BACKPROPAGATION THROUGH THE VOID: OPTIMIZING CONTROL VARIATES FOR BLACK-BOX GRADIENT ESTIMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Gradient-based optimization is the foundation of deep learning and reinforcement learning. Even when the mechanism being optimized is unknown or not differentiable, optimization using high-variance or biased gradient estimates is still often the best strategy. We introduce a general framework for learning low-variance, unbiased gradient estimators for black-box functions of random variables. These estimators can be jointly trained with model parameters or policies, and are applicable in both discrete and continuous settings. We give unbiased, adaptive analogs of state-of-the-art reinforcement learning methods such as advantage actor-critic. We also demonstrate this framework for training discrete latent-variable models.

1 INTRODUCTION

Gradient-based optimization has been key to most recent advances in machine learning and reinforcement learning. The back-propagation algorithm (Rumelhart & Hinton, 1986), also known as reverse-mode automatic differentiation (Speelpenning, 1980; Rall, 1981) computes exact gradients of deterministic, differentiable objective functions. The reparameterization trick (Williams, 1992; Kingma & Welling, 2014; Rezende et al., 2014) allows backpropagation to give unbiased, low-variance estimates of gradients of expectations of continuous random variables. This has allowed effective stochastic optimization of large probabilistic latent-variable models.

Unfortunately, there are many objective functions relevant to the machine learning community for which backpropagation cannot be applied. In reinforcement learning, for example, the function being optimized is unknown to the agent and is treated as a black box (Schulman et al., 2015). Similarly, when fitting probabilistic models with discrete latent variables, discrete sampling operations create discontinuities giving the objective function zero gradient with respect to its parameters. Much recent work has been devoted to constructing gradient estimators for these situations. In reinforcement learning, advantage actor-critic methods (Sutton et al., 2000) give unbiased gradient estimates with reduced variance obtained by jointly optimizing the policy parameters with an estimate of the value function. In discrete latent-variable models, low-variance but biased gradient estimates can be given by continuous relaxations of discrete variables (Maddison et al., 2016; Jang et al., 2016).

A recent advance by Tucker et al. (2017) used a continuous relaxation to construct a control variate for functions of discrete random variables. Low-variance estimates of the expectation of the control variate can be computed using the reparameterization trick to produce an unbiased estimator with lower variance than previous methods. Furthermore, Tucker et al. (2017) showed how to tune the free parameters of these relaxations to minimize their variance during training.

In this work we generalize the method of Tucker et al. (2017) to learn a free-form control variate parameterized by a neural network, giving a lower-variance, unbiased gradient estimator which can be applied to a wider variety of problems with greater flexibility. Most notably, our method is applicable even when no continuous relaxation is available, as in reinforcement learning or black box function optimization. Furthermore, we derive improved variants of popular reinforcement learning methods with unbiased, action-dependent gradient estimates and lower variance.

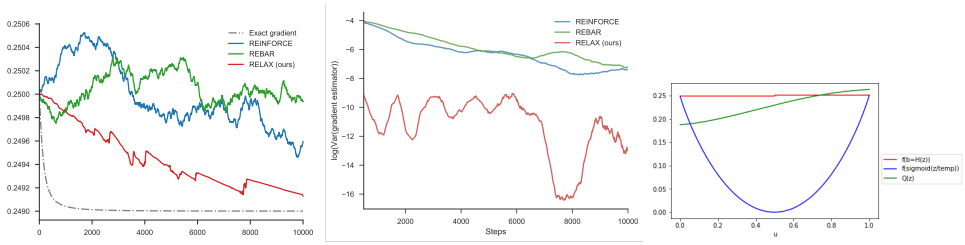

Figure 1: *Left:* Training curves using different gradient estimators on a toy problem: $\mathcal{L}(\theta) = \mathbb{E}_p(b|\theta)[(b - 0.499)^2]$ *Centre:* Variance of each estimator's gradient. *Right:* The relaxation being used by REINFORCE (red), REBAR (blue), and the optimal relaxation (green).

2 Background: Gradient estimators

How can we choose the parameters of a distribution to maximize an expectation? This problem comes up in reinforcement learning, where we must choose the parameters θ of a policy distribution $\pi(a|s, \theta)$ to maximize the expected reward $\mathbb{E}_{\tau\sim\pi}[R]$ over state-action trajectories τ. It also comes up in fitting latent-variable models, when we wish to maximize the marginal probability $p(x|\theta) = \sum p(x|z)p(z|\theta) = \mathbb{E}_{p(z|\theta)}[p(x|z)]$. In this paper, we'll consider the general problem of optimizing

$$\mathcal{L}(\theta) = \mathbb{E}_{p(b|\theta)}[f(b)]\,. \tag{1}$$

When the parameters θ are high-dimensional, gradient-based optimization is a appealing because it provides information about how to adjust each parameter individually. Stochastic optimization is essential for scalablility, but is only guaranteed to converge to a fixed point of the original objective when the stochastic gradients $\hat{g}$ are unbiased, i.e. $\mathbb{E}[\hat{g}] = \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}[f(b)]$ (Robbins & Monro, 1951).

How can we build unbiased, stochastic estimators of $\frac{\partial}{\partial\theta}\mathcal{L}(\theta)$? There are several standard methods:

The score-function gradient estimator One of the most generally-applicable gradient estimators is known as the score-function estimator, or REINFORCE (Williams, 1992):

$$\hat{g}_{\text{reinforce}} = f(b)\,\frac{\partial}{\partial\theta}\log p(b|\theta), \qquad b \sim p(b|\theta) \tag{2}$$

This estimator is unbiased, but in general has high variance. Intuitively, this estimator is limited by the fact that it doesn't use any information about how f depends on b, only on the final outcome $f(b)$.

The reparameterization trick When f is continuous and differentiable, and the latent variables b can be written as a deterministic, differentiable function of a random draw from a fixed distribution, the reparameterization trick (Williams, 1992; Kingma & Welling, 2014; Rezende et al., 2014) creates a low-variance, unbiased gradient estimator by making the dependence of b on θ explicit:

$$\hat{g}_{\text{reparam}} = \frac{\partial}{\partial\theta}f(b(\theta,\epsilon)) = \frac{\partial f}{\partial b}\frac{\partial b(\theta,\epsilon)}{\partial\theta}, \qquad \epsilon \sim p(\epsilon) \tag{3}$$

This gradient estimator is often used when training high-dimensional, continuous latent-variable models, such as variational autoencoders or GANs. One intuition for why this gradient estimator is preferable to REINFORCE is that it depends on $\partial f/\partial b$, which exposes the dependence of f on b.

Control variates Control variates are a general method for reducing the variance of a Monte Carlo estimator. Given an estimator $g(b)$, a control variate is a function $\hat{g}(b)$ with a known mean $\mathbb{E}_{p(b)}[\hat{g}]$. Subtracting the control variate from our estimator and adding its mean gives us a new estimator:

$$\hat{g}_{\text{new}}(b) = g(b) - \hat{g}(b) + \mathbb{E}_{p(b)}[\hat{g}(b)] \tag{4}$$

This new estimator has the same expectation as the old one:

$$\mathbb{E}_{p(b)}[g_{\text{new}}(b)] = \mathbb{E}_{p(b)}\left[g(b) - \hat{g}(b) + \mathbb{E}_{p(b)}[\hat{g}(b)]\right] = \mathbb{E}_{p(b)}[g(b)] \tag{5}$$

Importantly, the new estimator has lower variance than $g(b)$ if $\hat{g}(b)$ is positively correlated with $f(b)$.

Algorithm 1 LAX: Optimizing parameters and a gradient control variate simultaneously.

Require: $f(\cdot)$, $\log p(b|\theta)$, reparameterized sampler $b = T(\theta, \epsilon)$, neural network $r_\phi(\cdot)$
while not converged **do**
$\epsilon_i \sim p(\epsilon)$ ▷ Sample noise
$b_i \leftarrow T(\epsilon_i, \theta)$ ▷ Compute input
$g_\theta \leftarrow [f(b_i) - r_\phi(b_i)] \nabla_\theta \log p + \nabla_\theta r(b_i)$ ▷ Estimate gradient
$g_\phi \leftarrow 2 g_\theta \frac{\partial g_\theta}{\partial \phi}$ ▷ Estimate gradient of variance of gradient
$\theta \leftarrow \theta + \alpha_1 g_\theta$ ▷ Update parameters
$\phi \leftarrow \phi + \alpha_2 g_\phi$ ▷ Update control variate
end while
return θ

3 GRADIENT ESTIMATION WITH LAX AND RELAX

The reparameterization trick has become the favored method for gradient estimation recently but it is only applicable when f is a known, differentiable function of continuous random variables. Similarly, for functions f of discrete variables, methods involving continuous relaxations of discrete distributions such as concrete Maddison et al. (2016), gumbel-softmax Jang et al. (2016), and REBAR Tucker et al. (2017) have become popular. These methods also require that f is known and differentiable with respect to its inputs. Moreover, they require that f be computable at and behave predictably at inputs outside of the domain of the function. While these assumptions often hold, there are many problems of interest which make these approaches inapplicable.

In this section, we introduce a gradient estimator for the expectation of a function $\frac{\partial}{\partial \theta} \mathbb{E}_{p(b|\theta)}[f(b)]$. We make no assumptions about the structure of f itself. This allows the estimator to be applied to a wide-range of applications with little or no modification. This estimator combines the score function estimator, the reparameterization trick, and control variates. We obtain an unbiased estimator whose variance can potentially be as low as reparameterization-trick estimator, even when f is not differentiable or not computable.

We begin with the case where $p(b|\theta)$ is a distribution over continuous random variables which is reparameterizable. We then handle the case where $p(b|\theta)$ is a distribution over discrete random variables.

3.1 THE LAX ESTIMATOR FOR CONTINUOUS RANDOM VARIABLES

We begin with the score-function gradient estimator:

$$\hat{g}_{\text{reinforce}} = f(b) \frac{\partial}{\partial \theta} \log p(b|\theta), \qquad b \sim p(b|\theta) \tag{6}$$

Optimally, we would use the reparameterization trick but we do not assume that f admits this. Instead, we build a surrogate *relaxation* of f using a neural network r_ϕ, and use the reparameterization trick on r_ϕ instead. Using the fact that the score-function estimator and reparameterization estimator have the same expectation:

$$\mathbb{E}_{p(\epsilon)} \left[r_\phi(b(\epsilon, \theta)) \frac{\partial}{\partial \theta} \log p(b|\theta) \right] = \mathbb{E}_{p(\epsilon)} \left[\frac{\partial r_\phi}{\partial b} \frac{\partial b(\theta, \epsilon)}{\partial \theta} \right] \tag{7}$$

We can simply add the score-function estimator and subtract the reparameterization estimator, to obtain:

$$\hat{g}_{\text{LAX}} = (f(b) - r_\phi(b)) \frac{\partial}{\partial \theta} \log p(b|\theta) + \frac{\partial}{\partial \theta} r_\phi(b) \qquad b = b(\theta, \epsilon), \epsilon \sim p(\epsilon). \tag{8}$$

This gives an unbiased estimator for any choice of r_ϕ.

3.2 THE RELAX ESTIMATOR FOR DISCRETE RANDOM VARIABLES

When $p(b|\theta)$ is a distribution over discrete random variables, more care must be taken. We assume that there exists a continuous, reparameterizable distribution $p(z|\theta)$ and a determanistic mapping

$H(z)$ such that $H(z) = b \sim p(b|\theta)$ when $z \sim p(z|\theta)$. Examples of such distributions and functions can be found in appendix A. Most discrete distributions of interest fit into this formulation.

Introducing the parametric function r_ϕ, we derive our estimator similarly to Tucker et al. (2017) with

$$\begin{aligned}
&\frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}[\,f(b)] \\
&= \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b)\right] - \mathbb{E}_{p(z|\theta)}\left[r_\phi(z)\right] + \frac{\partial}{\partial\theta}\mathbb{E}_{p(z|\theta)}\left[r_\phi(z)\right] \\
&= \frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}\left[f(b) - \mathbb{E}_{p(z|b,\theta)}\left[r_\phi(z)\right]\right] + \frac{\partial}{\partial\theta}\mathbb{E}_{p(z|\theta)}\left[r_\phi(z)\right] \\
&= \mathbb{E}_{p(b|\theta)}\left[\left(f(b) - \mathbb{E}_{p(z|b,\theta)}\left[r_\phi(z)\right]\right)\frac{\partial}{\partial\theta}\log p(b|\theta) - \frac{\partial}{\partial\theta}\mathbb{E}_{p(z|b,\theta)}\left[r_\phi(z)\right]\right] + \frac{\partial}{\partial\theta}\mathbb{E}_{p(z|\theta)}\left[r_\phi(z)\right]
\end{aligned} \tag{9}$$

which allows us to define our estimator as

$$\hat{g}_{\text{RELAX}} = \left(f(b) - r_\phi(\tilde{z})\right)\frac{\partial}{\partial\theta}\log p(b|\theta) + \frac{\partial}{\partial\theta}r_\phi(z) - \frac{\partial}{\partial\theta}r_\phi(\tilde{z}) \qquad b = H(z), z \sim p(z|\theta), \tilde{z} \sim p(z|b,\theta) \tag{10}$$

which is also unbiased for any r_ϕ. We note that the distribution $p(z|b,\theta)$ must also be reparameterizable. This is the case for Bernoulli and Categorical random variables. We demonstrate how to perform this conditional reparameterization in appendix A.

3.3 Optimizing the gradient control variate with gradients

Both presented estimators are unbiased for any choice of the function r_ϕ, so the only remaining problem is to choose a r_ϕ that gives low variance to $\hat{g}_{\text{LAX}}$ and $\hat{g}_{\text{RELAX}}$. How can we find a ϕ which gives our estimator low variance? We simply optimize r_ϕ using stochastic gradient descent, at the same time as we optimize the parameters of our model or policy.

To optimize r_ϕ, we require the gradient of the variance of our gradient estimator. To estimate these gradients, we could simply differentiate through the empirical variance over each mini-batch. Or, following Tucker et al. (2017), we can construct an unbiased, single-sample estimator using the fact that our gradient estimator is unbiased. For any unbiased gradient estimator $\hat{g}$ with parameters ϕ:

$$\frac{\partial}{\partial\phi}\text{Variance}(\hat{g}) = \frac{\partial}{\partial\phi}\mathbb{E}[\hat{g}^2] - \frac{\partial}{\partial\phi}\mathbb{E}[\hat{g}]^2 = \frac{\partial}{\partial\phi}\mathbb{E}[\hat{g}^2] = \mathbb{E}\left[\frac{\partial}{\partial\phi}\hat{g}^2\right] = \mathbb{E}\left[2\hat{g}\frac{\partial\hat{g}}{\partial\phi}\right]. \tag{11}$$

Thus, our estimator of the gradient of the variance of $\hat{g}$ is given by $2\hat{g}\frac{\partial\hat{g}}{\partial\phi}$.

What is the form of the variance-minimizing r_ϕ? From inspection of the square of the estimator in (8) we can see that this loss encourages $r_\phi(b)$ to approximate $f(b)$, but with a weighting based on $\frac{\partial}{\partial\theta}\log p(b)$. Moreover, as $r_\phi \to f$ then $\hat{g}_{\text{LAX}} \to \frac{\partial}{\partial\theta}r_\phi$. Thus, this objective encourages a balance between the variance of the reparameterization estimator and the variance of the REINFORCE estimator.

This method of directly minimizing the variance of the gradient estimator stands in contrast to other methods such as Q-Prop Gu et al. (2016) and advantage actor-critic Mnih et al. (2016), which train the control variate to minimize the squared error $(f(b) - r_\phi(b))^2$. Our algorithm, which jointly optimizes the parameters θ and the relaxation r_ϕ is given in Algorithm **??**.

3.4 Designing the control variate

We have introduced a generic family of algorithms which produce unbiased and potentially low variance estimates of $\frac{\partial}{\partial\theta}\mathbb{E}_{p(b|\theta)}[f(b)]$. Furthermore, we provide a tractable method to optimize the parameters of the control variate to minimize the variance of the estimator. The generality of this approach allows us to design our control variate r_ϕ in any way we please. This allows us to utilize any known structure that exists in f to build a suitable control variate.

If, for example, f is a known, differentiable, function of discrete random variables, we can utilize the concrete relaxation Maddison et al. (2016) and let $r_\phi(z) = f(\sigma_\lambda(z))$ in which case our estimator is

exactly the REBAR estimator. We are also free to add a learned component to the concrete relaxation and let $r_\phi(z) = f(\sigma_\lambda(z)) + \hat{r}_\rho(z)$ where $\hat{r}_\rho$ is a neural network with parameters ρ. We have taken this approach in our experiments training discrete variational auto-encoders. If no information about f is known, then we are also free to let r_ϕ be a generic function approximator such as a neural network. This is the approach we have taken in our reinforcement learning experiments. The variance-reduction objective introduced above allows us to use any differentiable, parametric function as our control variate.

4 REINFORCEMENT LEARNING

Reinforcement learning is strong motivating problem for methods similar to ours. In reinforcement learning we seek to optimize the parameters of a policy distribution $\pi(a|s;\phi)$ to maximize the discounted sum of future rewards given that policy $\mathbb{E}_{\pi(\phi)}[\sum_{t=1}^{\infty} \gamma^{t-1} r_t]$ We can view the sum of future rewards as a black-box function of state-action trajectories τ sampled from our policy. Thus, as before we have reduced the problem to that of estimating $\frac{\partial \mathbb{E}_\tau[f(\tau)]}{\partial \phi}$ which is the standard policy gradient algorithm Sutton et al. (2000).

We seek to compute

$$\frac{\partial \mathbb{E}_\tau[R]}{\partial \theta} = \mathbb{E}\Big[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} \sum_{t'=t}^{T} r_t\Big]$$

but the estimator on the right hand side can have potentially high variance. Instead, we typically compute

$$\frac{\partial \mathbb{E}_\tau[R]}{\partial \theta} = \mathbb{E}_\tau\Big[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} [(\sum_{t'=t}^{T} r_{t'}) - b(s_t)]\Big]$$

Where $b(s_t)$ is an estimate of the state-value function, $b(s) \approx V^\pi(s) = \mathbb{E}_\tau[R|s_1 = s]$. This is unbiased for any choice of b since b does not depend on a_t.

4.0.1 REINFORCEMENT LEARNING WITH LAX AND RELAX

We now describe how we apply LAX and RELAX estimators in the RL setting. In the case of RL, b is τ, which denotes a series of actions and states $[s_1, a_1, s_2, a_2, ..., s_T, a_T]$, and the function f refers to the underlying MDP, which maps τ to the sum of discounted reward, $R = \sum_{t=1}^{T} \gamma^t r_t$ with discount factor γ.

We show the continuous case, and leave the discrete case to Appendix. We assume $\pi(a_t|s_t)$ is reparametrizable meaning that we can write $a_t = g_\theta(\epsilon, s_t)$, where ϵ does not depend on θ. We again introduce a new differentiable function $r_\phi(a, s)$, and it's a function of the action and the state pair. We observe that $\forall t$, we have (see details in Appendix **??**)

$$\mathbb{E}_{p(\tau)}\Big[\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} r_\phi(a_t, s_t)\Big] = \mathbb{E}_{p(a_{1:t-1}, s_{1:t})}\Big[\frac{\partial}{\partial \theta} \mathbb{E}_{\pi(a_t|s_t,\theta)}\Big[r_\phi(a_t, s_t)\Big]\Big]$$

Then by adding and subtracting the same term, we have

$$\begin{aligned}
\frac{\partial}{\partial \theta}\mathbb{E}_{p(\tau)}[f(\tau)] &= \mathbb{E}_{p(\tau)}\left[f(\tau) \cdot \frac{\partial}{\partial \theta} \log p(\tau;\theta)\right] - \sum_t \mathbb{E}_{p(\tau)}\Big[\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} r_\phi(a_t, s_t)\Big] + \\
&\quad \sum_t \mathbb{E}_{p(a_{1:t-1}, s_{1:t})}\Big[\frac{\partial}{\partial \theta} \mathbb{E}_{\pi(a_t|s_t,\theta)}\Big[r_\phi(a_t, s_t)\Big]\Big] \\
&= \mathbb{E}_{p(\tau)}\left[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta}\left(\sum_{t'=t}^{T} r_{t'} - r_\phi(a_t, s_t)\right)\right] + \sum_t \mathbb{E}_{p(a_{1:t-1}, s_{1:t})}\Big[\mathbb{E}_{p(\epsilon_t)}\Big[\frac{\partial}{\partial \theta} r_\phi(g_\theta(\epsilon_t, s_t), s_t)\Big]\Big] \\
&= \mathbb{E}_{p(\tau)}\left[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta}\left(\sum_{t'=t}^{T} r_{t'} - r_\phi(a_t, s_t)\right) + \frac{\partial}{\partial \theta} r_\phi(g_\theta(\epsilon_t, s_t), s_t)\right]
\end{aligned}$$

Hence our estimate is defined as:

$$\hat{g}^{\text{RL}}_{\text{LAX}} = \sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} \left(\sum_{t'=t}^{T} r_{t'} - r_\phi(a_t, s_t) \right) + \frac{\partial}{\partial \theta} r_\phi(g_\theta(\epsilon_t, s_t), s_t), \tag{12}$$
$$a_t = g_\theta(\epsilon_t, s_t) \qquad \epsilon_t \sim p(\epsilon_t).$$

5 Related work

The recently-developed REBAR method (Tucker et al., 2017) is the work most related to ours. REBAR estimates the gradient of expectations of functions of Bernoulli random variables. This method uses the REINFORCE estimator with a control variate. The control variate is derived from the original loss function evaluated at continuously-relaxed inputs (Maddison et al., 2016; Jang et al., 2016). The expectation of this control variate is estimated with low variance via the reparameterization trick. The REBAR estimator can been seen as a special case of the RELAX estimator when $r_\phi(z) = f(\sigma_\lambda(z))$.

Unfortunately, REBAR and concrete require the function being optimized, whose input is only defined at discrete inputs, to also accept continuous inputs, be differentiable w.r.t. those inputs, and behave predictably with respect to those continuous inputs. While often true, these are strong assumptions to make. Furthermore, REBAR and concrete require that the function being optimized is known. This makes REBAR and the concrete relaxation inapplicable for optimizing black-box functions, as in reinforcement learning settings where the reward is an unknown function of the environment.

In contrast, LAX and RELAX can be used in these settings. LAX and RELAX only require that we can query the function being optimized, and can sample from and differentiate $p(b|\theta)$.

Can RELAX be used to optimize deterministic black-box functions? The answer is yes, with the caveat that one must introduce stochasticity to the inputs. Thus, RELAX is most suitable for problems where one is already optimizing a distribution over inputs, such as in inference or reinforcement learning.

There has been a great deal of other recent work in the area of gradient estimation. Miller et al. (2017) reduce the variance of reparameterization gradients in an orthogonal way to ours by approximating the gradient-generating procedure with a simple model and using that model as a control variate. NVIL (Mnih & Gregor, 2014), VIMCO (Mnih & Rezende, 2016) provide reduced variance gradient estimation in the special case of discrete latent variable models and discrete latent variable models with monte-carlo objectives. Salimans et al. (2017) estimate gradients using a form of finite differences, evaluating hundreds of different parameter values in pararallel to construct a gradient estimator. In contrast, our method is a simple-sample estimator.

Staines & Barber (2012) address the general problem of developing gradient estimators for deterministic black-box functions or discrete optimization. They introduce a sampling distribution, and optimize an objective similar to ours. Wierstra et al. (2014) also introduce a sampling distribution to build a gradient estimator, and consider optimizing the sampling distribution.

In the reinforcement learning setting, the work most similar to ours is Q-prop Haarnoja et al. (2017). Like our method, Q-prop reduces the variance of the policy gradient with an learned, action-dependent control variate whose expectation is approximated via a monte-carlo sample from a taylor series expansion of the control variate. Unlike our method, their control variate is trained off-policy. While our method is applicable in both the continuous and discrete action domain, Q-prop is only applicable in environments with continuous actions. We are interested in the potential of training our control variate off-policy, but we leave that for further work.

6 Applications

We demonstrate the effectiveness of our estimator on a number of challenging optimization problems. Following Tucker et al. (2017) we begin with a simple toy example to illuminate the potential of our method and then continue to the more relevant problems of optimizing binary VAE's and reinforcement learning.

6.1 TOY EXPERIMENT

We seek to minimize $\mathbb{E}_{p(b|\theta)}[(b-t)^2]$ as a function of the parameter θ where $p(b|\theta) = \texttt{Bernoulli}(b|\theta)$. Tucker et al. (2017) set the target $t = .45$. We focus on the more challenging case where $t = .499$. With this setting of the target, REBAR and competing methods suffer from high variance and are unable to discover the optimal solution $\theta = 0$.

The fixed Concrete relaxation of REBAR is unable to produce a gradient whose signal outweighs the sample noise and is therefore unable to solve this problem noticeably faster than REINFORCE. Figure **??** plots the learned relaxations for a fixed value of θ.

It can be seen that RELAX learns a relaxation whose derivative points in the direction of decreased loss for all values of reparameterization noise u, whereas REBAR's fixed relaxation only does so for values of $u > t$.

6.2 DISCRETE VARIATIONAL AUTOENCODER

As in Tucker et al. (2017), we benchmark the RELAX estimator on the task of training a variational autoencoder (Kingma & Welling, 2014; Rezende et al., 2014) where the latent variables are Bernoulli. As in Tucker et al. (2017), we compare training the variational lower-bound across the MNIST and Omniglot (Lake et al., 2015) datasets. As in Tucker et al. (2017) we test models with 1 and 2 layers of 200 Bernoulli random variables with linear mappings between them.

In the one layer models we optimize the ELBO

$$\mathcal{L}(\theta) = \mathbb{E}_{q(b|x)}[\log p(x|b) + \log p(b) - \log q(b|x)]$$

where $q(b_1|x) = \sigma(x \cdot W_q + \beta_q)$ and $p(x|b_1) = \sigma(b_1 \cdot W_p + \beta_p)$ with weight matrices W and biases β. The parameters of the prior $p(b)$ are also learned. Details of the two layer model can be found in the Appendix C.1.1.

To take advantage of the available structure in the loss function, we choose the form of our control variate to be

$$r_\phi(z) = \hat{r}_\rho(z) + f(\sigma_\lambda(z))$$

where $\hat{r}_\rho$ is a neural network with parameters ρ and $f(\sigma_\lambda(z))$ is the discrete loss function (the evidence lower-bound) evaluated at continuously relaxed inputs as in REBAR.

We compare against the state-of-the-art baseline of REBAR Tucker et al. (2017). In all experiments, we the learned control variate improved the training and validation performance as well as increased the rate of convergence.

Dataset	Model	Concrete	NVIL	MuProp	REBAR	RELAX
MNIST	1 layer	-111.3	-112.5	-111.7	-111.6	**-111.20**
	2 Layer	-99.62	-99.6	-99.07	-98.22	**-98.00**
Omniglot	1 layer	-117.23	-117.44	-117.09	-116.63	**-116.57**
	2 Layer	-109.95	-109.98	-109.55	-108.71	**-108.54**

Table 1: Best obtained training objective.

MNIST	REBAR ours	RELAX
1 layer	-114.32	**-113.62**
2 Layer	-101.20	**-100.85**
Omniglot		
1 layer	-122.44	**-122.11**
2 Layer	-115.83	**-115.42**

Table 2: Best obtained validation objective.

To obtain training curves we created our own implementation of REBAR, which gave identical or slightly improved performance to the implementation of Tucker et al. (2017).

While we obtained a modest improvement in training and validation scores (tables 1 and 2), the most notable improvement provided by RELAX is in its rate of convergence. Training curves for all models can be seen in figures 2 and 3. In table 3 we compare the number of training epochs that are required to match the best validation score of REBAR. In all experiments, RELAX provides an increase in rate of convergence.

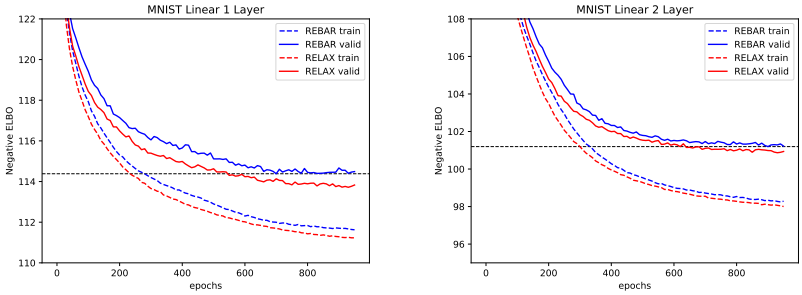

Figure 2: Training curves on MNIST. The horizontal dashed line indicates the lowest validation error obtained by REBAR.

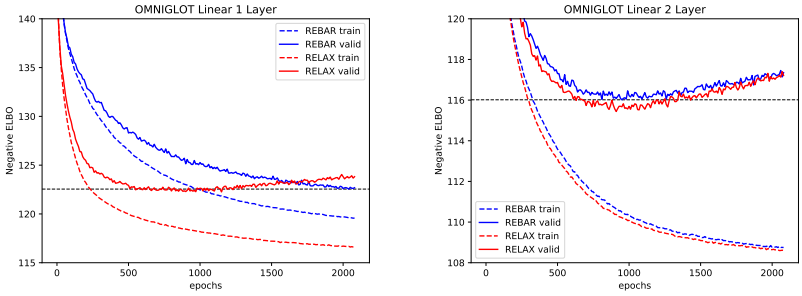

Figure 3: Training curves on OMNIGLOT. The horizontal dashed line indicates the lowest validation score obtained by REBAR.

MNIST	REBAR ours	RELAX
1 layer	857	**531**
2 Layer	900	**620**
Omniglot		
1 layer	2086	**566**
2 Layer	1027	**673**

Table 3: Epochs needed to achieve REBAR's best validation score.

6.3 Reinforcement Learning

We test our approach on a few simple reinforcement learning environments with discrete and continuous actions. We use the RELAX and LAX estimators for discerete and continuous actions, respectively. We compare with the advantage actor-critic algorithm Sutton et al. (2000) as a baseline. In all experiments we utilized the same learning rate for the policy network for RELAX and A2C so differences in performance depended solely on the control variate used.

To compare these approaches in the most illustrative setting possible we do not use any reward bootstrapping in either model. After each episode terminates, we generate the discounted reward for each time-step, treat the episode as a single batch of data, and perform one step of gradient decent. We are aware that better results could be obtained with bootstrapping and larger batch sizes but we wanted to work in the highest possible variance setting to demonstrate the variance reduction capabilities of our approach.

6.3.1 DISCRETE EXPERIMENTS

We test our approach in the discrete action domains of the CartPole and LunarLander as provided by the OpenAI gym Brockman et al. (2016). For all models, the policy and control variate were both neural networks with 2 layers of 10 units using the ReLU nonlinearity. We first tuned the learning rate using the baseline A2C then ran RELAX with the same learning rate. In both domains we observe improved performance and sample efficiency using our method. Again, we are aware that improved results could be obtained by further tuning the learning rate for RELAX but we wanted to illustrate the improvement provided solely by the improved control variate.

Experimental Details We estimate the policy gradient using a single Monte Carlo sample produced from a one episode roll-out of the current policy. We note that improved performance could be achieved by using more samples, but we intended to test our model in the highest-possible variance setting. In all experiments both the policy and control variate were two layer neural networks with RELU non-linearities. Each intermediate layer's output had dimension 10. We searched over two hyper-parameters; the global learning rate and the scaling on the loss from the control variate or value function. For each we tested values in $[.01, .003, .001]$. between our model and the baseline and the learning rates were also constant across tests making it that all improvements derive from having a lower variance estimate of the policy gradient. Presented results are obtained by averaging over 5 runs with different random seeds. We run the CartPole and LunarLander for $250,000$ and $5,000,000$ time-steps respectively.

CartPole	A2C	RELAX
Episodes until solve	1152 ± 90	$\mathbf{472 \pm 114}$
LunarLander		
Episodes until solve	10703 ± 4087	$\mathbf{5156.8 \pm 794.9}$

Table 4: Mean Episodes to Solve. CartPole is considered solved if the agent receives an average reward over 195 in the last 100 Episodes. LunarLander is considered solved if the agent receives an average reward over 200 in the last 100 frames.

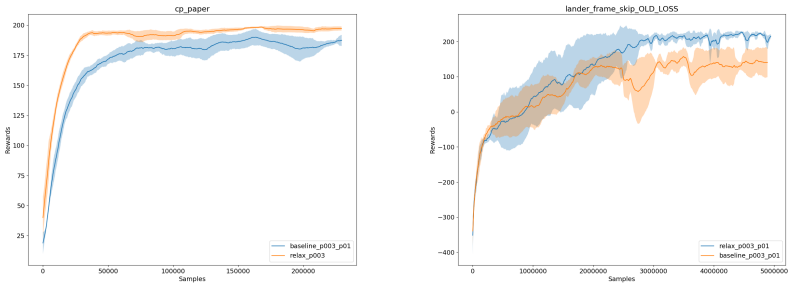

Figure 4: Discrete RL Experiments.
Left: CartPole. *Right:* LunarLander.

Mnih et al. (2015)

[Do we use ADAM (Kingma & Ba, 2015) for optimization?]

7 Conclusions and Future Work

In this work, synthesized and generalized many of the standard approaches for constructing gradient estimators. We proposed a simple and generic gradient estimator that can be applied to expectations of known or black-box functions of discrete or continuous random variables. We also derive a simple extension to apply our method to reinforcement learning in both discrete and continuous action domains. This approach is relatively simple to implement and adds minimal computational overhead.

The foundation of our approach is the score function gradient estimator with a control variate whose expectation can be estimated with low variance using the reparameterization trick. This control variate is neural network which is trained directly to minimize the variance of the estimated gradients. The central result of this paper is that learning the function in the control variate leads to even better convergence properties and lower variance gradient estimates.

Other possible applications:

GANs (Goodfellow et al., 2014) that generate text or other discrete objects.

Learning to parse (Kusner et al., 2017)

VAEs with continuous latent variables but non-differentiable likelihood functions.

References

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv:1702.08165*, 2017.

Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

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

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *International Conference on Learning Representations*, 2014.

Matt J Kusner, Brooks Paige, and José Miguel Hernández-Lobato. Grammar variational autoencoder. *arXiv preprint arXiv:1703.01925*, 2017.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

Andrew C Miller, Nicholas J Foti, Alexander D'Amour, and Ryan P Adams. Reducing reparameterization gradient variance. *arXiv preprint arXiv:1705.07880*, 2017.

Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pp. 1791–1799, 2014.

Andriy Mnih and Danilo Rezende. Variational inference for monte carlo objectives. In *International Conference on Machine Learning*, pp. 2188–2196, 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL http://dx.doi.org/10.1038/nature14236.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

Louis B Rall. Automatic differentiation: Techniques and applications. 1981.

Danilo J Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31st International Conference on Machine Learning*, pp. 1278–1286, 2014.

Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pp. 400–407, 1951.

David E Rumelhart and Geoffrey E Hinton. Learning representations by back-propagating errors. *Nature*, 323:9, 1986.

Tim Salimans, Jonathan Ho, Xi Chen, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3528–3536, 2015.

Bert Speelpenning. *Compiling Fast Partial Derivatives of Functions Given by Algorithms*. PhD thesis, University of Illinois at Urbana-Champaign, 1980.

Joe Staines and David Barber. Variational optimization. *arXiv preprint arXiv:1212.4507*, 2012.

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

George Tucker, Andriy Mnih, Chris J Maddison, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *arXiv preprint arXiv:1703.07370*, 2017.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

APPENDICES

A CONDITIONAL RE-SAMPLING FOR DISCRETE RANDOM VARIABLES

When applying the RELAX estimator to a function of discrete random variables $b \sim p(b|\theta)$, we require that there exists a distribution $p(z|\theta)$ and a deterministic mapping $H(z)$ such that if $z \sim p(z|\theta)$ then $H(z) = b \sim p(b|\theta)$. Treating both b and z as random, this procedure defines a probabalistic model $p(b, z|\theta) = p(b|z)p(z|\theta)$. The RELAX estimator requires reparameterized samples from $p(z|\theta)$ and $p(z|b, \theta)$. We describe how to sample from these distributions in the common cases of $p(b|\theta) = \text{Bernoulli}(\theta)$ and $p(b|\theta) = \text{Categorical}(\theta)$.

Bernoulli When $p(b|\theta)$ is Bernoulli distribution we let $H(z) = \mathbb{I}(z > 0)$ and we sample from $p(z|\theta)$ with

$$z = \log \frac{\theta}{1-\theta} + \log \frac{u}{1-u}, \qquad u \sim \text{uniform}[0, 1].$$

We can sample from $p(z|b, \theta)$ with

$$\tilde{z} = \begin{cases} v \cdot \theta & b = 0 \\ v(1-\theta) + \theta & b = 1 \end{cases}$$

where $v \sim \text{uniform}[0, 1]$.

Categorical When $p(b|\theta)$ is a Categorical distribution where $\theta_i = p(b = i|\theta)$, we let $H(z) = \text{argmax}(z)$ and we sample from $p(z|\theta)$ with

$$z = \log \theta - \log(-\log u), \qquad u \sim \text{uniform}[0, 1]^k$$

where k is the number of possible outcomes.

Intuitively, to sample from $p(z|b, \theta)$ we should first sample $v \sim \text{uniform}[0, 1]^k$, then compute $g_b = \log \theta_b - \log(-\log(v_b))$. Then we must determine how to scale each $v_{i \neq b}$ such that $g_{i \neq b} < g_b$. We can define v' such that

$$v'_i = \begin{cases} v_i & i = b \\ v_i \cdot (v_b)^{\frac{\theta_i}{\theta_b}} & i \neq b \end{cases}$$

and then $\tilde{z} = \log \theta - \log(-\log v')$ which is our sample from $p(z|b, \theta)$.

B REINFORCEMENT LEARNING PROOFS

Proof 1 We begin with

$$\mathbb{E}_\tau \Big[\frac{\partial m(a_t, s_t)}{\partial \theta} \Big] = \mathbb{E}_{a_{1:t-1}, s_{1:t}} \Big[E_{a_{t:T}, s_{t+1:T}} \Big[\frac{\partial m(a_t, s_t)}{\partial \theta} \Big] \Big] \tag{13}$$

$$= \mathbb{E}_{a_{1:t-1}, s_{1:t}} \Big[E_{a_t} \Big[\frac{\partial m(a_t, s_t)}{\partial \theta} \Big] \Big] \tag{14}$$

$$= \mathbb{E}_{a_{1:t-1}, s_{1:t}} \Big[\frac{\partial E_{a_t}[m(a_t, s_t)]}{\partial \theta} \Big] \tag{15}$$

$$= \mathbb{E}_{a_{1:t-1}, s_{1:t}} \Big[E_{a_t} \Big[\frac{\partial \log \pi(a_t|s_t, \theta)}{\partial \theta} m(a_t, s_t) \Big] \Big] \tag{16}$$

$$= E_\tau \Big[\frac{\partial \log \pi(a_t|s_t, \theta)}{\partial \theta} m(a_t, s_t) \Big] \tag{17}$$

which completes our proof.

In the discrete action setting our policy parameterizes a soft-max distribution which we use to sample actions. In the discrete case we define $z_t = f(\pi, u) = \sigma(\log \pi - \log(-\log(u)))$ where $u \sim \text{Unif}[0, 1]$, $a_t = \text{argmax}(z_t)$, σ is the soft-max function.

We also define $\tilde{z_t} \sim p(z_t|a_t)$ so if the z_t are Gumbel-softmax samples, then $\tilde{z_t}$ are Gumbel-softmax samples constrained so that the largest value is that of the index of a_t. See appendix A for how to efficiently generate samples $\tilde{z_t}$.

In the discrete case, we use $m(\tilde{z}_t, s_t) \cdot \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta}$ as our control variate giving us

$$\frac{\partial \mathbb{E}_\tau[R]}{\partial \theta} = \mathbb{E}_\tau\Big[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta}[(\sum_{t'=t}^{T} r_{t'}) - m(\tilde{z}_t, s_t)]\Big]$$

which is biased, so we must add a term to remove this bias which will be $\frac{\partial m(z_t,s_t)}{\partial \theta} - \frac{\partial m(\tilde{z}_t,s_t)}{\partial \theta}$ making the full estimator

$$\frac{\partial \mathbb{E}_\tau[R]}{\partial \theta} = \mathbb{E}_\tau\Big[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta}[(\sum_{t'=t}^{T} r_{t'}) - m(\tilde{z}_t, s_t)] + \frac{\partial m(z_t,s_t)}{\partial \theta} - \frac{\partial m(\tilde{z}_t,s_t)}{\partial \theta}\Big].$$

For this estimator to be unbiased, we must have

$$\mathbb{E}_\tau\Big[\sum_{t=1}^{T} \frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} m(\tilde{z}_t, s_t) - \Big(\frac{\partial m(z_t,s_t)}{\partial \theta} - \frac{\partial m(\tilde{z}_t,s_t)}{\partial \theta}\Big)\Big] = 0$$

and we claim that $\forall t$

$$\mathbb{E}_\tau\Big[\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} m(\tilde{z}_t, s_t) - \Big(\frac{\partial m(z_t,s_t)}{\partial \theta} - \frac{\partial m(\tilde{z}_t,s_t)}{\partial \theta}\Big)\Big] = 0$$

We introduce the notation $a(z) = \text{argmax}(z)$

$$0 = \frac{\partial}{\partial \theta}\mathbb{E}_\tau\Big[m(z_t,s_t) - m(z_t,s_t)\Big] = \mathbb{E}_{a_{<t}, s_{\leq t}}\Big[\frac{\partial}{\partial \theta}\mathbb{E}_{z_t|s_t}\Big[m(z_t,s_t)\Big] - \frac{\partial}{\partial \theta}\mathbb{E}_{a_t|s_t}\Big[\mathbb{E}_{z_t|a_t,s_t}\Big[m(z_t,s_t)\Big]\Big]\Big] \tag{18}$$

We drop the outer expectation for brevity and continue

$$\frac{\partial}{\partial \theta}\mathbb{E}_{z_t|s_t}\Big[m(z_t,s_t)\Big] - \frac{\partial}{\partial \theta}\mathbb{E}_{a_t|s_t}\Big[\mathbb{E}_{z_t|a_t,s_t}\Big[m(z_t,s_t)\Big]\Big] \tag{19}$$

$$= \frac{\partial}{\partial \theta}\mathbb{E}_{z_t|s_t}\Big[m(z_t,s_t)\Big] - \mathbb{E}_{a_t|s_t}\Big[\mathbb{E}_{z_t|a_t,s_t}\Big[m(z_t,s_t)\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} - \frac{\partial}{\partial \theta}\mathbb{E}_{z_t|a_t,s_t}\Big[m(z_t,s_t)\Big]\Big] \tag{20}$$

$$= \mathbb{E}_{z_t|s_t}\Big[\frac{\partial}{\partial \theta} m(z_t,s_t)\Big] - \mathbb{E}_{a_t|s_t}\Big[\mathbb{E}_{z_t|a_t,s_t}\Big[m(z_t,s_t)\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} - \mathbb{E}_{z_t|a_t,s_t}\Big[\frac{\partial}{\partial \theta} m(z_t,s_t)\Big]\Big] \tag{21}$$

Thus due to the markov property of the MDP, then we can wrap all of these expectations into the expectation over trajectories giving us

$$\mathbb{E}_{a_{<t}, s_{\leq t}}\Big[\mathbb{E}_{z_t|s_t}\Big[\frac{\partial}{\partial \theta} m(z_t,s_t)\Big] - \mathbb{E}_{a_t|s_t}\Big[\mathbb{E}_{z_t|a_t,s_t}\Big[m(z_t,s_t)\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} - \mathbb{E}_{z_t|a_t,s_t}\Big[\frac{\partial}{\partial \theta} m(z_t,s_t)\Big]\Big]\Big] \tag{22}$$

$$= \mathbb{E}_\tau\Big[\frac{\partial \log \pi(a_t|s_t,\theta)}{\partial \theta} m(\tilde{z}_t, s_t) - \Big(\frac{\partial m(z_t,s_t)}{\partial \theta} - \frac{\partial m(\tilde{z}_t,s_t)}{\partial \theta}\Big)\Big] = 0 \tag{23}$$

C Experimental Details

C.1 Discrete VAE

We run all models for $2,000,000$ iterations with a batch size of 24. For the REBAR models, we tested learning rates in $\{.005, .001, .0005, .0001, .00005\}$.

RELAX adds more hyperparameters. These are the depth of the neural network component of our control variate r_ρ, the weight decay placed on the network, and the scaling on the learning rate for

the control variate. We tested neural network models with l layers of 200 units using the ReLU nonlinearity with $l \in \{2, 4\}$. We trained the control variate with weight decay in $\{.001, .0001\}$. We trained the control variate with learning rate scaling in $\{1, 10\}$.

To limit the hyperparameter search and to demonstrate the impact of reduced-variance gradient estimates, we choose the base learning rate for RELAX to be the best performing learning rate from REBAR. We believe further improvement could be achieved by tuning this parameter.

All presented results are from the models which achieve the highest ELBO on the validation data.

C.1.1 TWO LAYER MODEL

In the two layer models we optimize the ELBO

$$\mathcal{L}(\theta) = \mathbb{E}_{q(b_2|b_1)q(b_1|x)}[\log p(x|b_1) + \log p(b_1|b_2) + \log p(b_2) - \log q(b_1|x) - \log q(b_2|b_1)]$$

where $q(b_1|x) = \sigma(x \cdot W_{q_1} + \beta_{q_1})$, $q(b_2|b_1) = \sigma(b_1 \cdot W_{q_2} + \beta_{q_2})$, $p(x|b_1) = \sigma(b_1 \cdot W_{p_1} + \beta_{p_1})$, and $p(b_1|b_2) = \sigma(b_2 \cdot W_{p_2} + \beta_{p_2})$ with weight matrices W and biases β. As in the one layer model, the prior $p(b_2)$ is also learned.

C.2 DISCRETE RL

C.3 CONTINUOUS RL