Structural Credit Assignment in Neural Networks using Reinforcement Learning

Dhawal Gupta, Gabor Mihucz, Matthew K. Schlegel
Department of Computing Science, Alberta Machine Intelligence Institute (Amii)
University of Alberta
`{dhawal,mihucz,mkschleg}@ualberta.ca`

James E. Kostas, Philip S. Thomas
College of Information and Computer Sciences
University of Massachusetts
`{jekostas,pthomas}@cs.umass.edu`

Martha White
Department of Computing Science
CIFAR AI Chair, Amii
University of Alberta
`whitem@ualberta.ca`

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. We show that the standard on-policy REINFORCE approach, even with a variety of variance reduction approaches, learns suboptimal solutions. We introduce an off-policy approach, to facilitate reasoning about the greedy action for other agents and help overcome stochasticity in other agents. We conclude by showing that these networks of agents can be more robust to correlated samples when learning online.

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 [44], 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 [23] and the hedonistic neuron. It is not surprising that the idea of using an RL agent for each node is

35th Conference on Neural Information Processing Systems (NeurIPS 2021).

found in early work, including the original REINFORCE algorithm [61], 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 [14] and CoAgent Networks (CoANs) [56; 55; 24]. 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 [60]. 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 [8], rather than improved RL approaches. Other work on credit assignment is loosely inspired by the idea of bootstrapping in RL, including synthetic gradients [20; 26] and fixed-point propagation [36].

Overall, however, the broader space of RL algorithms has not been leveraged to learn CoANs. 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 and off-policy learning—that can be leveraged to improve learning in CoANs. 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 when moving away from the standard iid learning setting, showing it can perform better than backprop on a continual learning problem with a highly correlated dataset.

1.1 Other 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 [7; 28], the method of auxiliary variables [9] and fixed point propagation [36]. Kickback approximates the backprop update for ReLu networks, using an approximation to the gradient that allows for local updates [5]. Feedback alignment [31; 38] 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 [46]. Fiete and Seung [14] 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; the connection is lost for other perturbations as well as for other RL algorithms.

Other work has focused on changes in weights across time. Spiking neural networks and the associated spike-timing-dependent plasticity (STDP) learning rule [32] adjusts weights based on the relative timing of activations for nearby nodes. Equilibrium propagation [47] 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 [35] and Contrastive Divergence [18]. 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 [29; 30], with some interesting insights that perturbations can be beneficial to induce exploration [30]. 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 [30; 34]. A more recent algorithm, called cross propagation [58], 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 [13] and learning when to activate parts of the network [8; 6]. 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 [4].

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 [37; 8; 42; 33], though the wider literature on stochastic computation graphs encompasses a broad range of stochastic neural networks [52; 48; 60]. Early work considered an EM-style algorithm [52] and a simple alternative, called the straight-through estimator [8], 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 [49], and a lower-variance update has been proposed [17].

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 [63; 62]; to use a single global reward plus some noise [11]; to use independent Q-learners [51]; or to estimate a global critic [39] potentially with local policy updates [15; 43]. 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 [54] 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 works empirically investigating 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 [53]. Stochastic neural networks, particularly with binary activations, have been shown to be difficult to train, but with some improvements to the gradient estimator, can have significant advantages, including providing a level of regularization [42]. In this work, we aim to provide a more comprehensive empirical investigation into the use of reinforcement learning approaches to train CoANs.

2 Structural Credit Assignment as a Finite Horizon RL Problem

In this section, we describe how to formalize the credit assignment problem in a feedforward neural network as a finite-horizon RL problem. We start in the simplest setting, where we have a fully connected feedforward neural network composed of k layers of size n. We assume we have inputs $\boldsymbol{x} \in \mathcal{X}$, prediction targets $y \in \mathcal{Y}$, hidden layer activations $\boldsymbol{a}_j = f(\boldsymbol{z}_j)$ for pre-activations $\boldsymbol{z}_j$ with activation function f. The basic idea for the finite horizon formulation is that the horizon is the number of layers, and 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.

Consider the following agent-environment interaction. On the first step, given the sampled input $\boldsymbol{x} \in \mathcal{X}$, the initial observation is $\boldsymbol{o}_0 = [\boldsymbol{x}, 0]$ where the 0 in the observation indicates the step-index in the finite horizon problem. The agent observes input $\boldsymbol{o}_0$ and then takes action $\boldsymbol{a}_0 \in \mathcal{A}$ that is a vector of activations (or pre-activations $\boldsymbol{z}_0$)—these actions can be discrete or continuous. The next observation is deterministically $\boldsymbol{o}_1 = [\boldsymbol{a}_0, 1]$ (or $\boldsymbol{o}_1 = [f(\boldsymbol{a}_0), 1] = [f(\boldsymbol{z}_0), 1]$). 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 $\hat{y}$. We depict this interaction in Figure 1.

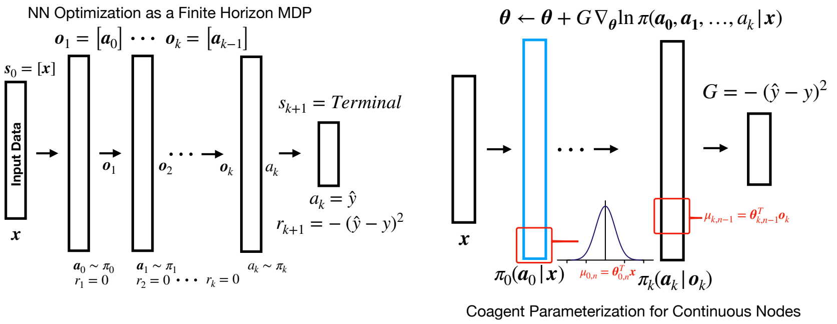
NN Optimization as a Finite Horizon MDP
$o_1 = [a_0] \cdots o_k = [a_{k-1}]$
$s_0 = [x]$
Input Data
x
o_1
o_2
o_k
a_k
$s_{k+1} = Terminal$
$a_k = \hat{y}$
$r_{k+1} = -(\hat{y} - y)^2$
$a_0 \sim \pi_0$
$r_1 = 0$
$a_1 \sim \pi_1$
$r_2 = 0 \cdots r_k = 0$
$a_k \sim \pi_k$
$\theta \leftarrow \theta + G\nabla_{\theta} \ln \pi(a_0, a_1, \ldots, a_k|x)$
$G = -(\hat{y} - y)^2$
$\mu_{k,n-1} = \theta_{k,n-1}^T o_k$
$\pi_0(a_0|x)$
$\mu_{0,n} = \theta_{0,n}^T x$
$\pi_k(a_k|o_k)$
Coagent Parameterization for Continuous Nodes

Figure 1: Structural Credit Assignment within a Neural Network as a Finite Horizon RL problem. The **left** figure maps components in the network to observations, actions and rewards. The **right** figure highlights one agent in the network and how it might be parameterized.

For this interaction to be Markov, the computation of the final reward requires the input $\boldsymbol{x}$. 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. This part of the input can be ignored by the policy, as it is for most NN architectures. Formally, 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 $\mathcal{A} = \mathbb{R}^n$, where each start state $\boldsymbol{s}_0 = [\boldsymbol{x}, 0]$ and later states for $j \geq 1$ are $\boldsymbol{s}_j = [\boldsymbol{x}, \boldsymbol{a}_{j-1}, j]$. The transitions are deterministic, with a reward of zero on each step until termination at step k giving reward $r_{k+1} = -\text{err}(\hat{y}, y)$.

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 is using policy gradients, then 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, is

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

Because $\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, a_k|\boldsymbol{x}) = \pi(\boldsymbol{a}_0|\boldsymbol{x}) \prod_{j=1}^{k} \pi_j(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})$, the probability of this trajectory is $p(\boldsymbol{x})\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, a_k|\boldsymbol{x})p(y|\boldsymbol{x})$. This is because $p(\boldsymbol{x})$ gives the probability of the start state; $\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, a_k|\boldsymbol{x})$ gives the probability of the trajectory from s_0 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 $G \overset{\text{def}}{=} -\text{err}(a_k, y)$

$$\max_{\pi} \int p(\boldsymbol{x})\pi(\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, a_k|\boldsymbol{x})p(y|\boldsymbol{x})\; G \; d\boldsymbol{x}\; d\boldsymbol{a}_0\; \ldots\; da_k\; dy = \max_{\pi} \mathbb{E}_{\pi, p(\boldsymbol{x}, y)}[G] \tag{1}$$

If we assume each policy π_j has parameters $\boldsymbol{\theta}_j$, then the stochastic gradient of this objective separates out into the following stochastic gradients for each policy separately: $G\nabla_{\boldsymbol{\theta}_j} \ln \pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j)$. We show this formally in Proposition 1 in Appendix A. A similar result has been shown for CoANs used in the RL setting [24, Theorem 1] and for stochastic computation graphs [60, 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 control variates to reduce the variance of the gradient, called baselines. The update is $(G - V(\boldsymbol{o}_j))\nabla_{\boldsymbol{\theta}_j} \ln \pi_j(\boldsymbol{a}_j|\boldsymbol{o}_j)$, where the learned baseline $V(\boldsymbol{o}_j)$ estimates expected return given $\boldsymbol{o}_j$: $G - V(\boldsymbol{o}_j)$ corresponds to the advantage for the actions selected. Several different critics have been proposed for stochastic computation graphs [60]; we discuss how to make similar baselines for this finite horizon problem in Appendix B.

The CoAN is trained as a stochastic network, but is evaluated under the standard (deterministic) backprop setting. The question we ask is: can a CoAN, trained with local updates, produce predictions with similar accuracy to those produced by an NN, trained with backprop? We therefore test the

CoAN using the greedy actions from the co-agents, to provide deterministic predictions. Effectively, we are comparing the parameters $\boldsymbol{\theta}$ produced by the CoAN to those produced by backprop.[1]

Remark: We described the formalism for the simpler feedforward setting, to facilitate understanding. All the ideas, however, 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.

3 Issues with Stochasticity in On-policy Updates for CoANs

In this section, we show that though CoANs with REINFORCE can learn, they are significantly hindered by stochasticity in other coagents. Variance is a well-known problem in stochastic networks. Here, we highlight that even for a variety of variance reduction approaches, learning plateaus at a suboptimal point. The issue is less severe with discrete actions—binary rather than continuous activations—but a significant gap remains when contrasting to idealized (low-variance) gradients.

3.1 Experimental Details

We investigate CoANs on problems where backprop is known to perform well, to provide a strong baseline and facilitate understanding the behavior, and potential issues, when learning in CoANs. We expect backprop to outperform CoANs here, 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 [27] for classifying handwritten digits, and the Boston Housing Dataset from UCI.

Where possible, we matched the architecture and optimization choices for backprop and the CoAN learner. We test both strategies using a single and double-layer neural network, with 64 hidden nodes and ReLU activations. 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 $\sigma \in \{0.1, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0\}$. 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 [57], 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 10 independent runs and compared using the area under curve (AUC).

Along with the standard CoAN with no baseline, we also test the effectiveness of three baselines. The **Global baseline** (Co-G) maintains a scalar running average of the global loss function parameterized by η the rate of decay (fixed at 0.99). The **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. The **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. More details on baselines are in Appendix B and pseudocode in Appendix H.

3.2 Results

We report the performance of the best performing parameters on a held-out test set with the same size as the validation set in Figure 2 (a). At a glance, it is clear that backprop outperformed all the CoANs. It is surprising that there is no improvement using the local baselines as compared to the global baseline. A natural question from these results is if the gap between CoANs and backprop is due

---

[1] The optimal parameters for the (surrogate) policy gradient objective above may not be the same as those for the deterministic network. Namely, the co-agents minimize $\mathbb{E}[(\hat{Y} - y)^2]$ across pairs $(\boldsymbol{x}, y)$, where $\hat{Y}$ is stochastic due to the stochastic co-agent policies. This is in contrast to minimizing the deterministic output $(\mathbb{E}[\hat{Y}] - y)^2$ or the loss where each co-agent takes a greedy action. The co-agents may actually adjust their outputs, to mitigate risk from the stochasticity in fellow co-agents. Somewhat surprisingly, we find in our experiments that there is no such gap, and optimizing this surrogate can produce optimal solutions.

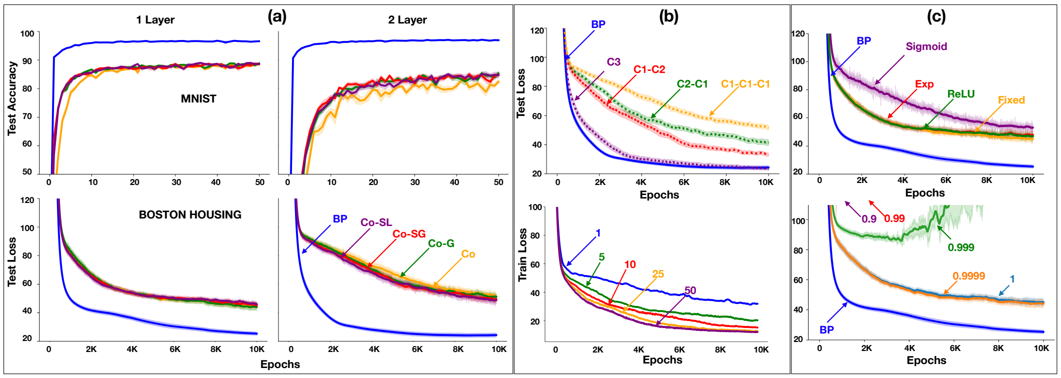

Figure 2: All curves are averaged for 10 runs with standard error bars. **(a)** Training different variants of CoAN's and backprop on MNIST and Boston Housing with one and two layer NNs. **(b)** Experiments highlighting issues with nondeterminism in coagents. The upper graph depicts performance with different coagent partitionings of the network. The lower graph depicts the training curve for 2 layer coagent network with C1-C1-C1 partitioning for different amounts of averaging i.e., $[1, 5, 10, 25, 50]$. **(c)** Failures of different strategies to gradually reduce nondeterminism of coagents overtime, on Boston Housing.

to the optimization process, or a poor stationary point of the CoAN itself. Experiments running the CoAN for longer showed only very slow improvement. 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.

To investigate the role of stochasticity in 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 C1). 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). 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 2 (b, upper), we see the deterministic network C3 performs similarly to the backprop network. As we add stochasticity back to intermediary layers, performance degrades significantly. This highlights that the stochasticity in the last layer is not a culprit, and that variance is lower for C1-C2 where stochasticity is in the first layer than for C2-C1 where stochasticity is in a later layer. In either case, the introduction of stochasticity in intermediate layers starkly reduces performance.

To ascertain that the issue is due to variance, rather than poor stationary points, we can use an idealized gradient that is not practical for our desired learning setting, but can act as a control for this experiment. We can obtain a low-variance gradient estimate simply by sampling the stochastic network many times, for the given pair $(\boldsymbol{x}, y)$. Such an update is non-local—requires significant coordination—and so is not a practical approach to reduce variance for our setting. But this averaged gradient, particularly with increasing samples for the average, gives insight into the optimization surface for these networks as well as the magnitude of the variance. Figure 2 (b, lower) shows the effect on learning with increasing samples used in the average gradient, for the two layer network. With increasing samples, the training of the CoAN significantly improves, nearly matching the learning speed of backprop with 50 samples. This highlights that the surrogate objective used by the CoAN results in reasonable solutions when testing the greedy actions given by the CoAN. It shows that there is significant variance in the gradient, and that baselines are not reducing that variance.

3.3 Failures of Simple On-policy Approaches to Reduce Nondeterminism

A natural next step is to consider simple strategies to gradually reduce the stochasticity of coagents. For example, it is common in RL to have a decay 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.

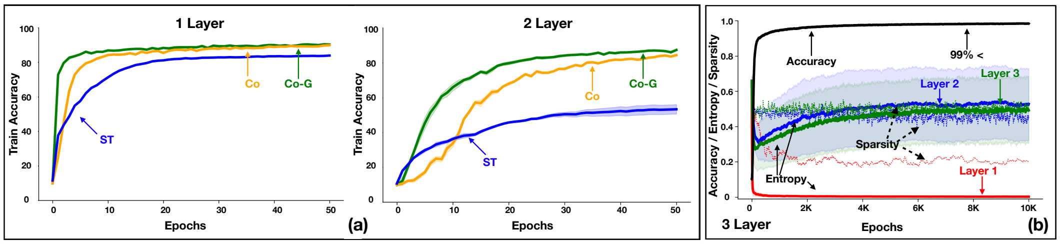

Figure 3: Experiments on MNIST, using 64 nodes per layer. **(a)** Learning curves for discrete agents (ST, Co, Co-G) with single and double-layers, averaged for 10 runs, for 50 epochs. **(b)** Learning progress of three layer network for 10K epochs, with entropy *(solid line)* and sparsity *(dotted line)* of each layer, along with accuracy (scaled to $[0, 1]$), which matches backprop (with accuracy $> 99\%$).

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 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 2 (c). One possible reason is that for smaller σ of 0.9 and 0.99 the gradient $\nabla_{\boldsymbol{\theta}} \ln \pi(\boldsymbol{a}|\boldsymbol{o})$ can get very big for low probability sampled actions. This instability can be avoided with discrete actions, or with off-policy learning, both of which we investigate next.

3.4 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 as a softmax parameterization adapts stochasticity when learning, and can become fully deterministic. As a baseline, we do gradient backpropagation via straight-through (ST) estimators [8; 50], 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, as well as the sparsity of the representation to ensure reasonable levels of activation.

In Figure 3(a) we can see that CoANs actually perform better than the baseline, the ST estimator, and here the baseline had a bigger positive effect (Co-G versus Co). ST estimators have difficulties when learning in more than a single layer [8], probably due to misalignment [50]. For Co-G, the ability to better control the entropy seems to help: in Figure 3(b) the drops in entropy—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. 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 slightly improving the accuracy. We note that these networks are able to match the performance of a standard NN with backprop (i.e., accuracy $> 99\%$) but it takes a very large number of training steps (approximately $10K$ epochs).

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, in Appendix C. Overall, we find that the former significantly increases performance in 1-layer networks for both REINFORCE coagents and especially action-value methods; however, the latter approach does not substantially reduce the entropy, nor improve performance.

4 Off-Policy Learning to Learn Nearly Deterministic 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. 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 in MNIST for the case of discrete coagents. We provide additional results in Appendix F showing that on-policy action-value methods do not improve on REINFORCE, further motivating the shift to off-policy learning.

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 $\boldsymbol{a}_{j-1}$. 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 $[\boldsymbol{a}_{j-1}, \boldsymbol{a}_j]$—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 co-agent. This nonstationarity and difficulties in credit assignment is well-recognized as an issue in multi-agent reinforcement learning [54; 15; 43]. 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 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.

The coagents do not need to reason about the greedy actions for upstream coagents, because action-values are *conditioned* on inputs produced by those coagents. For a given activation from the previous layer $\boldsymbol{a}_{j-1}$, the coagent learns $\pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})$. Under sufficiently high capacity policy parameterizations, the coagent can simply learn what to output for a variety of different inputs, including those that are more optimal for the given input $\boldsymbol{x}$. It is straightforward to show that in Equation (1), under unrestricted policy parameterizations, each policy can be optimized assuming these greedy action-values for downstream layers.

Practically, however, our coagent policies are likely to be under-parameterized. If an upstream coagent provides a wide range of activations $\boldsymbol{a}_{j-1}$, then the policy has to trade-off function approximation across this large input space. Ideally, it would focus function approximation on the $\boldsymbol{a}_{j-1}$ that are likely to produce good predictions $\hat{y}$. This trade-off is a common issue in policy gradient methods. We can incorporate a reweighting, to increase the importance of an update for an $\boldsymbol{a}_{j-1}$ that is more probable—more greedy—from the upstream coagent, and decrease the importance of an update for lower probability $\boldsymbol{a}_{j-1}$. Such importance is already implicit, because $\boldsymbol{a}_{j-1}$ are sampled proportionally to the probabilities. For this reason, we do not address it further here, but note that there are some insights for state reweightings in policy gradient methods to improve solutions [21; 19].

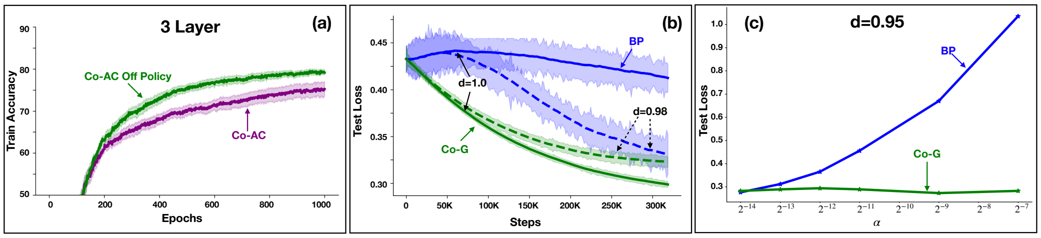

Figure 4: **(a)** Learning curves comparing on policy and off policy actor critic. Averaged for 10 runs, for 3 layer neural network with 64 nodes per layer. **(b)** CL experiments with problem difficulties at $d = 0.98$ *dashed* lines and *solid* lines for $d = 1.0$ comparing backprop and CoAN's. **(c)** Sensitivity plots in CL problem for learning rate (α), for problem difficulty of $d = 0.95$.

4.2 Results with Off-Policy CoANs

We perform a set of experiments where we estimate $Q_j(\boldsymbol{o}_j, \boldsymbol{a}_j)$ function for each layer in the CoAN, using the on-policy and off-policy approaches described in Section 4.1. 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 3.4. Note that here our goal is to understand if critics can help, and so allow for these larger critics per coagent; practically, depending on the setting, there may be stronger limitations on the critics.

Figure 4 (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. 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, to highlight that off-policy learning is a promising direction towards addressing the nondeterminism issue in CoANs.

5 Continual Learning Experiments

In the previous sections, we chose settings where backprop is effective, to make the results more interpretable and start from the standard learning setting. The motivation for CoANs, however, goes much beyond the standard setting. The goal is to facilitate learning in other settings, particularly in continual learning settings with correlated data and the need for real-time learning and decision-making. CoANs also naturally facilitate asynchronous inputs and updating [24], as well as learning with recurrence. Traditionally stochastic methods have known to regularize and help avoid local minima; we provide a small experiment showing this advantage of CoANs in Appendix D.

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. Because coagents are RL agents, the policies should better track and adapt to changes in the environment.

To measure coagents' 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 [41], with the same parameter settings and target function. 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.

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 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.

Backprop outperforms Co-G on iid samples, and 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 learns steadily, while backprop is incapable of learning. In addition, the sensitivity plot for this experiment in Figure 4 (c) 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. Co-G is also similarly insensitive to its σ hyperparameter, as shown in the Appendix E.

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. These results, however, are for carefully swept hyperparameters, and we 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.

6 Limitations and Next Steps

Once we have formalized this problem as a finite-horizon RL problem, it facilitates applying a variety of RL algorithms. The approaches in this work are arguably the simplest first choices, and do not incorporate useful advances in exploration strategies, policy gradient methods, off-policy learning and replay approaches. Our CoANs relied solely on the stochasticity in the coagents to explore. Such randomized exploration strategies are unlikely to be as efficient as directed exploration approaches, even when restricted to only those that learn values and not models, such as those using upper confidence bounds on values [16; 40; 25] and information-directing sampling [45]. Such directed exploration strategies have also recently been developed for policy gradient methods [2].

There have also been many recent advances to policy gradient methods, with better theoretical understanding and improvements for the off-policy setting. Of particular relevance is work that examines the importance of state weightings and dealing with distribution shift [3], which is key for the off-policy setting [19]. Another important insight is leveraging connections between approximate policy iteration and policy gradient methods, to obtain performance guarantees [12; 1; 59; 10]. This perspective assumes that an explicit policy and explicit values are learned, and facilitates off-policy learning using the learned values. One of these methods, called SBEED [12], further exploits recent advances in gradient-based methods for learning values off-policy.

All of these more advanced approaches rely on value estimation. Value estimates help direct exploration, by facilitating reasoning about uncertainty and incorporating optimism. They naturally facilitate off-policy learning and replay, because they allow us to learn from short experience tuples. They allow each agent to reason about counterfactual outcomes, and perform more policy updates in the background, asynchronously. The bias, however, currently precludes obtaining these benefits; we as yet do not know why it is so harmful. When using Actor-Critic in other settings, the bias in the critic does not seem to be so detrimental. One issue here may be that PG methods have not been specially designed for finite-horizon problems, where a different policy is used each step. For us, the changing policies make it less straightforward to use standard bias-reduction strategies to estimate critics, like eligibility traces. Understanding the influence of bias in the critic for structural credit assignment, and how to overcome it, is one of the most important open questions for CoANs.

7 Conclusion

In this paper, 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 highlight that the stochasticity amongst the coagents results in this suboptimality, and that it cannot be mitigated with standard variance reduction strategies or attempts to gradually reduce stochasticity of the coagents towards deterministic policies. 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, but that much more work needs to be done to improve the learned critics and mitigate bias.

Acknowledgments and Disclosure of Funding

This work was supported by NSERC Discovery, IVADO, CIFAR through CCAI Chair funding and by the Alberta Machine Intelligence Institute (Amii). This research was funded in part by NSF award #2018372. The authors would also like to thank Kirby Banman.

References

[1] Yasin Abbasi-Yadkori, Peter Bartlett, Kush Bhatia, Nevena Lazic, Csaba Szepesvari, and Gellert Weisz. POLITEX: Regret Bounds for Policy Iteration using Expert Prediction. In *International Conference on Machine Learning*, 2019.

[2] Alekh Agarwal, Mikael Henaff, S. Kakade, and Wen Sun. PC-PG: Policy Cover Directed Exploration for Provable Policy Gradient Learning. In *Advances in Neural Information Processing Systems*, 2020.

[3] Alekh Agarwal, Sham M. Kakade, Jason D. Lee, and Gaurav Mahajan. On the Theory of Policy Gradient Methods: Optimality, Approximation, and Distribution Shift. *arXiv:1908.00261*, 2020.

[4] Adrian K. Agogino and Kagan Tumer. Unifying temporal and structural credit assignment problems. In *Proceedings of the International Joint Conference on Autonomous Agents and Multi-Agent Systems*, 2004.

[5] David Balduzzi, Hastagiri Vanchinathan, and Joachim Buhmann. Kickback cuts backprop's red-tape: Biologically plausible credit assignment in neural networks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 2015.

[6] Emmanuel Bengio, Pierre-Luc Bacon, Joelle Pineau, and Doina Precup. Conditional computation in neural networks for faster models. *arXiv:1511.06297*, 2015.

[7] Yoshua Bengio. How Auto-Encoders Could Provide Credit Assignment in Deep Networks via Target Propagation. *arXiv:1407.7906*, 2014.

[8] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or Propagating Gradients Through Stochastic Neurons for Conditional Computation. *arXiv:1308.3432*, 2013.

[9] Miguel Carreira-Perpinan and Weiran Wang. Distributed optimization of deeply nested systems. In *Artificial Intelligence and Statistics*, 2014.

[10] Alan Chan, Hugo Silva, Sungsu Lim, Tadashi Kozuno, A. Rupam Mahmood, and Martha White. Greedification Operators for Policy Optimization: Investigating Forward and Reverse KL Divergences. *arXiv:2107.08285*, 2021.

[11] Yu-Han Chang, Tracey Ho, and Leslie Pack Kaelbling. All learning is local: Multi-agent learning in global reward games. In *Advances in Neural Information Processing Systems*, 2003.

[12] Bo Dai, Albert Shaw, Lihong Li, Lin Xiao, Niao He, Zhen Liu, Jianshu Chen, and Le Song. SBEED: Convergent Reinforcement Learning with Nonlinear Function Approximation. In *International Conference on Machine Learning*, 2018.

[13] Ludovic Denoyer and Patrick Gallinari. Deep Sequential Neural Network. *arXiv:1410.0510*, 2014.

[14] Ila R. Fiete and H. Sebastian Seung. Gradient Learning in Spiking Neural Networks by Dynamic Perturbation of Conductances. *Physical Review Letters*, 2006.

[15] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[16] Robert C Grande, Thomas J Walsh, and Jonathan P How. Sample Efficient Reinforcement Learning with Gaussian Processes. In *International Conference on Machine Learning*, 2014.

[17] Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. In *International Conference on Learning Representations*, 2016.

[18] Geoffrey E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 2002.

[19] Ehsan Imani, Eric Graves, and Martha White. An Off-policy Policy Gradient Theorem Using Emphatic Weightings. In *Advances in Neural Information Processing Systems*, 2018.

[20] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. In *International Conference on Machine Learning*, 2017.

[21] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

[22] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[23] A. Harry Klopf. *The Hedonistic Neuron: A Theory of Memory, Learning, and Intelligence*. Hemisphere Publishing Corporation, 1982.

[24] James Kostas, Chris Nota, and Philip Thomas. Asynchronous coagent networks. In *International Conference on Machine Learning*, 2020.

[25] Raksha Kumaraswamy, Matthew Schlegel, Adam White, and Martha White. Context-dependent upper-confidence bounds for directed exploration. *Advances in Neural Information Processing Systems*, 2018.

[26] Benjamin James Lansdell, Prashanth Ravi Prakash, and Konrad Paul Körding. Learning to solve the credit assignment problem. In *International Conference on Learning Representations*, 2020.

[27] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[28] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, 2015.

[29] Robert Legenstein, Dejan Pecevski, and Wolfgang Maass. A Learning Theory for Reward-Modulated Spike-Timing-Dependent Plasticity with Application to Biofeedback. *PLOS Computational Biology*, 2008.

[30] Robert Legenstein, Steven M. Chase, Andrew B. Schwartz, and Wolfgang Maass. A Reward-Modulated Hebbian Learning Rule Can Explain Experimentally Observed Network Reorganization in a Brain Control Task. *Journal of Neuroscience*, 2010.

[31] Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature Communications*, 2016.

[32] Henry Markram, Wulfram Gerstner, and Per Jesper Sjöström. Spike-Timing-Dependent Plasticity: A Comprehensive Overview. *Frontiers in Synaptic Neuroscience*, 2012.

[33] Thomas Merkh and Guido Montúfar. Stochastic Feedforward Neural Networks: Universal Approximation. *arXiv:1910.09763*, 2019.

[34] Thomas Miconi. Biologically plausible learning in recurrent neural networks reproduces neural dynamics observed during cognitive tasks. *eLife*, 2017.

[35] Javier Movellan. Contrastive Hebbian Learning in the Continuous Hopfield Model. *Connectionist Models*, 1991.

[36] Somjit Nath, Vincent Liu, Alan Chan, Xin Li, Adam White, and Martha White. Training recurrent neural networks online by learning explicit state variables. In *International Conference on Learning Representations*, 2020.

[37] Radford M. Neal. Learning stochastic feedforward networks. Technical report, 1990.

[38] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, 2016.

[39] Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 2008.

[40] Ian Osband, Benjamin Van Roy, Daniel Russo, and Zheng Wen. Deep Exploration via Randomized Value Functions. *Journal of Machine Learning Research*, 2019.

[41] Yangchen Pan, Kirby Banman, and Martha White. Fuzzy tiling activations: A simple approach to learning sparse representations online. In *International Conference on Learning Representations*, 2021.

[42] Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. Techniques for learning binary stochastic feedforward neural networks. In *International Conference on Learning Representations*, 2015.

[43] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. In *Proceedings of the International Conference on Machine Learning*, 2018.

[44] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[45] Daniel Russo and Benjamin Van Roy. Learning to Optimize via Information-Directed Sampling. In *Advances in Neural Information Processing Systems*, 2014.

[46] Hiroshi Saito, Kentaro Katahira, Kazuo Okanoya, and Masato Okada. Statistical mechanics of structural and temporal credit assignment effects on learning in neural networks. *Physical Review. E, Statistical, Nonlinear, and Soft Matter Physics*, 2011.

[47] Benjamin Scellier and Yoshua Bengio. Equilibrium propagation: Bridging the gap between energy-based models and backpropagation. *Frontiers in Computational Neuroscience*, 2017.

[48] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, 2015.

[49] Alexander Shekhovtsov and Viktor Yanush. Reintroducing Straight-Through Estimators as Principled Methods for Stochastic Binary Networks. *arXiv:2006.06880*, 2021.

[50] Alexander Shekhovtsov, Viktor Yanush, and Boris Flach. Path sample-analytic gradient estimators for stochastic binary networks. In *Advances in Neural Information Processing Systems*, 2020.

[51] Ming Tan. Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents. In *International Conference on Machine Learning*, 1993.

[52] Yichuan Tang and Ruslan Salakhutdinov. Learning stochastic feedforward neural networks. In *Advances in Neural Information Processing Systems*, 2013.

[53] Amirhossein Tavanaei, Masoud Ghodrati, Saeed Reza Kheradpisheh, Timothee Masquelier, and Anthony S. Maida. Deep Learning in Spiking Neural Networks. *Neural Networks*, 2019.

[54] Gerald Tesauro. Extending Q-Learning to General Adaptive Multi-Agent Systems. In *Advances in Neural Information Processing Systems*, 2003.

[55] Philip S Thomas. Policy Gradient Coagent Networks. In *Advances in Neural Information Processing Systems*, 2011.

[56] Philip S. Thomas and Andrew G. Barto. Conjugate markov decision processes. In *Proceedings of the International Conference on Machine Learning*, 2011.

[57] T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[58] Vivek Veeriah, Shangtong Zhang, and Richard S. Sutton. Crossprop: Learning representations by stochastic meta-gradient descent in neural networks. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD*, 2017.

[59] Nino Vieillard, Tadashi Kozuno, Bruno Scherrer, Olivier Pietquin, Remi Munos, and Matthieu Geist. Leverage the Average: an Analysis of KL Regularization in Reinforcement Learning. In *Advances in Neural Information Processing Systems*, 2020.

[60] Théophane Weber, Nicolas Heess, Lars Buesing, and David Silver. Credit Assignment Techniques in Stochastic Computation Graphs. In *The International Conference on Artificial Intelligence and Statistics*, 2019.

[61] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 1992.

[62] David H Wolpert and Moffett Field. Optimal Payoff Functions for Members of Collectives. *Modeling complexity in economic and social systems*, 2002.

[63] David H. Wolpert, Kevin R. Wheeler, and Kagan Tumer. General principles of learning-based multi-agent systems. In *Proceedings of the Annual Conference on Autonomous Agents*, 1999.

A Policy Gradient Theorem for Structural Credit Assignment in CoANs

Define $G \overset{\text{def}}{=} -\text{err}(a_k, y)$ and the policy gradient objective

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

where each policy π_j has parameters $\boldsymbol{\theta}_j$ and $\boldsymbol{\theta} = [\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_k]$.

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

$$\nabla_{\boldsymbol{\theta}_j} J(\boldsymbol{\theta}) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}, p(\boldsymbol{x}, y)}[G \nabla_{\boldsymbol{\theta}_j} \ln \pi_j(A_j | A_{j-1})]$$

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

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

Further we can write the derivative log term as:

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

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.

$\square$

B Baselines and Critics for CoANs

There has been quite a lot of development on variance reduction approaches for stochastic computational graphs (SCGs) [60]. 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 a consist of all actions outputted by the agents, and $Q(\boldsymbol{x}, a) = \mathbb{E}[-\text{err}(\hat{y}, Y)|X = \boldsymbol{x}]$ 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 $Q(\boldsymbol{x}, \boldsymbol{a}_{-j}) = \sum_{\boldsymbol{a}_j} \pi(\boldsymbol{a}_j|\boldsymbol{a}_{j-1})Q(\boldsymbol{x}, [\boldsymbol{a}_{-j}, \boldsymbol{a}_j])$, the expected value for policy j, assuming the other policies had fixed values. Then we can use $G - Q(\boldsymbol{x}, \boldsymbol{a}_{-j})$ for the update to π_j, where if this quantity is greater than zero the action probability is increased and other is decreased.

Once we have $Q(\boldsymbol{x}, a)$, 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. The joint action-values are a sufficient model, where the RL algorithm uses exploration to identify a good joint policy. This is much lower variance than only using the return, as the agent can directly reason about the outcomes of all actions jointly and removes 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.

There is, however, a more fundamental problem for us, as in decentralized learning, we cannot use $Q(\boldsymbol{x}, a)$ nor the baseline $Q(\boldsymbol{x}, \boldsymbol{a}_{-j})$. 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 taking into account the role of their action. This baseline essentially just centers the updates, so that the gradient updates directly increase probability of actions that resulted in $G - V(\boldsymbol{x}) > 0$, and decrease the probability of those that do not.

Such a baseline, however, is a minimal optimization improvement. Instead, we can also learn local baselines and critics. For each policy π_j, we can learn a simple linear baseline $V_j(s = \boldsymbol{o}_j) = \langle \boldsymbol{o}_j, \boldsymbol{\theta^{(v)}}_j \rangle$ and critic $Q_j(s = \boldsymbol{o}_j, \boldsymbol{a}_j) = \langle [\boldsymbol{o}_j, \boldsymbol{a}_j], \boldsymbol{\theta^{(q)}}_j \rangle$ using Monte carlo. Remember that observation for current agent is actions from previous layer i.e. $\boldsymbol{o}_j = [\boldsymbol{a}_{j-1}]$.

$$\begin{aligned}
\boldsymbol{\theta^{(v)}}_j &\leftarrow \boldsymbol{\theta^{(v)}}_j + \alpha(G - \langle \boldsymbol{o}_j, \boldsymbol{\theta^{(v)}}_j \rangle)\boldsymbol{o}_j \\
\boldsymbol{\theta^{(q)}}_j &\leftarrow \boldsymbol{\theta^{(q)}}_j + \alpha(G - \langle [\boldsymbol{o}_j, \boldsymbol{a}_j]^\top, \boldsymbol{\theta^{(q)}}_j \rangle)[\boldsymbol{o}_j, \boldsymbol{a}_j]^\top
\end{aligned}$$

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 only a baseline and a baseline and critic, learned with Monte Carlo. When only using a baseline, the updates are of the form

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

and when using an action-value critic, the update is

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

We use the following three simple baselines.

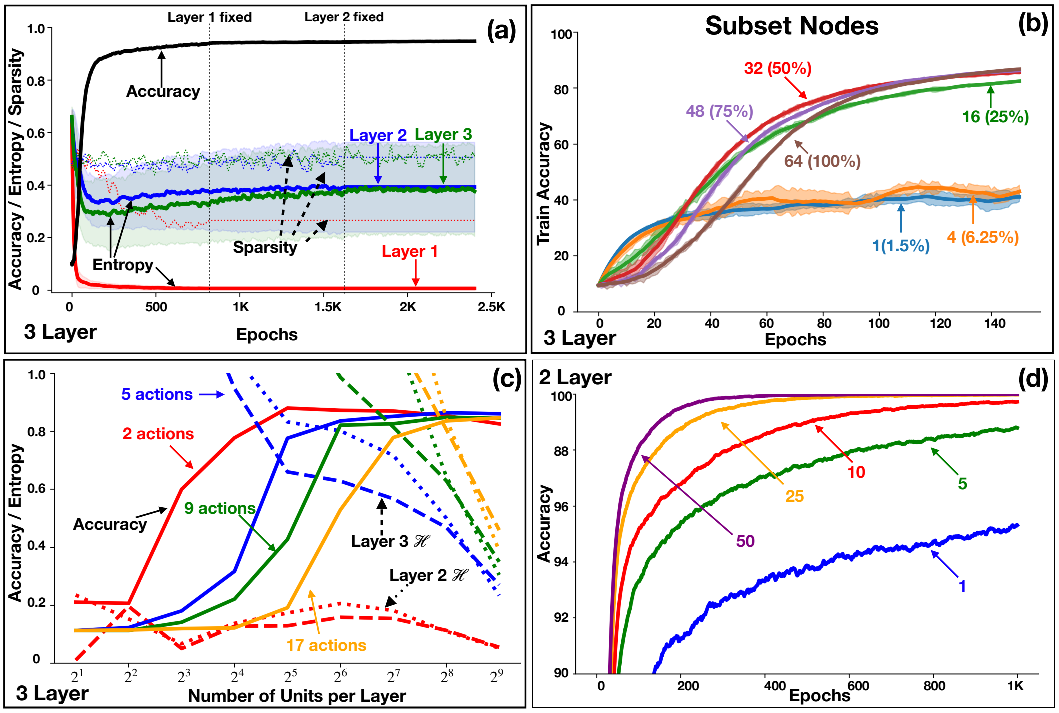

Figure 5: All experiments in this figure are on MNIST dataset, with experiment (a,b) use 64 nodes per layer. **(a)** Inter layer experiment : fixing earlier layers at regular intervals for a three layer network for 2.4K epochs (fixing at 800 and 1.6K epochs). Shows sparsity and entropy similar to Figure 3 (b). **(b)** Intra layer experiment : training a random fraction of subset of nodes in each layer. Shown for 150 epochs, averaged for 10 runs. **(c)** Representation experiment : sensitivity curves for different number of nodes per layer $\in \left[2^1, \ldots, 2^9\right]$ 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. **(d)** Averaging Gradients : shows the effect of averaging on performance of binary / discrete nodes.

- **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. $V(\boldsymbol{x})$.

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

- **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. $V_j(\boldsymbol{o}_j)$.

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

C Investigating Suboptimality of Discrete Agents

As we saw in section 3.4, stochasticity in intermediate layers don't decrease, even though the coagents match backprop in terms of accuracy if trained for long enough (i.e. > 99 % training data). This raises the question of why this happens. There can be multiple reasons, which include stochasticity in earlier layers not giving a stationary representation to the later layers, too many interacting nodes in a single layer, representation in intermediate layers being relatively weak to support deterministic policies. We study these properties in our subsequent experiments.

Figure 5(a,b,c) 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 the 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 experiment we train a subset of randomly selected subset of 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. Counter to our intuition of earlier layer being more stochastic we observe that layer 1 have a sharp decrease in its entropy , which might be because of a strong representation it gets in form of images. To test this we experiment with different number of nodes ($\in \left[2^1, \ldots, 2^9\right]$) in each layer. 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 5(c) , more number of nodes in intermediate layer do seem to help with entropy of those layers. Whereas 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 its better to have a bigger and maybe sparse representation. We observe some counter to the number of outputs in the case of action value case which we discuss in more detail in Appendix F.

Figure 5 (d) shows the effect of different amounts of averaging of gradients for the discrete nodes. Having better estimates of gradient allows for faster convergence to a good solution from backprop i.e. from $5 - 6K$ epochs (1 averaging) to about 500 epochs (50 averaging).

D 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.

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 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 to compare 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 [22] (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.

We report the performance of the different learning methods in Figure 6. 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 better than 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 6 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.

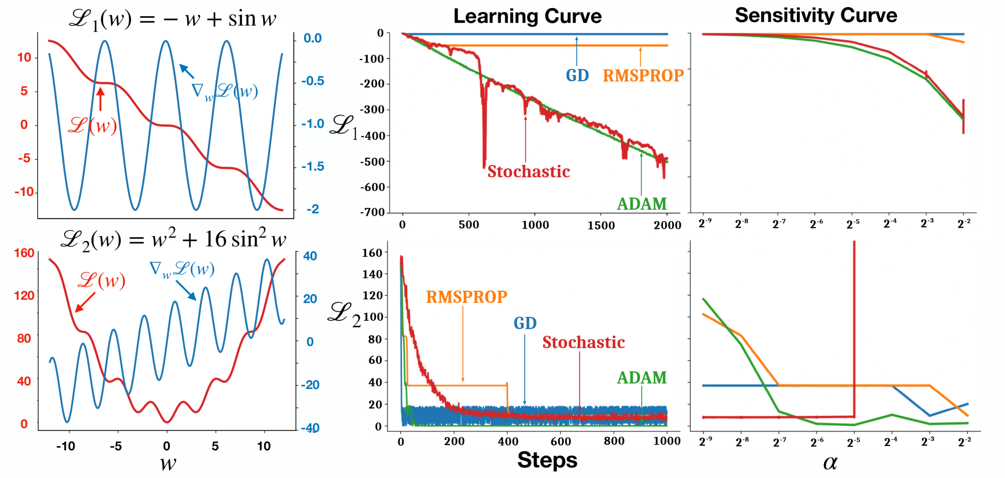

Figure 6: First column depicts the loss curve and corresponding derivative with respect to the scalar weight of both problems. Second column corresponds to the learning curve followed by the sensitivity of α for all the methods. The stochastic node is averaged over 50 runs.

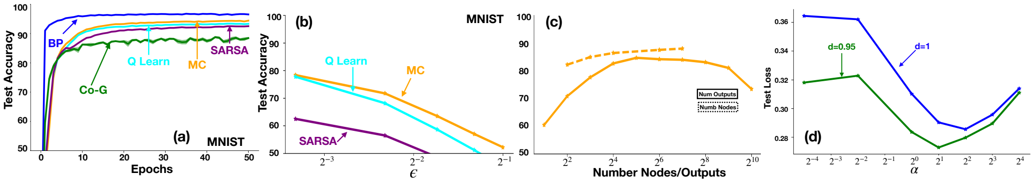

Figure 7: (a) learning curve of the Monte-Carlo, Q-learning and Sarsa agents with backprop and the continuous Co-G algorithm on MNIST for the first 50 epochs, results averaged over 10 runs. In action-value methods, the number of units per layer and number actions per unit are swept in $\{2^1, 2^2, \ldots, 2^6\}$. For all algorithms the stepsize is swept $\alpha \in \{2^{-7}, 2^{-9}, 2^{-11}, 2^{-13}, 2^{-14}\}$ for Co-G $\sigma \in \{0.1, 0.5, 1.0, 2.0, 4.0, 8.0, 16.0\}$. For the same experiment, (b) shows the effect of varying a fixed ϵ-value, while (c) depicts the effects of changing the number of units in a layer and the number of actions of each unit in the Monte Carlo-agent on MNIST in a more extensive sweep, with results averaged over 10 runs. (d) displays the error of the continuous Co-G agent on the correlated dataset with difficulties $d \in \{0.95, 1\}$ varying the σ-values, with results averaged over 200 runs.

E Continual Learning Experiments

Dataset description

As in [41], we create a synthetic supervised problem with target function $y_t = sin(2\pi x_t^2)$. The Piecewise Random Walk Problem produces datapoints centred around a specified number of means, with each datapoint occurring n times, with n being the specified segment length of the dataset. The parameter d controls the difficulty, with 0 being an iid dataset and 1 standing for a dataset with maximum possible temporal correlation that still maintains the stationary distribution. Due to Theorem 2 of [41], the stationary distribution remains identical across all d values, hence the different difficulties are comparable.

Our first experiment with 180,000 training steps and an extensive hyperparameter sweep consists of 150 different means and 1200 segments with d $\in \{0, 0.85, 0.95, 1\}$. The subsequent experiment with 320,000 training steps comprises of 200 different mean values and 1600 segments, with d $\in \{0.98, 1\}$.

Figure 7 (d) shows the sensitivity curve of Co-G wrt the standard deviation parameter on the correlated dataset with 180,000 training steps with difficulties d=0.95 and d=1; and demonstrates that similarly to the step-size, Co-G is not sensitive to the standard deviation on either difficulties.

F Effects of the Number of Outputs, the Number of Coagents and Exploration in Action-Value CoANs

Action-value methods are a natural choice for discrete-action networks. We experiment with three action-value variants of the PG-methods evaluated in the main paper, i.e. Monte Carlo (MC), the action-value counterpart to the REINFORCE-style CoAN policy gradient (Co-PG) variants, SARSA, the action-value version of the on-policy actor-critic-style Co-PG, and Q-Learning, the action-value equivalent to the off-policy actor-critic-style Co-PG. Similarly to our observation in PG methods, we can observe that off-policy learning when bootstrapping can help us reduce the gap between MC methods, as seen in Figure 7 (a,b), where Q-learning outperforms SARSA.

The number of units in a layer is a highly relevant number for future implementations as each coagent could be run in an asynchronous distributed way, therefore we investigated whether increasing this number is beneficial for improving accuracy. Figure 7 (c) shows that if we fix the number of actions at four, and increase the number of units per layer, we get consistently better results. Conversely, if we fix the number of units per layer at 4, the agents achieve higher accuracy as we increase the number of actions until 32, after which performance degrades. Action-value methods achieve their best performance with 64 units per layer and 64 actions per unit on MNIST within 50 epochs.

Action-value CoANs are sensitive to ϵ-values as well. As Figure 7 (b) demonstrates, the higher the fixed ϵ-value, the lower accuracy the algorithm is capable of achieving.

G Extra details on experiment

G.1 Implementation difference between Continous and Discrete PG agents

In the case of contiunous agents, as the outputs are real valued we can treat each layer of the network as an agent and hence be a Gaussian distribution. Whereas in the case of discrete coagents, as the outputs of agents are 0 and 1 values, we can't exactly 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 agents which transforms the binary input $\boldsymbol{o}_k$ to real valued actions, and uses backprop to update this linear layer, but still uses the CoAN update for the agents in all the other layers.

As for more details, each coagent in discrete case has $|\mathcal{A}|$ number of heads. The outputs of these heads are fed through a softmax activation and then actions are sampled using a Bernoulli distribution. Each action either 0 or 1 appropriately.

For the case of off-policy agents, we need to do more evaluations as opposed to on-policy case. We need to do a forward pass on the next layer coagents with a greedy policy to acquire their set of greedy actions. We use the state and greedy actions for the next layer coagents to obtain the target for current layer and its critic.

G.2 Asset details for experiments

The code for the following experiments has been implemented using the Python (3.6) programming language with the opensource available tensor manipulation and auto grad package of PyTorch (v1.7.1). Also a lot of the mathematical and data manipulation techniques have been handled using the Numpy(v.1.19.4) and Scikit-learn (v0.24.0) packages. Datasets used in this paper are all open sources datasets with the appropriate links provided in the main paper. The figures provided in the paper have been crafted using the Matplotlib (v3.3.3) library. We also used the Anaconda package manager (v4.9.2) for installing and setting up different environments. Code used to produce the results and the experiments have been provided in submission.

G.3 Compute requirements

The experiments were executed on a computer equipped with 2 Intel Platinum 8260 Cascade Lake @ 2.4Ghz processors (i.e. 48 core and thread count) with 187 GBs of memory. The experiments in the main text took a total compute of around 2.5 CPU core years [2]. The experiments in the supplementary material took compute of about 6 CPU core years.

---

[2] A single CPU core computing for a full year (i.e. 365 days) or running the above machine for around 7.5 days continuously at full capacity.

H Pseudo Codes

H.1 On Policy PG method

We present the pseudocode for REINFORCE based methods 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.

These codes are provided for the continuous coagents, extension to binary nodes is simple, by replacing each agent with $|\mathcal{A}|$ heads, and using a softmax activation with a Bernoulli distribution to sample actions instead of a Gaussian. Also the last softmax layer for classification for discrete agents is just a usual linear layer without a Gaussian parameterization unlike continuous agents.

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 : $b$.
Initialize: Gaussian policy for all $n \times k$ coagents, with linear funciton approximation for $\mu_{i,j}$ and a fixed $\sigma$, paramterized by $\theta_{i,j}$.
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 $\phi_i$
Input Param : $\alpha$ - stepsize for gradient descent, of both coagent policies and value functions.
repeat
  $x, y \sim \mathcal{D}$ batch of size $b$
  $o_0 = x$
  // Forward Pass
  for $i = 0$ to $k - 1$ do
    for $j = 0$ to $n - 1$ do
      $a_{i,j} \sim \mathcal{N}(\mu_{i,j}(o_i), \sigma)$
      $u_{i,j} = \text{ReLU}(a_{i,j})$
    end for
    // Build next coagent layer state
    $o_{i+1} = (u_{i,0}, \dots, u_{i,n-1})$
  end for
  $\hat{y} \sim \mathcal{N}(f(o_k)), \sigma)$
  $\hat{y} = \text{softmax}(\hat{y})$
  $\mathcal{L} = \text{CrossEntropy}(y, \hat{y})$
  // Backward Pass
  for $i = k$ to $0$ do
    for $j = 0$ to $n - 1$ do
      // Update coagents
      $g_{i,j} = (\mathcal{L} - v_i(o_i))\nabla_{\theta_{i,j}} \log(\pi_{i,j}(a_{i,j}|o_i, \sigma))$
      $\theta_{i,j} -= \alpha g_{i,j}$
    end for
    // Update baselines
    $g_i^v = -2 * (\mathcal{L} - v_i(o_i))\nabla_{\phi_i} v_i(o_i)$
    $\phi_i -= \alpha g_i^v$
  end for
until Number of $e$ epochs achieved
```

H.2 Off-Policy PG method

Below is the pseudo code for the off policy variant of the CoAN with binary representation, for classification, note the use of CrossEntropy loss instead of the MSE loss.

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 funciton approximation paramterized by $\theta_{i,j}$.
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 $\omega_i$
Input Param : $\alpha$ - stepsize for gradient descent, for all components
repeat
  $x, y \sim \mathcal{D}$ batch of size $b$
  $o_0 = x$
  // 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
    $a_i = (a_{i,0}, \dots a_{i,n-1})$
    $o_{i+1} = a_i$
  end for
  $\hat{y} = f(\boldsymbol{o}_k)$
  $\mathcal{L} =$ CrossEntropy$(y, \hat{y})$
  $q_{k+1}(.) = \mathcal{L}$
  // Backward Pass
  for $i = k$ to $0$ do
    // Update critics with target policy
    $a_{i+1} \leftarrow \pi_i^{\text{greedy}}(\boldsymbol{o}_{i+1})$
    $Q_{tar} = q_{i+1}(o_{i+1}, a_{i+1})$
    $g_i^q = -2 * (Q_{tar} - q_i(o_i, a_i))\nabla_{\omega_i} q_i(o_i, u_i s)$
    $\omega i -= \alpha g_i^q$
    for $j = 0$ to $n - 1$ do
      $g_{i,j} = (Q_{tar})\nabla_{\theta_{i,j}} \ln(\pi_{i,j}(a_{i,j}|o_i))$
      $\theta_{i,j} -= \alpha g_{i,j}$
    end for
  end for
until Number of $e$ epochs achieved
```

H.3 Bootstrapping CoANs

Below are the pseudocode implementations of the action-value CoAgent Network algorithms as discussed in section F.

Algorithm 3 Q-learning CoAN for classification

```
Input: Number of coagents per layer $n$ , Number of layers $k$, Number of outputs $p$ feature size $d$ and $\mathcal{D}$ be the dataset distribution. Number of epochs $e$, Batch Size : $b$.
Initlialize: Final layer called as $f$, parameterized by $\theta$.
Initlialize : $k+1$ Linear value functions $(q_0, \dots q_k)$, one for each layer, parameterized by $\phi_i$
Input Param : $\alpha$ - stepsize for gradient descent, for all components
repeat
  $x, y \sim \mathcal{D}$ batch of size $b$
  $o_0 = x$
  // Forward Pass
  for $i = 0$ to $k-1$ do
    for $j = 0$ to $n-1$ do
      $a_{i,j} \sim \epsilon - greedy(q_i(o_i, \phi_i))$
      $u_{i,j} = a_{i,j}$
    end for
    // Build next coagent layer state
    $o_{i+1} = (u_{i,0}, \dots, u_{i,n-1})$
  end for
  $\hat{y} \sim \epsilon$-greedy$(f(o_k)), \theta)$
  $\hat{y} =$ softmax$(\hat{y})$
  // Backward Pass
  $\mathcal{L}_{final} =$ CrossEntropy$(y, \hat{y})$
  $\theta - = \alpha \nabla \mathcal{L}_{final}$
  for $i = k-1$ to $0$ do
    // Update coagents
    for $j = 0$ to $n-2$ do
      $g_{i,j} = -MSE(q_i(o_i, a_{i,j}), (-\mathcal{L}_{final} + max(q_i(o_i, a_{i,j+1})))/2)$
      $\phi_{i,j} - = \alpha \nabla g_{i,j}$
    end for
    if $j == n-2$ and $i == k-1$ then
      $j = j+1$
      $g_{i,j} = -MSE(q_i(o_i, a_{i,j}), -\mathcal{L}_{final})$
      $\phi_{i,j} - = \alpha \nabla g_{i,j}$
    end if
    if $j == n-2$ and $i \neq k-1$ then
      $j = j+1$
      $g_{i,j} = -MSE(q_i(o_i, a_{i,j}), (-\mathcal{L}_{final} + max(q_{i+1}(o_{i+1}, a_{i+1,0})))/2)$
      $\phi_{i,j} - = \alpha \nabla g_{i,j}$
    end if
  end for
until Number of $e$ epochs achieved
```

The algorithm for the Sarsa CoAN is identical to Q-learning, except instead of the maximum Q-value, we take the Q-value of the actual action taken by the unit.

H.4 Monte Carlo CoAN

Algorithm 4 Monte Carlo CoAN for classification

```
Input: Number of coagents per layer $n$ , Number of layers $k$, Number of outputs $p$ feature size $d$ and $\mathcal{D}$ be the dataset distribution. Number of epochs $e$, Batch Size : $b$.
Initlialize: Final layer called as $f$, parameterized by $\theta$.
Initlialize : $k+1$ Linear value functions $(q_0, \ldots q_k)$, one for each layer, parameterized by $\phi_i$
Input Param : $\alpha$ - stepsize for gradient descent, for all components
repeat
  $x, y \sim \mathcal{D}$ batch of size $b$
  $o_0 = x$
  // Forward Pass
  for $i = 0$ to $k - 1$ do
    for $j = 0$ to $n - 1$ do
      $a_{i,j} \sim \epsilon - greedy(q_i(o_i, \phi_i))$
      $u_{i,j} = a_{i,j}$
    end for
    // Build next coagent layer state
    $o_{i+1} = (u_{i,0}, \ldots, u_{i,n-1})$
  end for
  $\hat{y} \sim \epsilon$-greedy$(f(o_k)), \theta)$
  $\hat{y} =$ softmax$(\hat{y})$
  // Backward Pass
  $\mathcal{L}_{final} =$ CrossEntropy$(y, \hat{y})$
  $\theta - = \alpha \nabla \mathcal{L}_{final}$
  for $i = k - 1$ to $0$ do
    // Update coagents
    for $j = 0$ to $n - 1$ do
      $g_{i,j} = MSE(q_i(o_i, a_{i,j}), -\mathcal{L}_{final})$
      $\phi_{i,j} - = \alpha \nabla g_{i,j}$
    end for
  end for
until Number of $e$ epochs achieved
```

H.5 Action-value CoANs for regression

The action-value CoAN algorithms for regression are the same as the action-value CoAN algorithms for classification, except when calculating $\mathcal{L}_{final}$, we use the standard MSE rather than Cross-Entropy as the loss function.