Structural Credit Assignment in Neural Networks using Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

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 and 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.

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 [34], 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 [15] 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 [51], 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 [9] and CoAgent Networks (CoANs) [46; 45; 16]. The work in CoANs 1) nicely formalizes the idea

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

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 [50]. 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 [5], rather than improved RL approaches. Other work on credit assignment is loosely inspired by the idea of bootstrapping in RL, including synthetic gradients [13; 17] and fixed-point propagation [27].

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, 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 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 [4, 19], the method of auxiliary variables [6] and fixed point propagation [27]. Kickback approximates the backprop update for ReLu networks, using an approximation to the gradient that allows for local updates [2]. Feedback alignment [22; 29] 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 [36]. Fiete and Seung [9] 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.

Other work has focused on changes in weights across time. Spiking neural networks and the associated spike-timing-dependent plasticity (STDP) learning rule [23] adjusts weights based on the relative timing of activations for nearby nodes. Equilibrium propagation [37] 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 [26] and Contrastive Divergence [12]. 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 [20; 21], with some interesting insights that perturbations can be beneficial to induce exploration [21]. 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 [21, 25]. A more recent algorithm, called cross propagation [48], 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 [8] and learning when to activate parts of the network [5, 3]. 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 [1].

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 [28, 5, 32, 24], though the wider literature on stochastic computation graphs encompasses a broad range of stochastic neural networks [42, 38, 50]. Early work considered an EM-style algorithm [42] and a simple alternative, called the straight-through estimator [5], 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 [39], and a lower-variance update has been proposed [11].

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 [53, 52]; to use a single global reward plus some noise [7]; to use independent Q-learners [41]; or to estimate a global critic [30] potentially with local policy updates [10, 33]. 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 [44] 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 empirical 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 [43]. 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 [32]. 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 $x \in \mathcal{X}$, prediction targets $y \in \mathcal{Y}$, hidden layer activations $h_j = f(z_j)$ for pre-activations 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 $x \in \mathcal{X}$, the initial observation $o_0 = [x, 0]$ where the 0 indicates the step-index in the finite horizon problem. The agent observes input o_0, takes action $a_0 \in \mathcal{A}$ that is a vector of activations h_1 (or pre-activations z_1), and the next observation is deterministically $o_1 = [a_0, 1] = [h_1, 1]$ (or $o_1 = [f(a_0), 1] = [f(z_1), 1]$). Then the agent inputs o_1 and outputs the activations for the next layer a_1 and obtains next observation $o_2 = [a_1, 2]$ (or $o_2 = [f(a_1), 2]$). This transition is Markov, because given o_1 and a_1, the next observation o_2 is independent of 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 x, and so the reward depends on x on this last step. We can obtain Markov states by simply including x in each observation. But, because the policies are not given the ability to use 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 $\mathcal{S} = \mathcal{X} \times \{0\} \cup \mathcal{X} \times \mathbb{R}^n \times \{1, \dots, k-1\}$, actions $\mathcal{A} = \mathbb{R}^n$ and observation function $o(s) = s$ if s has index 0 and otherwise $o(s) = [h, j]$ for $s = [x, h, j]$.

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 $\pi_j(a_j|o_j)$ for $j \in \{0, \dots, k-1\}$ 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, looks like

$$o_0 = x, a_0, r_1 = 0, o_1 = [a_0], \dots, o_k = [a_{k-1}], a_k = \hat{y}, r_{k+1} = -\text{err}(\hat{y}, y), \text{Termination}$$

Because $\pi(a_0, a_1, \dots, a_k|x) = \pi(a_0|x) \prod_{j=1}^{k} \pi_j(a_j|a_{j-1})$, the probability of this trajectory is $p(x)\pi(a_0, a_1, \dots, a_k|x)p(y|x)$. This is because $p(x)$ gives the probability of the start state; $\pi(a_0, a_1, \dots, a_k|x)$ fully defines the probability of the trajectory because the state outcomes are deterministic given the action (activation or pre-activation); and $p(y|x)$ defines the probability of the reward on termination, since the target is stochastic. This provides a straightforward policy gradient objective, for undiscounted return $G \stackrel{\text{def}}{=} -\text{err}(a_k, y)$

$$\max_{\pi} \int p(x)\pi(a_0, a_2, \dots, a_k|x)p(y|x)G dx da_0 \dots da_k dy = \max_{\pi} \mathbb{E}[\pi(a_0, a_1, \dots, a_k|x)G]$$

If we assume each policy π_j has parameters θ_j, then the stochastic gradient of this objective separates out into the following stochastic gradients for each policy separately: $G\nabla_{\theta_j} \ln \pi_j(a_j|o_j)$. We show this formally in Proposition 1 in Appendix B. A similar result has been shown for CoANs used in the RL setting [16, Theorem 1] and for stochastic computation graphs [50, 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 $(G - V(o_j))\nabla_{\theta_j} \ln \pi_j(a_j|o_j)$, where the learned baseline $V(o_j)$ estimates expected value, meaning $G - V(o_j)$ corresponds to the advantage for the actions selected. Several different critics have been proposed for stochastic computation graphs [50]; we discuss how to make similar baselines for this finite horizon problem in Appendix C.

The local policy gradient update, however, is limited in that it can make it difficult to learn nearly deterministic coagents. In fact, we show in the next section that, contrary to the common belief, the culprit for poor performance is not variance in the policy gradient, but rather nondeterminism in other coagents. 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. 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. As before, this process is only partially observable because the final reward depends on the input x.

3 Issues with Nondeterminism in CoANs

In this section, we show that though CoANs with REINFORCE can learn, they are hindered by nondeterminism in other coagents. We investigate a variety of variance reduction approaches, and

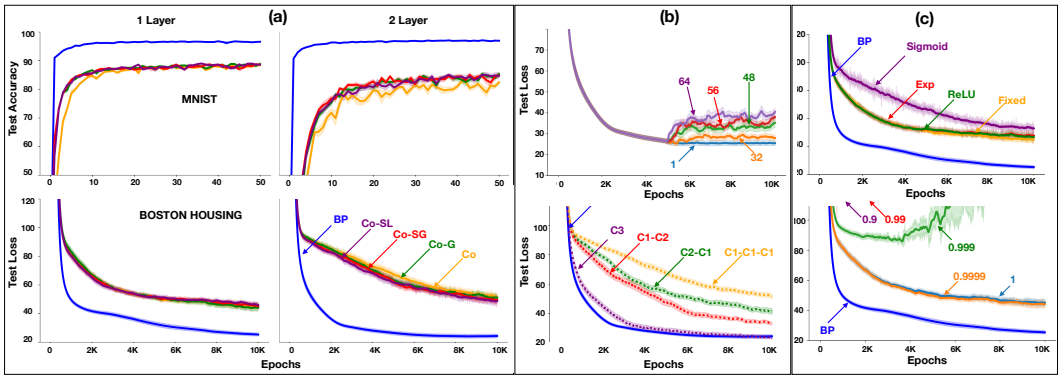

Figure 1: All curves are averaged for 10 runs with standard error bars. **(a)** Training different variants of CoAN's and backprop on MNIST and Boston Housing with one and two layer NNs. **(b)** Experiments highlighting issues with nondeterminism in coagents. The upper graph shows the effect of training with backprop followed by CoAN training for subsets of nodes. The lower graph depicts performance with different coagent partitionings of the network. **(c)** Failures of different strategies to gradually reduce nondeterminism of coagents overtime, on Boston Housing.

find that learning plateaus at a suboptimal point regardless. We investigate more deeply and find that the coagents learn suboptimal action selection due to other coagents outputting stochastic actions.

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 [18] 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 [47], 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.

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 1(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 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 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 causing this gap. We pretrain a two-layer neural network using backprop (up to 5K epochs) and then switch to the CoAN algorithm. We allow a subset of nodes in the first layer to be stochastic and updated. As shown in Figure 1(b), performance starts to degrade rapidly after more than half the nodes are stochastic, though for 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 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). 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 1 (b), we see the deterministic network C3 performs similarly to the backprop network. And as we add stochasticity 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.

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 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 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 1 (c). One possible reason is that for smaller σ of 0.9 and 0.99 the gradient $\nabla_\theta \ln \pi(a|o)$ 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 [5; 40], 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 sparsity of the representation to ensure reasonable levels of activation.

In Figure 2(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 [5], probably due to misalignment [40]. We also include an on-policy Monte Carlo algorithm that learns action-values, which performs poorly; we revisit this when we discuss off-policy learning in the next section.

For Co-G, the ability to better control the entropy does seem to be helping, where in Figure 2(b) the drops in entropy—earlier for layer 1 and then at 200 epochs for layer 2 and 3—cause a sudden rise in accuracy. However, the entropy does not fully decrease and there is some unintuitive behavior.

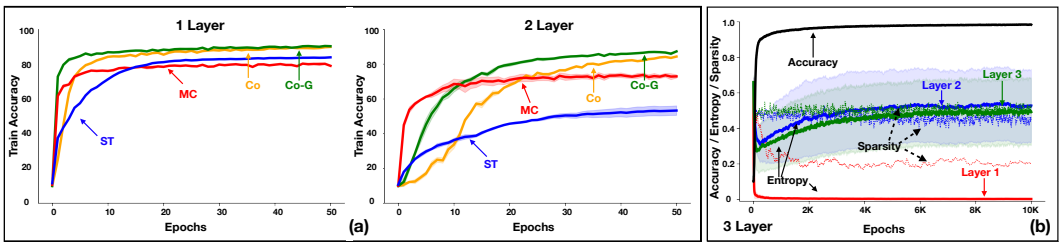

Figure 2: All experiments in this figure are on MNIST dataset, using 64 nodes per layer. **(a)** Learning curves for discrete agents (ST, MC, Co, Co-G) with single and double-layers, averaged for 10 runs, for 50 epochs. **(b)** Learning progress of three layer network for 10K epochs, with entropy *(solid line)* and sparsity *(dotted line)* of each layer, along with accuracy (scaled to $[0, 1]$).

The first layer becomes deterministic faster, which is surprising as it relies on stochastic actions of downstream agents. The entropy for layer 2 and layer 3 also decreases, but eventually the entropy starts to creep back up while maintaining or slightly improving the accuracy. 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 D. 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. We provide additional results in Appendix G.1 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 $Q_j(o_j, a_j)$, for the coagent taking action a_j given input o_j, namely the previous hidden layer h_{j-1}. The update then uses $(Q_j(h_{j-1}, \tilde{h}_j) - V_j(h_{j-1}))\nabla_{\theta_j}) \ln \pi_j(\tilde{h}_j|h_{j-1})$, 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

$$\theta_j^{(q)} \leftarrow \theta_j^{(q)} + \alpha(0 + Q_{j+1}(o_{j+1}, a_{j+1}) - Q_j(o_j, a_j))\nabla Q_j(o_j, a_j)$$

where $Q_j(o_j, a_j)$ can simply be a linear function of o_j, a_j—namely a linear function of $[h_{j-1}, h_j]$—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

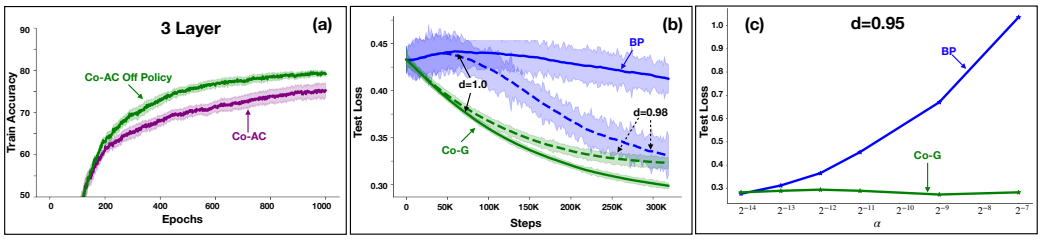

Figure 3: **(a)** Learning curves comparing on policy and off policy actor critic. Averaged for 10 runs, for 3 layer neural network with 64 nodes per layer. **(b)** CL experiments with problem difficulties at $d = 0.98$ *dashed* lines and *solid* lines for $d = 1.0$ comparing backprop and CoAN's. **(c)**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

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 [44, 10, 33]. 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

$$\theta_j^{(q)} \leftarrow \theta_j^{(q)} + \alpha(0 + \max_{a'} Q_{j+1}(o_{j+1}, a') - Q_j(o_j, a_j))\nabla Q_j(o_j, a_j)$$

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. Note the coagents need not reason about greedy actions for upstream coagents, because action-values are *conditioned* on inputs produced by those coagents.

4.2 Results with Off-Policy CoANs

We perform a set of experiments where we estimate $Q_j(o_j, a_j)$ 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 3 (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 [16], 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 E.

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 [31], 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 900th 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.

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 3 (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 F.

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.

6 Conclusions

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 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 [16], 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 [50], 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 facilitates 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.

References

[1] 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.

[2] 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.

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

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

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

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

[7] 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.

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

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

[10] 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.

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

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

[13] 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.

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

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

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

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

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

[19] 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.

[20] 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.

[21] 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.

[22] 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.

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

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

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

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

[27] 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.

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

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

[30] 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.

[31] 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.

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

[33] 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.

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

[35] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[36] 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.

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

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

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

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

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

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

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

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

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

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

[47] 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.

[48] 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.

[49] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 1992.

[50] 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.

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

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

[53] 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.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Section 3, Section 4.2
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] Provides a general framework to do structural credit assignment for neural networks
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A] This is an empirical work

(b) Did you include complete proofs of all theoretical results? [N/A] We have a proof for a general framework we propose

3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Appendix H
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Yes all figures have details of seeds and we talk about the hyperparameters used before setting up each experiment
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] Same as above
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Appendix H
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes] Appendix H
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Appendix H
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A] Open Source datasets
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]