2006.06880*. 5

Shekhovtsov, A., Yanush, V., & Flach, B. (2020). Path Sample-Analytic Gradient Estimators for Stochastic Binary Networks, In *Advances in Neural Information Processing Systems, NeurIPs*. 24, 25

Tan, M. (1993). Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents, In *International Conference on Machine Learning, ICML*. 5

Tang, Y., & Salakhutdinov, R. (2013). Learning Stochastic Feedforward Neural Networks, In *Advances in Neural Information Processing Systems, NeurIPs*. 5

Tavanaei, A., Ghodrati, M., Kheradpisheh, S. R., Masquelier, T., & Maida, A. S. (2019). Deep Learning in Spiking Neural Networks. *Neural Networks*. 6

Tesauro, G. (2003). Extending Q-Learning to General Adaptive Multi-Agent Systems, In *Advances in Neural Information Processing Systems, NeurIPs*. 6, 30

Thomas, P. S. (2011). Policy Gradient Coagent Networks, In *Advances in Neural Information Processing Systems, NeurIPs*. 2

Thomas, P. S., & Barto, A. G. (2011). Conjugate Markov Decision Processes, In *International Conference on Machine Learning, ICML*. 2

Tieleman, T., & Hinton, G. (2012). Lecture 6.5—RmsProp: Divide the Gradient by a Running Average of its Recent Magnitude. 15

Veeriah, V., Zhang, S., & Sutton, R. S. (2017). Crossprop: Learning Representations by Stochastic Meta-Gradient Descent in Neural Networks, In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*. 5

Weber, T., Heess, N., Buesing, L., & Silver, D. (2019). Credit Assignment Techniques in Stochastic Computation Graphs, In *Artificial Intelligence and Statistics, AISTATS*. 2, 5, 12, 15, 41, 47–50

Williams, R. J. (1992). Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Learning*. 1

Wolpert, D. H., & Field, M. (2002). Optimal Payoff Functions for Members of Collectives. *Modeling Complexity in Economic and Social Systems*. 5

Wolpert, D. H., Wheeler, K. R., & Tumer, K. (1999). General Principles of Learning-Based Multi-Agent Systems, In *Proceedings of the Annual Conference on Autonomous Agents, AAMAS*. 5

Appendix A

Appendix

A.1 Stochastic Computation Graph for Finite Horizon MDP

In this section we aim to connect our formalism to the more general concepts of stochastic computation graph (SCG). This allows us to use concepts from that work and develop algorithms for same.

Weber et al. (2019) showed that the finite horizon MDP can be seen as stochastic computation graphs. Since we formalize our problem as a finite horizon MDP, our problem can also be seen as an instance of SCG. First, we define the SCG framework and then present the MDP in the same.

Definition 1 (Stochastic Computation Graph)**.** *A stochastic computation graphs $\mathcal{G}$ is a directed acyclic graph, with two classes of nodes (also called as variables).*

- ***Stochastic Nodes*** *(represented with circles ◯, and set $\mathcal{S}$[1]), which are distributed conditionally on their parents*
- ***Deterministic Nodes*** *(represented with squares □, and set $\mathcal{D}$), deterministic functions of their parents. Within deterministic nodes we also have other classes of nodes as follows*
  - ***Parameter Nodes****: These nodes don't have parents in the graph,*

[1]not to be confused with the state set of an MDP, for this section we will focus on this being the set of stochastic nodes in the computation graph.

and are set externally, for e.g. the parameters we wish to differentiate with respect to $\theta \in \Theta$

- ***Loss Nodes*** *(represented as diamonds $\diamond$), which represents the cost / losses with respect to which we aim to compute he gradient of our parameter nodes Θ, denoted as $l \in \mathcal{L}$.*

A parent v of a node w is connected to it by a directed edge (v, w). Let $L = \sum_{l \in \mathcal{L}} l$ be the total cost.

For simplicity, for a node v, we let h_v denote the set of parents of v in the graph. If there exists a path between v and w, such that $a_0 = v, a_1, \ldots, a_{M-1}, a_M = w$, where (a_{m-1}, a_m) are directed edges of the graph, we can say, w descends from v, denoted as $v \prec w$.

Also, we define the gradients of any sampling operation to be zero, i.e., the gradient of non-deterministic descendants with respect to inputs is always zero.

Gradient estimation for SCG The expected loss $J(\theta) = \mathbb{E}_{s \in \mathcal{S}}[L]$. The gradient of $J(\theta)$ wrt to θ is given by Weber et al., 2019, Theorem 1

$$\frac{d}{d\theta} J(\theta) = \mathbb{E}\left[\sum_{v \in \mathcal{S}, \theta \prec v} s(v, \theta) L + \sum_{l \in L, \theta \prec l} \frac{dl}{d\theta} \right] \tag{A.1}$$

The first part of the equation corresponds to the REINFORCE update. The second part includes all the paths to cost, containing a deterministic path from the parameters (as gradients through stochastic path automatically are defined as 0).

Score Function. For any stochastic variable v, we let $\log p(v)$ denote the conditional log-probability of v given its parents, and let $s(v, \theta)$ denote the score function $\frac{d \log p(v)}{d\theta}$

Figure A.1 shows an example of an SCG, through which we will explain different concepts as well.

Definition 2 (Value Function)**.** *Let $\mathcal{X}$ be an arbitrary subset of $\mathcal{G}$, $\boldsymbol{x}$ an assignment of the possible values to variables in $\mathcal{X}$ and S an arbitrary scalar*

value in graph. The value function for set $\mathcal{X}$ is the expectation of the quantity S conditioned on $\mathcal{X}$:

$$V : \boldsymbol{x} \to V(\boldsymbol{x}; S) = \mathbb{E}_{\mathcal{G}\setminus\mathcal{X}|\mathcal{X}=\boldsymbol{x}}[S]$$

Intuitively, the value function averages out the effect of other nodes (not in the set under consideration $\mathcal{X}$) on the cost.

Rooted cost to go. Considering a node $v \in \mathcal{G}$, we define $L(v) = \sum_{l\in\mathcal{L}, v\prec l} l$, where we can replace v with a set of nodes also i.e. V, and $L(V)$ would represent the cost for that set. The scalar S is usually the cost to go to $L(v)$ for some node (set of nodes for V).

We are going to define two more functions that generalize to the policy gradients.

Definition 3 (Baselines)**.** *A baseline B for v is any function of the graph such that $\mathbb{E}[s(v,\theta)B] = 0$. A baseline set $\mathcal{B}$ is an arbitrary subset of the non-descendants of v.*

Baselines are of interest because any arbitrary scalar function of $\mathcal{B}$, is a baseline for v. Common choices for baseline include, constants i.e. $\mathcal{B} = \emptyset$, or $\mathcal{B} = h_v$, i.e. parents of v.

Definition 4 (Critic)**.** *A critic Q of cost $L(v)$ for v is any function of the graph such that $\mathbb{E}[s(v,\theta)(L(v) - Q)] = 0$*

Having defined critics and baselines, consider an arbitrary baseline B_v and arbitrary critic Q_v for each stochastic node v, we can define a surrogate loss for equation A.1 which provides an unbiased gradient, i.e. Weber et al., 2019, Theorem 2

$$\frac{d}{d\theta}J(\theta) = \mathbb{E}\left[\sum_{v\in\mathcal{S}, \theta\prec v} s(v,\theta)(Q_v - B_v) + \sum_{l\in\mathcal{L}, \theta\prec l}\frac{d}{d\theta}l\right] \tag{A.2}$$

Where difference $Q_v - B_v$ between critic and baseline is called as the *advantage* function. We skip the proof for brevity.

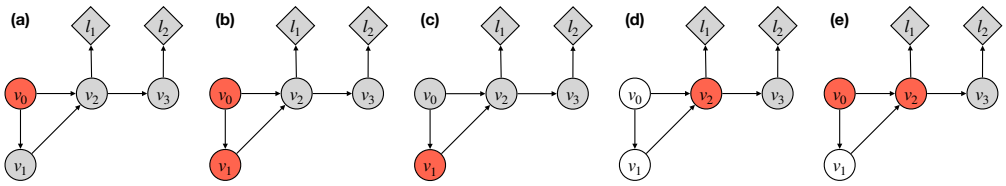

Figure A.1: (a-e) Different value functions for the same SCG with $\mathcal{X}^a = \{v_0\}, \mathcal{X}^b = \{v_0, v_1\}, \mathcal{X}^c = \{v_1\}, \mathcal{X}^d = \{v_2\}, \mathcal{X}^e = \{v_0, v_2\}$. The variables averaged over are shaded in light gray.
(a,b,c) are valid baselines for v_2; (d,e) cannot act as baselines for v_2 since v_2 belongs to those sets, but can act as baseline for v_3.
(a,b,e) are critics for v_0, but (c,d) are not. (b) is a critic for v_1. (c) is not a critic for v_1 Finally (d,e) are critics for v_2
This plot is adapted from from Weber et al., 2019, Figure 3

Definition 5 (Baseline value function and critic value function)**.** ***Baseline value function*** *is defined as a value function for a node v which has a baseline set $\mathcal{B}$.*

The ***critic value function*** *is defined as a value function for set $\mathcal{C}$ and node v, such that $v \in \mathcal{C}$, and $\log p(v)$ and $L(v)$ are conditionally independent given $\mathcal{C}$.*

Figure A.2 shows the example of a finite horizon MDP defined as an SCG, with appropriate node types. This graph structure gives us a clear picture to identify appropriate baselines and critics for our SCG.

For the standard MDP setup (Figure A.2) in RL, $\mathcal{C}$ consists of the state s and action a, which is taken by stochastic policy π in state s with probability $\log \pi(a|s)$ which is a deterministic function of (s, a). The definition is more general than the conventional usage of critics as it does not require $\mathcal{C}$ to contain all stochastic ancestors nodes required to evaluate $\log p(v)$.

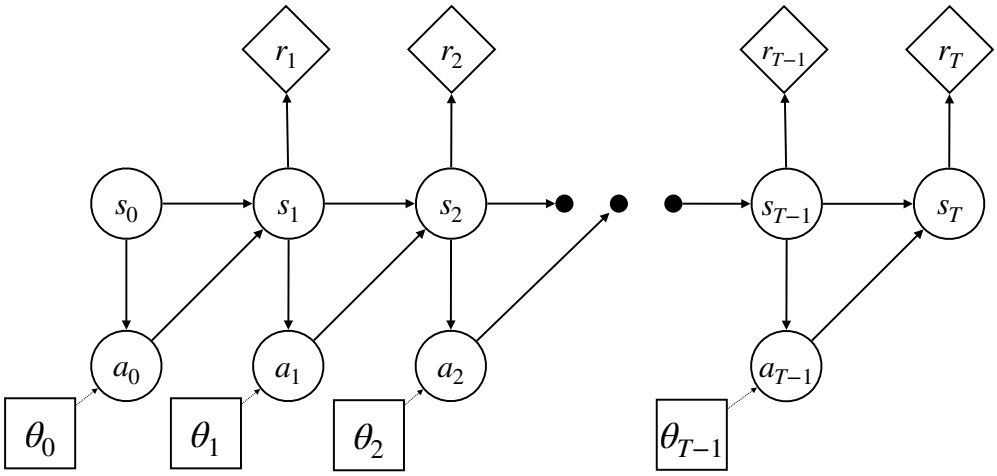

Figure A.2: SCG for a finite horizon MDP. Each step uses a different set of parameters for their policy, unlike standard MDP setting, where the policy applied at each step might share the same parameters.