Sampling Aware Reinforcement Learning

Lunjun Zhang [1] **Bradly C Stadie** [1] **Jimmy Ba** [1]

Abstract

We reinterpret the problem of finding intrinsic rewards in reinforcement learning (RL) as a bilevel optimization problem. Using this interpretation, we can make use of recent advancements in the hyperparameter optimization literature, mainly from Self-Tuning Networks (STN), to learn intrinsic rewards. To facilitate our methods, we introduces a new general conditioning layer: Conditional Layer Normalization (CLN). We evaluate our method on several continuous control benchmarks in the Mujoco physics simulator. On all of these benchmarks, the intrinsic rewards learned on the fly lead to higher final rewards.

1. Introduction

The design of good reward functions is a perennial problem in reinforcement learning (RL). Often, rewards are built from high level primitives such as object positions and velocities. For example, if we want to teach a humanoid robot to walk upright, then its reward function would likely depend on the position of its head and its forward velocity. Hand-designed rewards have the advantage of being easy to interpret by the humans designing them. However, these rewards can not be concerned with human interpretability alone. They must also provide signal to an RL algorithm that is responsible for actually training the agent. Unfortunately, it may be the case that easily interpretable hand-designed rewards are inefficient from an optimization perspective. That is to say, it is possible that alternative reward functions exist that lead to faster convergence and better final performance – even when performance is evaluated on the original hand-designed reward function. The central question of this paper is: How can we best recover these more efficient reward functions?

The concept of recovering a more efficient reward function is not a new idea. A sizable number of papers consider the problem of learning intrinsic rewards. In practice, an intrinsic reward is simply any function learned by the agent that is not the original human-provided reward function. These intrinsic rewards are often tied to some exploration objective. For instance, an agent might be encouraged to visit novel states or scenarios (Houthooft et al., 2016). Alternatively, an exploration objective might encourage the agent to learn something fundamental about the environment such as a dynamics model (Jaderberg, 2017). While learning intrinsic rewards through exploratory objectives is an active area of research, there exists no agreed upon methods for measuring novelty or exploration progress.

A recent algorithm named LIRPG (Zheng et al., 2018) addresses the problem of learning intrinsic rewards in a more direct way. Instead of forcing the agent to also optimize an auxiliary objective, LIRPG directly learns a parameterized intrinsic reward function. The parameterized reward is trained by using the chain rule to backpropogate through the intrinsic reward function with respect to the agent's overall reward. LIRPG is appealing because it makes no assumptions about the underlying problem, can be combined with exploration strategies, and directly optimizes the true objective of learning a more efficient reward function. In this paper, we seek to take the ideas underlying LIRPG and take them a step further.

Our key insight is that learning intrinsic rewards can be treated as a bilevel optimization problem. In this setting, the policy optimizes the intrinsic reward in the inner loop and the parameterized intrinsic reward function optimizes the inner-loop policy performance in the outer loop. Under this interpretation, LIRPG is simply performing Gradient-based bilevel optimization through Reversible Learning (Maclaurin et al., 2015). But, we can take things further still. Rather than gradient based planning, we propose to learn intrinsic rewards by leveraging self-tuning networks (Mackay et al., 2019) (STNs).

In practice, STNs were designed to solve the hyperparameter optimization problem. While one could treat intrinsic rewards as a hyperparameter and use STNs to directly search for a parameterized reward function, this process would be inefficient in practice due to the large search space over reward models and the cost of evaluating the parameterized reward functions. To overcome this problem, we use ideas

[1]University of Toronto. Correspondence to: no name <no@no.com>.

Proceedings of the 36^{th} International Conference on Machine Learning, Long Beach, California, PMLR 97, 2019.

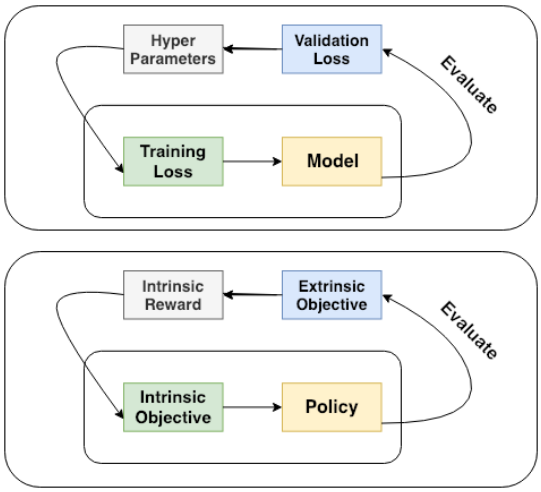

Figure 1: High level comparison of our method (bottom) vs. hyper-parameter optimization (top). In this paper, we will apply recent advances from the hyper parameter optimization literature to the problem of learning intrinsic rewards. We see that the problem of learning intrinsic rewards is analogous to hyper parameter optimization, with the performance on the ground truth extrinsic objective taking the place of the validation loss and the intrinsic objective taking the place of the training loss. Thus, like hyper parameter optimization, learning an intrinsic reward can be cast as a bilevel optimization problem.

from STNs to instead learn parameters in a new type of neural network transformation: Conditional Layer Normalization (CLN). By using a clever gating architecture from the intrinsic reward function's parameters into the parameter space of the policy's CLN transformation, we can allow gradient signal to flow from the policy back through the intrinsic reward function. This architecture allows us to train the intrinsic reward function much more efficiently than prior methods.

2. Related Work

Exploration and Curiosity As discussed in the prequel, many exploration and curiosity methods can be characterized as an attempt to learn an intrinsic reward function that encourages the agent to encounter novelty in its environment. Older work, such as R-Max an Bayesian Exploration Bonuses (BEB) offered strong convergence guarantees but did not scale to the high dimensional problems usually considered in deep RL (Kearns & Singh, 2002; Brafman & Tennenholtz, 2002; Kolter & Ng, 2009). Consequently, there has been a flurry of work on discovering the right novelty metric (Carmel & Markovitch, 1999; Tang et al., 2016; Houthooft et al., 2016; Stadie et al., 2015; Osband et al., 2016; Bellemare et al., 2016; Ostrovski et al., 2017). Throughout his career, Schmidhuber and his colleagues have written extensively on the problem of exploration and curiosity. We recommend the reader review (Schmidhuber, 2015b; Ngo et al., 2012; Graziano et al., 2011; Schmidhuber, 1991; 2015a; Storck et al., 1995; Sun et al., 2011; Kompella et al., 2002; Schmidhuber et al., 1997) for an overview. Novelty-seeking exploration methods are largely orthogonal to our work. In principle, they can be freely combined with the algorithms presented in this paper. Although, learning how to balance an exploratory intrinsic reward and the more exploitative intrinsic rewards presented in this paper remains a challenge.

Hierarchical RL provide an alternative route for learning intrinsic rewards. Hierarchical RL typically involves some sort of high level manager that can set goals for a lower level actor. In this setting, it's possible the actor will receive an intrinsic reward from the manager for following goals (Vezhnevets et al., 2017; Bacon & Precup, 2015; Tessler et al., 2016; Rusu et al., 2016; Barto & Mahadevan, 2003; Wiering & Schmidhuber, 1997). Most work in HRL focuses on abstracting policies across multiple time-scales and goal dimensions. Along the way, these works typically make fairly rigid assumptions about the optimal architecture to produce a hierarchical learner.

Reward Design This paper makes extensive use of the LIRPG method from the reward design literature. Below, we show that our method is effectively a generalize of LIRPG to use STNs rather than gradient based planning. See (Zheng et al., 2018) for a more complete overview of the relevant reward design literature. The upshot is that, prior to LIRPG, there did not exist a reasonable algorithm for learning intrinsic rewards that discovered the intrinsic rewards fast enough to positively impact the online performance of the RL algorithm. Reward shaping, a related problem, is addressed by (Ng et al., 1999). We actually applied the reward shaping algorithms in (Ng et al., 1999) to our paper, but found they did not create a measurable impact on final performance. Finally, (Xu et al., 2018) introduces a method for treating reward function parameters, such as the discount rate, as hyperparameters. These parameters are then optimized via meta learning. Meta-learning these parameters is an orthogonal problem to learning an intrinsic reward function. That approach can be freely combined with ours.

Neuron-wise and Feature-wise Transformation

Layer Normalization (Ba et al., 2016) normalizes the summed inputs to the neurons of a layer on a single training case. Each neuron is given its own adaptive bias and gain after the normalization and before the non-linearity:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l \qquad \sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2} \quad (1)$$

where H denotes the number of hidden units of the network, and a_i^l is the i-th unit of the l-th layer.

Feature-wise Linear Modulation (FiLM), introduced by (Perez et al., 2017), is a recent approach towards visual reasoning problems. Input queries are passed into a deep network, which learn the coefficients for an affine transformation of a hidden layer:

$$\begin{bmatrix}\gamma_{i,c}\\ \beta_{i,c}\end{bmatrix} = f_c(x_i) \qquad \hat{a}_{i,c} = \gamma_{i,c}a_{i,c} + \beta_{i,c} \tag{2}$$

Where γ_c and β_c are modulating parameters for the activation α_c of a layer c, and f_c is any function of the input. FiLM layers are shown to build robustness into the model, by effectively selecting relevant input features. Gating architectures like the one investigated in this work are an active and ongoing area of research (Chaplot et al., 2017b; Munkhdalai et al., 2018).

Hyperparameter Optimization We make use of Self Tuning Networks (STNs) and recent techniques from the hypernetworks literature to help learn intrinsic rewards (Mackay et al., 2019; Lorraine & Duvenaud, 2018). There are several alternative approaches to hyperparemeter optimization. Most notably, Bayesian hyperparameter optimization has seen success in recent years (Snoek et al., 2015). For gradient based approaches wherein the algorithm directly backpropogates through the policy update rule, there are methods such as (Franceschi et al., 2017; Maclaurin et al., 2015).

3. Learning Intrinsic Rewards

3.1. Problem Formulation and Notation

In this section, we will use the following notation:

- r^η: intrinsic reward function parameterized by η.
- $\pi_{\theta,\eta}$: policy function parameterized by θ and conditioned upon the intrinsic reward r^η. Thus, it is also indirectly parameterized by η.
- τ: a trajectory following the policy $\pi_{\theta,\eta}$. We may write $\tau \sim \pi_{\theta,\eta}$.
- θ_k, η_i: the policy parameters and the intrinsic reward function parameters in the k-th round and i-th round of optimization, respectively.
- $\mathcal{J}^{EX}$: The extrinsic RL objective, given by $\mathbb{E}_{\tau\sim\pi_{\theta,\eta}}\left[\sum_t r_t\right]$, where r_t is the extrinsic reward given by the environment at timestep t.
- $\mathcal{J}^{IN}$: The intrinsic objective, which can be thought of as the surrogate objective of the policy. It maximizes the cumulative sum of intrinsic rewards, $\mathbb{E}_{\tau\sim\pi_{\theta,\eta}}\left[\sum_t r_t^\eta\right]$.

The policy $\pi_{\theta,\eta}$ maximizes intrinsic objective $\mathcal{J}^{IN}$. Meanwhile, the intrinsic objective maximizes the policy performance under the extrinsic objective $\mathcal{J}^{EX}$. Thus, the problem becomes a bilevel optimization problem:

$$\boldsymbol{\theta}^* = \arg\max_\theta \mathcal{J}^{IN}(\theta, \eta) \tag{3}$$

subject to

$$\boldsymbol{\eta}^* = \arg\max_\eta \mathcal{J}^{EX}(\theta, \eta) \tag{4}$$

We note that this is analogous to the problem of hyperparameter optimization (See Figure 1). If $\mathbf{w}$ and $\boldsymbol{\lambda}$ denote the parameters and hyperparameters of a model, and if $\mathcal{L}_T$ and $\mathcal{L}_V$ denote the training loss and the validation loss, we arrive at a very similar bilevel optimization formulation (Mackay et al., 2019; Franceschi et al., 2018; Maclaurin et al., 2015; Pedregosa, 2016):

$$\boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\lambda}} \mathcal{L}_V(\boldsymbol{\lambda}, \boldsymbol{w}) \text{ subject to } \boldsymbol{w} = \arg\min_{\boldsymbol{w}} \mathcal{L}_T(\boldsymbol{\lambda}, \boldsymbol{w}) \tag{5}$$

where the training loss $\mathcal{L}_T$ corresponds to the intrinsic objective $\mathcal{J}^{IN}$, and the validation loss corresponds to the the extrinsic objective $\mathcal{J}^{EX}$. By realizing that the two problems share a very similar structure, we speculate that recent advances in hyperparameter optimization can be leveraged to better learn intrinsic rewards for RL agents. In the next section, we attempt to bridge the gap between the two problems by re-examining existing approaches, particularly the gradient-based methods, under this framework of bilevel optimization.

3.2. Optimizing the Intrinsic Reward Objective

3.2.1. Learning Intrinsic rewards with Reverse Mode Differentiation

In this paper, we mainly consider gradient-based approaches. Denote by $\boldsymbol{w}^*(\boldsymbol{\lambda}) = \arg\min_{\boldsymbol{w}} \mathcal{L}_T$ the best-response function. Gradient-based approaches previously tackled hyperparameter optimization problems mainly by approximating the Jacobian of the best-response function. For instance, (Maclaurin et al., 2015) considers reverse-mode differentiation of the learning process to acquire meta-gradients for the hyperparameters. In their case, these parameters are the terms in stochastic gradient descent with momentum. Expanding these ideas out and applying them to our setting, we can derive a direct policy gradient formula for intrinsic rewards. First, we differentiation through the update rules

in the intrinsic reward setting. The resulting policy gradient update is given by:

$$\theta_{k+1} = \theta_k + \alpha \frac{\partial}{\partial \theta_k} \mathcal{J}^{IN}(\theta_k, \eta_i) \tag{6}$$

$$= \theta_k + \alpha \mathbb{E}_{\tau_k} \left[\sum_t \frac{\partial}{\partial \theta_k} \log \pi_{\theta_k, \eta_i} \hat{A}^{IN}_{\tau_k}(t) \right] \tag{7}$$

where $\hat{A}$ is an advantage estimator, usually a generalized advantage estimator (GAE) (Schulman et al., 2016). The policy gradient loss can be replaced by other surrogate losses such as the ones in TRPO and PPO (Schulman et al., 2015; 2017). Those methods have been well studied. The chief challenge here is the mechanism to train the intrinsic reward network. We can compute the meta-gradient of the extrinsic RL objective as:

$$\eta_{i+1} = \eta_i + \beta \frac{d}{d\eta_i} \mathcal{J}^{EX} \tag{8}$$

where

$$\frac{d}{d\eta_i} \mathcal{J}^{EX} \tag{9}$$

$$= \frac{\partial}{\partial \eta_i} \mathcal{J}^{EX}(\theta_{k+1}, \eta_i) + \frac{\partial \mathcal{J}^{EX}}{\partial \theta_{k+1}} \frac{d\theta_{k+1}}{d\eta_i} \tag{10}$$

$$= \mathbb{E}_{\tau_{k+1}} \left[\sum_t \frac{\partial \log \pi_{\theta_{k+1}}}{\partial \eta_i} \hat{A}^{EX}_{\tau_{k+1}}(t) \right] + \tag{11}$$

$$\frac{\partial \mathcal{J}^{EX}}{\partial \theta_{k+1}} \left(\mathbb{E}_{\tau_k} \left[\sum_t \frac{\partial^2 \log \pi}{\partial \theta_k \partial \eta_i} \hat{A}^{IN}_{\tau_k}(t) \right] \right) + \tag{12}$$

$$\frac{\partial \mathcal{J}^{EX}}{\partial \theta_{k+1}} \left(\mathbb{E}_{\tau_k} \left[\sum_t \frac{\partial \log \pi}{\partial \theta_k} \frac{\partial \hat{A}^{IN}_{\tau_k}(t)}{\partial \eta_i} \right] \right) + \tag{13}$$

$$\frac{\partial \mathcal{J}^{EX}}{\partial \theta_{k+1}} \left(I + \mathbb{E}_{\tau_k} \left[\sum_t \frac{\partial^2 \log \pi}{\partial \theta_k^2} \hat{A}^{IN}_{\tau_k}(t) \right] \right) \frac{d\theta_k}{d\eta_i} \tag{14}$$

If we only consider backpropogation through a single step, Equation 14 would be zero. Moreover, if the policy is not conditioned upon the intrinsic reward function itself, then Equations 11 and 12 will be zero as well. With these assumptions, we would be left with only Equation 13, which is exactly LIRPG (Zheng et al., 2018).

LIRPG, in principle, is equivalent to applying reverse-mode differentiation through the learning process to approximate the Jacobian of the best-response function. We note that Equations 12, 13, and 14 are all concerned with the term $\frac{\partial \mathcal{J}^{EX}}{\partial \theta_{k+1}} \frac{d\theta_{k+1}}{d\eta_i}$. There is an intuitive understanding for this term. The term is asking the intrinsic reward function to compute the intrinsic reward it *could have* given the policy such that, upon receiving this term, the policy could have achieved better performance on the extrinsic objective.

3.2.2. Our approach: Intrinsic rewards by estimating the best response function

We now consider an alternative approach to finding intrinsic rewards. This alternative approach is the primary contribution of this paper. In short, our idea is to directly learn the approximate the best-response function. We want a mapping from any hyperparameters $\boldsymbol{\theta}$ to the optimal weights of the model $\boldsymbol{w}^*$, which are the weights of a converged model trained on the inner-loop optimization loss (either the training loss or the policy gradient loss under intrinsic rewards). One way to learn this direct mapping is via a HyperNetwork (Ha et al., 2017). We will make use of Self Tuning Networks (STN) (Mackay et al., 2019), which take the idea behind HyperNetworks a step further by proposing a gating-based architecture for the best-response function. In this setup, the best-response function is trained on the inner-loop training objective given a set of hyperparameters. The hyperparameters are then tuned via the "response gradients", which are essentially the gradients of the outer-loop training loss taken with respect to the hyperparamters. In other words, the response gradients flow through the best-response function before they reach the hyperparameters. Denote the best-response function as $\boldsymbol{w}^*_\phi$ parameterized by ϕ:

$$\boldsymbol{\theta}^* \approx \boldsymbol{w}^*_\phi(\boldsymbol{\eta}) \tag{15}$$

With these adjustments, the gradient updates for training the intrinsic reward function's parameters $\boldsymbol{\eta}$ becomes:

$$\frac{d}{d\eta_i} \mathcal{J}^{EX} = \frac{d\mathcal{J}^{EX}}{d\boldsymbol{\theta}} \frac{d\boldsymbol{w}^*_\phi}{d\eta_i} \tag{16}$$

$$= \mathbb{E}_\tau \left[\sum_t \frac{\partial \log \pi_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} \frac{d\boldsymbol{w}^*_\phi}{d\eta_i} \hat{A}^{EX}_\tau(t) \right] \tag{17}$$

As for training the HyperNet policy we arrive at the gradient,

$$\frac{d}{d\phi_k} \mathcal{J}^{IN} = \frac{d\mathcal{J}^{IN}}{d\boldsymbol{\theta}} \frac{d\boldsymbol{w}^*_\phi}{d\phi_k} \tag{18}$$

$$= \mathbb{E}_\tau \left[\sum_t \frac{\partial \log \pi_{\boldsymbol{\theta}}}{\partial \boldsymbol{\theta}} \frac{d\boldsymbol{w}^*_\phi}{d\phi_k} \hat{A}^{IN}_\tau(t) \right] \tag{19}$$

Let us examine the differences between the expressions in Equation 10, developed under the reverse-mode differentiation framework, and 16, developed under HyperNet framework. Equation 17 is equivalent to Equation 11 (from the expanded version of Equation 10). One might naturally ask: why have the other terms (Equation 12, 13, and 14) vanished in the HyperNet framework?

The answer lies in the assumption we are making. We assume that all the weights in the policy network are conditioned upon the hyperparameters (which, in this case, is a function itself). This means that if there is any part of the

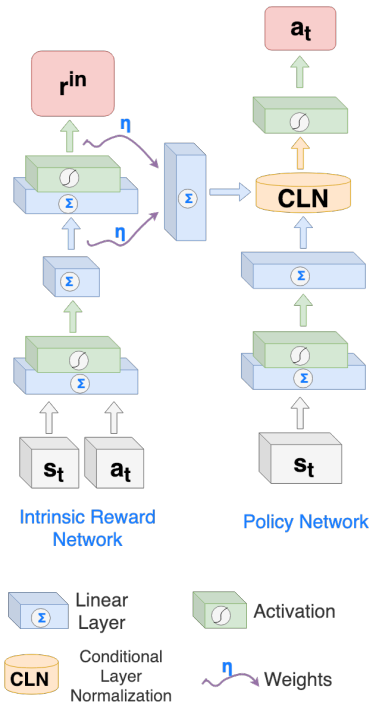

Figure 2: Overview of our model.

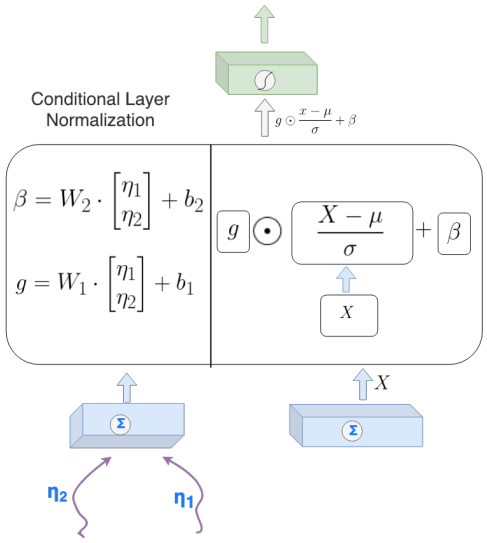

Figure 3: Conditional Layer Normalization. Rather than learning the bias and gate directly as in traditional layer norm, we apply a linear transformation to the weights $[\eta_1, \eta_2]$ from the intrinsic reward model. The image of the intrinsic reward parameters under the linear transformation is then used directly to furnish the bias and gating parameters.

policy network that is independent of the hyperparameters, then the HyperNet assumption will no longer hold true and we need to take Equation 12, 13, and 14 into consideration. In reality, it is often unrealistic to use conditioning on all the weights in a neural network; instead, we usually use gating architectures to *modulate* the behaviour of the network (Perez et al., 2017; Dumoulin et al., 2017; Dhingra et al., 2017; Chaplot et al., 2017a; van den Oord et al., 2016). Yet, the meta-gradients under the HyperNet assumption do have many appealing properties, since they do not require differentiation through the learning process. The upshot is that the more powerful the gating mechanism is, the more accurate the HyperNet meta-gradients becomes when compared against the ground truth gradient. Thus, a more powerful gating mechanism will result in a stronger gradient signal for the HyperNetwork approach. Therefore, the key to the success of HyperNet framework is a powerful conditioning mechanism.

3.3. Learning Intrinsic Rewards With Conditional Layer Norm

The best-response function maps a set of weights to another set of weights. The challenge we face in designing our conditioning architecture is two-fold. First, the information we want to condition our policy upon is a function itself, parameterized by a neural network. Secondly, the conditioning architecture will directly determine how the policy is being modulated and ergo how the meta gradients will flow back during training. With these challenges in mind, we introduce Conditional Layer Normalization (CLN). The key idea in CLN is that the gain and bias parameters from Layer Normalization, rather than being adaptive, will instead be the output of the best-response function (See Figure 3). The input to the best-response function is the weights in the last two layers of the intrinsic reward network. Crucially, the second last layer has an information bottleneck (small number of units) and no activation (see Figure 2). In this work, we use an affine transformation as the best-response function. However, there is an obvious extension to deeper models using our model.

For the l-th layer of the network, Conditional Layer Normalization can be formulated mathematically as:

$$CLN(\mathbf{F}^l \mid \boldsymbol{\eta}) = f\left[\frac{\mathbf{g}^l}{\sigma^l} \odot (\mathbf{a}^l - \mu^l) + \mathbf{b}^l\right] \quad (20)$$

$$\mathbf{g}^l = f_c(\boldsymbol{\eta}) \quad (21)$$

$$\mathbf{b}^l = h_c(\boldsymbol{\eta}) \quad (22)$$

f_c and h_c are both affine transformations in this case (there are obvious extensions to non-linear transformations by adding more layers). We find that for the problems we are concerned with in this paper, applying CLN on the final layer of the policy network is sufficient.

To take advantage of recent developments in the policy gradient literature, we use Proximal Policy Optimization (PPO)

(Schulman et al., 2017) as our policy optimizer. During the policy optimization round, the policy parameters $\boldsymbol{\theta}_{k+1}$ are optimized under the surrogate loss

$$\mathbb{E}_{\tau_k}\left[\text{clip}\left(\frac{\pi_{\theta_{k+1},\eta_i}(\tau_k)}{\pi_{\theta_k,\eta_i}(\tau_k)}\right)_{1-\epsilon}^{1+\epsilon}\hat{A}(\tau_k)\right] \tag{23}$$

Meanwhile, in the validation round, the parameters in the intrinsic reward function $\boldsymbol{\eta}_{i+1}$ are trained upon

$$\mathbb{E}_{\tau_{k+1}}\left[\text{clip}\left(\frac{\pi_{\theta_{k+1},\eta_{i+1}}(\tau_{k+1})}{\pi_{\theta_k,\eta_i}(\tau_{k+1})}\right)_{1-\epsilon}^{1+\epsilon}\hat{A}(\tau_{k+1})\right] \tag{24}$$

Note that in the denominator of the clipping ratio in the validation round, we still use π_{θ_k,η_i} rather than the technically correct $\pi_{\theta_{k+1},\eta_i}$. We find this to be quite important in stabilizing the training process. While we choose to use PPO in this paper due to its popularity and good baseline performance, the combination of CLN and other policy gradient methods is easily possible. We leave this exploration to future work.

4. Experiments

Experimental Setup We evaluate our method on several continuous control benchmarks from OpenAI Gym (Brockman et al., 2016). All tasks use the MuJoCo physics simulator (Todorov et al., 2012). We do not modify the default gym environments in any way. Our methods are parallelized by spawning between 4-16 copies of each environment, each with a different random seed. Each environment is rolled out under the current policy for 2048 timesteps and then gradient information is computed. We then use MPI to average the gradients across each worker. Finally, the weights on each node are updated using this averaged gradient. For all experiments in this section, learning curves are averaged over 5 trials. Each trial represents a full run of the algorithm starting from a different random seed. The shaded regions in the graphs capture 75 % of the total variance present in the learning curves.

Comparison of our method with LIRPG and PPO Figure 4 shows the performance of LIRPG, PPO, and our method (labeled CLN). For all environments, CLN has the slowest convergence rate. However, its final performance is consistently better than the baseline methods across across most tasks. We suspect that the slower convergence is due to the initial difficulty in learning a stable intrinsic reward mapping. See the conversation below analyzing the intrinsic reward graph. For LIRPG, we used the implementation provided by the authors in (Zheng et al., 2018). Overall, we found LIRPG to be highly unstable, even after extensive tuning. It could be LIRPG works best on sparse reward environments, rather than the dense reward environments considered in this paper. For PPO, we used the reference implementation in OpenAI Baselines.

Ablation Analysis on the Three Gradient Terms from Section 3 In the prequel, we derived a full policy gradient formula for the bilevel optimization problem of learning a paramaterized intrinsic reward function. We were curious how much impact each of the three terms, Equations 17, 12, and 13 had on the learning process. In particular, Equation 17 is significantly cheaper to compute than Equations 13 and 12. Due to its second order term, 12 often bottlenecks the optimization process. The results on the Hopper environment are presented in Figure 5. Surprisingly, we see that Equation 17 is the most dominant term. Optimizing against Equation 17 alone produces better performance than optimizing against all three terms combined. This is especially surprising in light of Equation 13 being largely similar to the gradient update from LIRPG. Our intuition suggested that combining these methods should produce the best performance of all. Our best hypothesis is that most of the signal is indeed coming from Equation 17. Another possibility is that balancing all three terms creates difficulties in the optimization process. It is possible that including the right constants in front of all three terms could significantly improve performance. Because of these results, we choose to optimize against only Equation 17 in all experiments labeled CLN.

Analyzing the intrinsic reward We plot the intrinsic reward over time for the Hopper environment in Figure 6. The intrinsic reward oscillates for several million timesteps, seemingly learning nothing. Once it stabilizes, the intrinsic reward starts to increase. It continues to increase seemingly indefinitely, even after the training algorithm has converged. In principle, there is nothing wrong with this because the training objective is unbounded. However, it does seem strange that the intrinsic reward signal does not plateau or otherwise start oscillating again.

Why not delayed reward environments? In LIRPG, the authors used a delayed-reward version of the MuJoCo environments. In these environments, rewards are accumulated for N timesteps. After N timesteps, the agent receives the accumulated rewards all at once. We chose not to use these environments for two reasons. First, the artificial sparsity in the environments does not reflect true sparse reward problems. In the delayed reward environments, the agent receives rewards periodically, in contrast to sparse reward settings where the agent typically receives a reward for making major progress. Second, our algorithm is not meant to be a solution for sparse learning problems. We are concerned with learning a method for finding intrinsic rewards that leads to better performance on standard dense reward RL problems. Since the environments from gym are more commonly used than the delayed reward environments, we choose to use those instead.

What about other feature transformations such as

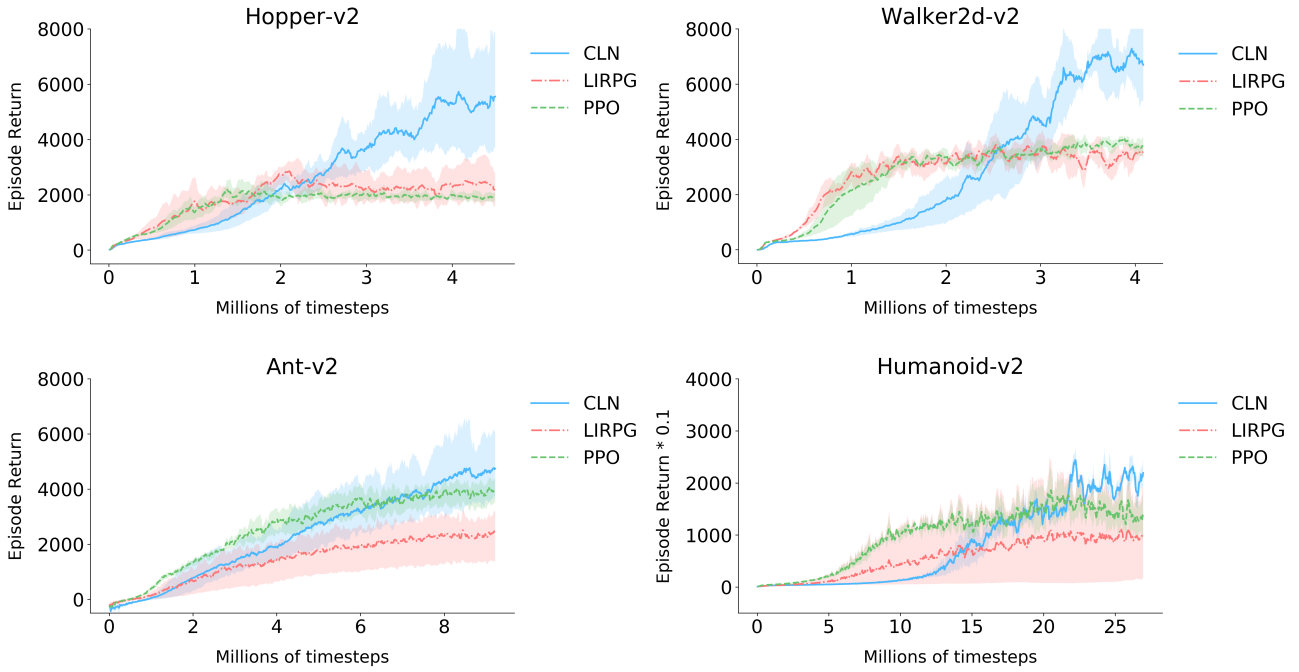

Figure 4: Learning curves for PPO, LIRPG, and our method (labeled CLN). Our method is typically the slowest to converge. However, its final performance is significantly better than the baseline methods. We suspect this slow convergence is due to the difficulty in learning a sensible intrinsic reward function. Humanoid reward is scaled by 0.1.

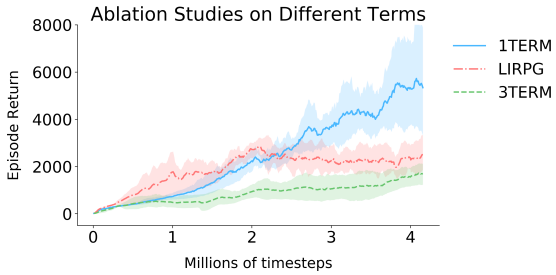

Figure 5: Measuring the impact of Equations 17, 12, and 13 on performance. The blue curve (1-term) corresponds to optimizing with just Equation 17. The red term (LIRPG) cooresponds to optimizing with just equation 13 and is equivilent to LIRPG. The green curve (3term) optimizes against all three equations simultaneously. Surprisingly, we get the best performance when optimizing against Equation 17 alone. Results obtained on the Hopper environment. They are similar across other environments.

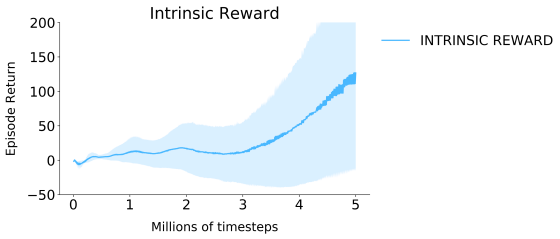

Figure 6: Plot of intrinsic reward obtained by the trained policy on the Hopper environment. For the first 3 million timesteps, the intrinsic reward is mostly flat with occasional oscillations. After 3 millino timesteps, the intrinsic reward obtained by the policy starts increasing and does not stop increasing, even after the policy converges.

FiLM etc? We made a choice to link our intrinsic reward model to the policy model by using a gating architecture the fed into a layer normalization. While we could have also adapted FiLM or another feature transformation technique, rather than layer normalization, preliminary experiments suggested there was no additional benefit in doing so. The simplicity of the layer normalization architecture made it easy to tune and adapt to the problems we consider in this paper.

5. Closing Remarks

The chief difficulties in this work were: 1) Fully implementing the CLN gradient Equation 17. Implementing the second order term (Equation 12) proved especially difficult. We consider any work that eases the implementation of second order methods in stochastic computational graph software to be a worthwhile endeavor. With the growing number of papers in the meta learning literature, the need for easy ways to optimize these graphs should only grow. 2) Tuning the hyper parameters in our method. While it seemed initially that our method was extremely selective to our choice of hyper parameters, we later found a few tricks that greatly improved the stability of our algorithm. In particular, we found that the terms chosen for the policy ratio that multiplies the advantage in the PPO update were crucial for achieving stable performance (See Equation 24). Perhaps there is still more work to be done on improving the stability of policy gradient methods. Despite numerous recent improvements, they still seem fairly brittle.

Potential future work could backport several of the tricks developed in this paper to supervised learning. Using these tricks, it might be possible to regularize supervised learning problems with an intrinsic loss that greatly accelerates the speed of training. Another interesting direction for future work is extending the methods in this paper to work better with exploration strategies. How can we best balance the extreme tendencies for exploitation present in this algorithm with the need for exploration? Similarly, can the methods in this work be extended to work on sparse reward problems? This appears particularly challenging, because our method relies on a constant reward signal to train the intrinsic reward.

References

Ba, J., Kiros, R., and Hinton, G. E. Layer normalization. *CoRR*, abs/1607.06450, 2016.

Bacon, P. and Precup, D. The option-critic architecture. *In NIPS Deep RL Workshop*, 2015.

Barto and Mahadevan. Recent advances in hierarchical reinforcement learning. *Discrete Event Dynamic Systems*, 2003.

Bellemare, M., Srinivasan, S., Ostrovski, G., Schaul, T., Saxton, D., and Munos, R. Unifying count based exploration and intrinsic motivation. *NIPS*, 2016.

Brafman, R. I. and Tennenholtz, M. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research*, 2002.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Mujoco: A physics engine for model-based control. *arXiv:1606.01540*, 2016.

Carmel, D. and Markovitch, S. Exploration strategies for model-based learning in multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 1999.

Chaplot, D., Sathyendra, K., Pasumarthi, R., Rajagopal, D., and Salakhutdinov, R. Gated-attention architectures for task-oriented language grounding. *ACL Workshop on Language Grounding for Robotics*, 2017a.

Chaplot, D. S., Sathyendra, K. M., Pasumarthi, R. K., Rajagopal, D., and Salakhutdinov, R. Gated-attention architectures for task-oriented language grounding. *arXiv:1706.07230*, 2017b.

Dhingra, B., Liu, H., Yang, Z., Cohen, W., and Salakhutdinov, R. Gated-attention readers for text comprehension. *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2017.

Dumoulin, V., Shlens, J., and Kudlur, M. A learned representation for artistic style. *ICLR*, 2017.

Franceschi, L., Donini, M., Frasconi, P., and Pontil, M. Forward and reverse gradient based hyperparameter optimization. *ICML*, 2017.

Franceschi, L., Frasconi, P., Salzo, S., Grazzi, R., and Pontil, M. Bilevel programming for hyperparameter optimization and meta-learning. *ICML*, 2018.

Graziano, V., Glasmachers, T., Schaul, T., Pape, L., Cuccu, G., Leitner, J., and Schmidhuber, J. Artificial Curiosity for Autonomous Space Exploration. *Acta Futura*, 1(1): 41–51, 2011. doi: 10.2420/AF04.2011.41. URL http://dx.doi.org/10.2420/AF04.2011.41.

Ha, D., Dai, A., and V. Le, Q. Hypernetworks. *ICLR*, 2017.

Houthooft, R., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. Vime: Variational information maximizing exploration. *NIPS*, 2016.

Jaderberg, M. e. a. Reinforcement learning with unsupervised auxiliary tasks. *5th Int. Conf. Learn. Representations*, 2017.

Kearns, M. J. and Singh, S. P. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 2002.

Kolter, J. Z. and Ng, A. Y. Near-bayesian exploration in polynomial time. *ICML*, 2009.

Kompella, Stollenga, Luciw, and Schmidhuber. Exploring the predictable. *Advances in Evolutionary Computing*, 2002.

Lorraine, J. and Duvenaud, D. Stochastic hyperparameter optimization through hypernetworks. *CoRR, abs/1802.09419*, 2018.

Mackay, M., Vicol, P., Lorraine, J., Duvenaud, D., and Grosse, R. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. *ICLR*, 2019.

Maclaurin, D., Duvenaud, D., and Adams, R. P. Gradient-based hyperparameter optimization through reversible learning. *In Proceedings of the 32nd International Conference on Machine Learning*, 2015.

Munkhdalai, T., Yuan, X., Mehri, S., and Trischler, A. Rapid adaptation with conditionally shifted neurons. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 3664–3673, Stockholmsmssan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL http://proceedings.mlr.press/v80/munkhdalai18a.html.

Ng, A. Y., Harada, D., and Russell, S. J. Policy invariance under reward transformations: Theory and application to reward shaping. *ICML*, 1999.

Ngo, H., Luciw, M., Forster, A., and Schmidhuber, J. Learning skills from play: Artificial curiosity on a Katana robot arm. *Proceedings of the International Joint Conference on Neural Networks*, pp. 10–15, 2012. ISSN 2161-4393. doi: 10.1109/IJCNN.2012.6252824.

Osband, I., Blundell, C., Pritzel, A., and Van Roy, B. Deep exploration via bootstrapped dqn. *NIPS*, 2016.

Ostrovski, G., Bellemare, M. G., Oord, A. v. d., and Munos, R. Count-based exploration with neural density models. *arXiv:1703.01310*, 2017.

Pedregosa, F. Hyperparameter optimization with approximate gradient. *ICML*, 2016.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. Film: Visual reasoning with a general conditioning layer. *arXiv:1709.07871*, 2017.

Rusu, A. A., Rabinowitz, N. C., Desjardins, G., Soyer, H., Kirkpatrick, J., Kavukcuoglu, K., Pascanu, R., and Hadsell, R. Progressive neural networks. *CoRR, vol. abs/1606.04671*, 2016.

Schmidhuber, J. A Possibility for Implementing Curiosity and Boredom in Model-Building Neural Controllers. *Meyer, J.A. and Wilson, S.W. (eds) : From Animals to animats*, pp. 222–227, 1991.

Schmidhuber, J. On Learning to Think: Algorithmic Information Theory for Novel Combinations of Reinforcement Learning Controllers and Recurrent Neural World Models. *arXiv*, pp. 1–36, 2015a. ISSN 10450823. doi: 1511.09249v1. URL http://arxiv.org/abs/1511.09249.

Schmidhuber, J. Continual curiosity-driven skill acquisition from high-dimensional video inputs for humanoid robots. *Artificial Intelligence*, 2015b.

Schmidhuber, J., Zhao, J., and Schraudolph, N. Reinforcement learning with self-modifying policies. *Learning to learn, Kluwer*, 1997.

Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. Trust region policy optimization. *ICML*, 2015.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. High-dimensional continuous control using generalized advantage estimation. *ICLR*, 2016.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv:1707.06347*, 2017.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Ali, M., Adams, R. P., and et al. Scalable bayesian optimization using deep neural networks. *ICML*, 2015.

Stadie, B. C., Levine, S., and Abbeel, P. Incentivizing exploration in reinforcement learning with deep predictive models. *arXiv:1507.00814*, 2015.

Storck, J., Hochreiter, S., and Schmidhuber, J. Reinforcement driven information acquisition in non-deterministic environments. *Proceedings of the International ...*, 2: 159–164, 1995.

Sun, Y., Gomez, F., and Schmidhuber, J. Planning to be surprised: Optimal Bayesian exploration in dynamic environments. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6830 LNAI:41–51, 2011. ISSN 03029743. doi: 10.1007/978-3-642-22887-2_5.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, X., Duan, Y., Schulman, J., De Turck, F., and Abbeel, P. exploration: A study of count-based exploration for deep reinforcement learning. *arXiv:1611.04717*, 2016.

Tessler, C. Givony, S., Zahavy, T., Mankowitz, D., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. *arXiv:1604.07255*, 2016.

Todorov, E., Erez, T., and Tassa, Y. Mujoco: A physics engine for model-based control. *IROS*, 2012.

van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., and Kavukcuoglu, K. Conditional image generation with pixelcnn decoders. *Advances in Neural Information Processing Systems*, 2016.

Vezhnevets, A. S., Osindero, S., Schaul, T., Heess, N., Jaderberg, M., Silver, D., and Kavukcuoglu, K. Feudal networks for hierarchical reinforcement learning. *arXiv preprint arXiv:1703.01161*, 2017.

Wiering and Schmidhuber. Hq-learning. *Adaptive Behavior*, 1997.

Xu, Z., van Hasselt, H., and Silver, D. Meta-gradient reinforcement learning. *arXiv:1805.09801*, 2018.

Zheng, Z., Oh, J., and Singh, S. On learning intrinsic rewards for policy gradient methods. *In NIPS*, 2018.