Backprop-Q: Generalized Backpropagation for Stochastic Computation Graphs

Xiaoran Xu[1], Songpeng Zu[1], Yuan Zhang[2]*, Hanning Zhou[1], Wei Feng[1]
[1]Hulu Innovation Lab, Beijing, China
{xiaoran.xu, songpeng.zu, eric.zhou, wei.feng}@hulu.com
[2]School of Electronics Engineering and Computer Science, Peking University, Beijing, China
yuan.z@pku.edu.cn

Abstract

In real-world scenarios, it is appealing to learn a model carrying out stochastic operations internally, known as *stochastic computation graphs* (SCGs), rather than learning a deterministic mapping. However, standard backpropagation is not applicable to SCGs. We attempt to address this issue from the angle of cost propagation, with local surrogate costs, called *Q-functions*, constructed and learned for each stochastic node in an SCG. Then, the SCG can be trained based on these surrogate costs using standard backpropagation. We propose the entire framework as a solution to generalize backpropagation for SCGs, which resembles an actor-critic architecture but based on a graph. For broad applicability, we study a variety of SCG structures from one cost to multiple costs. We utilize recent advances in reinforcement learning (RL) and variational Bayes (VB), such as off-policy critic learning and unbiased-and-low-variance gradient estimation, and review them in the context of SCGs. The generalized backpropagation extends transported learning signals beyond gradients between stochastic nodes while preserving the benefit of backpropagating gradients through deterministic nodes. Experimental suggestions and concerns are listed to help design and test any specific model using this framework.

1 Introduction

The *Credit assignment problem* has been seen as the fundamental learning problem. Given a long chain of neuron connections, it studies how to assign "credit" to early-stage neurons for their impact on final outcome through downstream connections. It dates back to a well-known approach, called *backpropagation* [1]. Error signals are propagated from the output layer to hidden layers, guiding weight updates. The "credit" is a signal of loss gradient calculated by the chain rule. Meanwhile, some work attempts to seek an alternative to exact gradient computation, either by finding a biologically plausible implementation [2, 3, 4, 5], or using synthetic gradients [6]. The insight is that building feedback pathways may play a more crucial role than assuring the preciseness of propagated gradients. More specifically, instead of gradients, feedback learning signals can be target values [3], synthetic gradients [6], or even signals carried through random feedback weights [4, 5].

However, the great success of deep neural networks in a variety of real-world scenarios is largely attributed to the standard gradient-based backpropagation algorithm due to its effectiveness, flexibility, and scalability. The major weakness is its strict requirement that neural networks must be deterministic and differentiable, with no stochastic operations permitted internally. This limits the potential of neural networks for modeling large complex stochastic systems. Therefore, rather than figuring out

---

*Work done during the internship in Hulu

an alternative to backpropagation, we aim at extending it to become applicable to arbitrary stochastic computation graphs. Specifically, we propose to conduct the propagation process across stochastic nodes, with propagated learning signals beyond gradients, while preserving the benefit of standard backpropagation when transporting error gradients through the differentiable part.

Recently, many efforts have focused on solving the tasks that require effective training by backpropagation along with sampling operations, called *backpropagation through stochastic neurons*. As one of the early work, [7] studied four families of solutions to estimate gradients for stochastic neurons, including the straight-through estimator, but limited to binary neurons.

In variational inference and learning, training with samples arises from the fact that it optimizes an expectation-form objective, a variational lower bound, with respect to distribution parameters. Based on Monte Carlo sampling, several unbiased and low-variance estimators have been proposed for continuous and discrete random variables, using the techniques such as the reparameterization trick [8, 9, 10], control variates [11, 12, 13], continuous relaxation [14, 15] and most recently, hybrid methods combining the previous techniques [16, 17, 18]. However, these methods studied a direct cost $f(z)$ defined on random variables, without systematically considering the effect of long-delayed costs after a series of stochastic operations, which is the key of the credit assignment problem.

In reinforcement learning, a Markov decision process can be viewed as a chain of stochastic actions and states, and the goal is to maximize the expected total rewards, with delayed rewards considered. The temporal-difference (TD) learning method [19], along with policy gradient methods [20, 21] and various on- and off-policy techniques, such as experience replay [22, 23, 24], separate target network [22, 25, 23, 24], advantage function [21, 26] and controlled policy optimization [27, 28], provide a powerful toolbox to solve temporal credit assignment [29]. However, rare work has thought of reinforcement learning from a nonsequential perspective, for example, a more structured decision graph, made of a mix of policy networks, with various value functions interwoven and learned jointly.

The learning problem for SCGs was first clearly formulated in [30], solved by a modification of standard backpropagation, much like a graph-based policy gradient method without critic learning. Inspired by this work, we study the possibility of backpropagating value-based signals in TD-style updates, as a complement to gradient-based signals, and propose a more generalized framework to implement backpropagation, called *Backprop-Q*, applicable to arbitrary SCGs, absorbing many useful ideas and methods recently introduced in RL and VB.

In this paper, our contributions mainly focus on two aspects: (1) cost propagation and (2) how to construct and learn local surrogate costs. For cost propagation, to transport expectation of costs back through stochastic nodes, we introduce a Backprop-Q network associated with a set of tractable sample-based update rules. For local surrogate costs, we parameterize each by a neural network with compact input arguments, analogous to a critic (or a value function) in reinforcement learning. To the best of our knowledge, this paper is the first to consider learning critic-like functions from a graph-based view. Combined with standard backpropagation, our work depicts a big picture where feedback signals can go across stochastic nodes and go beyond gradients.

The primary purpose of this paper is to provide a learning framework with wide applicability and offer a new path to training arbitrary models that can be represented in SCGs, at least formally. In practice, much future work needs to be done to examine what specific type of SCG problems can be solved effectively and what trick needs to be applied under this framework. Despite lack of experimental demonstration, we list possible suggestions and concerns to conduct future experiments.

2 Preliminary

Stochastic computation graphs (SCGs). We follow Schulman's [30] definition of SCGs, and represent an SCG as $(\mathcal{X}, \mathcal{G}_{\mathcal{X}}, \mathcal{P}, \Theta, \mathcal{F}, \Phi)$. $\mathcal{X}$ is the set of random variables, $\mathcal{G}_{\mathcal{X}}$ the directed acyclic graph on $\mathcal{X}$, $\mathcal{P} = \{p_x(\cdot | Pa_x; \theta_x) \mid X \in \mathcal{X}\}$ the set of conditional distribution functions parameterized by Θ, and $\mathcal{F} = \{f_i(\mathcal{X}_i; \phi_i) \mid \mathcal{X}_i \subseteq \mathcal{X}\}$ the set of cost functions parameterized by Φ. Although an SCG contains parameter nodes (including inputs), deterministic nodes and stochastic nodes, for simplicity we leave out notations for deterministic nodes as they are absorbed into functions p_x or f_i. Note that $\mathcal{G}_{\mathcal{X}}$ and $\mathcal{P}$ make a probabilistic graphical model (PGM) such that the feedforward computation through SCG performs ancestral sampling. However, an SCG expresses different semantics from a PGM by $\mathcal{G}_{\mathcal{X}}$ in two aspects: 1) it contains costs; 2) the detail of deterministic nodes and their

connections to other nodes reveals a finer modeling of computation dependencies omitted by $\mathcal{G}_\mathcal{X}$. Furthermore, due to the flexibility of expressing dependencies in SCGs, parameters can be shared or interacted across Θ, Φ without being limited to local parameters.

Learning problem in SCGs. The learning problem in an SCG is formulated as minimizing an expected total cost $J(\Theta, \Phi) = \mathbb{E}_{\mathcal{X} \sim \mathcal{P}; \Theta}[\sum f_i(\mathcal{X}_i; \phi_i)]$ over distribution parameters in Θ and cost parameters in Φ jointly. J is usually intractable to compute and therefore approximated by Monte Carlo integration. When applying stochastic optimization, the stochasticity arises not only from mini-batch data but also from sampling procedure, resulting in imprecision and difficulty, compared to optimizing a deterministic neural network. However, SCGs apply to a much wider variety of tasks as long as their objective functions can be written in expectation.

SCGs for probabilistic latent models. For probabilistic latent models, the formulation using SCGs has two different ways: sampling via generative models or via inference networks. The former fits a latent-variable model by maximizing likelihood $p(x; \theta) = \mathbb{E}_{p(z;\theta)}[p(x|Z; \theta)]$ for a single observation. The latter is more popular, known as variational Bayes [11, 8, 9]. Here, the inference network acts as an SCG that performs actual sampling, and the generative model only provides probabilistic functions to help define a variational lower bound as the SCG's cost, with the approximate posterior as well. The final expected cost will be $\mathbb{E}_{q(z|x;\phi)}[\log p(Z; \theta) + \log p(x|Z; \theta) - \log q(Z|x; \phi)]$.

SCGs for reinforcement learning. SCGs can be viewed in the sense of reinforcement learning under known deterministic transition. For each stochastic node X, $p_x(x|Pa_x; \theta_x)$ is a policy where x means an action and Pa_x means the state to take action x. Whenever an action is taken, it possibly becomes part of a state for next actions taken at downstream stochastic nodes. Although it simplifies reinforcement learning without considering environment dynamics, it is no longer a sequential decision process, but a complex graph-based decision making, which integrates various policies, with rewards or costs coming from whatever branch leading to a cost function.

SCGs for stochastic RNNs. Traditional RNNs build a deterministic mapping from inputs to predictions, resulting in exposure bias when modeling sentences [31]. It is trained by ground truth words as oppose to words drawn from the model distribution. Using a stochastic RNN, an instance of SCGs, can overcome the issue, because the next word is sampled based on its previous words.

3 Basic Framework of Backprop-Q

In this section, we first demonstrate how to construct local surrogate costs and derive their update rules in one-cost SCGs. Then, we extend our methods to multi-cost SCGs with arbitrary structure.

3.1 One-Cost SCGs

Cost propagation. Why is cost propagation needed? If we optimize $\mathbb{E}_{X \sim p(\cdot; \theta)}[f(X)]$ over θ, we can get an unbiased gradient estimator by applying the REINFORCE [20] directly. However, considering a long chain with an objective $\mathbb{E}_{X_{1:t-1}}[\mathbb{E}_{X_t|X_{1:t-1}}[\mathbb{E}_{X_{t+1:T}|X_t}[f(X_T)]]]$, a given x_t is supposed to be associated with the conditional expected cost $\mathbb{E}_{X_{t+1:T}|x_t}[f(X_T)]$, rather than a delayed $f(x_T)$. The REINFORCE estimator is notorious for high variance due to the sampling-based approximation for $\mathbb{E}_{X_t|x_{1:t-1}}[\cdot]$ given $x_{1:t-1}$, and using $f(x_T)$ after sampling over $X_{t+1:T}$ across a long chain will make it much worse. Unlike [30] without addressing this issue, we aim at learning expected costs conditioned on each random variable and using Rao-Blackwellization [32] to reduce variance due to $\mathrm{Var}(\mathbb{E}_{Y|X}[f(Y)]) \leq \mathrm{Var}(f(Y))$. We find that these expected costs follow a pattern of computing expectation updates on one random variable each time, starting from the cost and flowing backward through all random variables.

Local surrogate costs. In a chain-like SCG, cost propagation based on expectation updates resembles learning a value function in reinforcement learning, which is a function of current state or state-action pair. However, in a general SCG, the expected costs appear more complex.

Theorem 1. *(SCG's gradient estimators) Given an SCG with a cost function f defined on $\mathcal{Z} \subseteq \mathcal{X}$, and each random variable associated with its own distribution parameter such that $X \sim p(\cdot|Pa_X; \theta_X)$, the gradient of the expected total cost J with respect to θ_X can be written as:*

$$\nabla_{\theta_X} J = \mathbb{E}_{An_X, X}\left[\nabla_{\theta_X} \log p(X|Pa_X; \theta_X) \cdot Q_X(Fr_{An_X \cup \{X\}})\right] \tag{1}$$

3

(a) A SCG with only stochastic nodes and the cost shown　　　(b) A Backprop-Q network corresponding to the left SCG
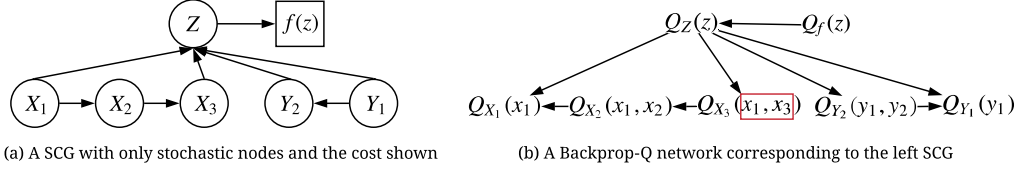
Figure 1: An instance of one-cost SCGs and its Backprop-Q network

where Pa_X is the set of X's parents, An_X the set of X's ancestors and $Fr_V \subseteq V$ the frontier [2] *of a set of random variables V, defined as: a subset of random variables from which the cost is reachable through random variables not in V. We also define a Q-function for each stochastic node, representing the expected cost depending on this random variable and its necessary ancestors such that:*

$$Q_X(Fr_{An_X \cup \{X\}}) := \mathbb{E}_{Z|Fr_{An_X \cup \{X\}}}[f(\mathcal{Z})] \tag{2}$$

The Q-function Q_X has an enlarged scope when a bypass goes around X to the cost. The scope incorporates the ancestor of X from which the bypass starts, carrying extra information needed at X when evaluating Q_X. The scope thus makes a frontier set for X and all its ancestors, indicating the Markov property that given this scope the remaining ancestors will not affect the cost. Therefore, Q_X acts as a *local surrogate cost* to X of the remote cost, much like seeing what the future looks like from the perspective of its own scope and trying to minimize $\mathbb{E}_{An_X,X}[Q_X(Fr_{An_X \cup \{X\}})]$.

Backprop-Q network. To propagate cost, we need to derive the rules of expectation updates. Let X, Y be two stochastic nodes such that $X \in Pa_Y$ and then we have: $Q_X(Sc_X) = \mathbb{E}_{V|Sc_X}[Q_Y(Sc_Y)]$, where scope $Sc_X = Fr_{An_X \cup \{X\}}$, $Sc_Y = Fr_{An_Y \cup \{Y\}}$ and $V = Sc_Y - An_X \cup \{X\}$ represents what variables are still unknown in Sc_Y at node X. Figure 1 shows that a Q-function may have more than one equivalent update rules, e.g., Q_{X_1} and Q_{Y_1}, when a node has multiple paths reaching the cost. The update rules between Q-functions can be represented by the reversed $\mathcal{G}_{\mathcal{X}}$ of an SCG, plus the cost as a root. We call it a *Backprop-Q network*. Each node in a Backprop-Q network is a Q-function [3], e.g., $Q_X(Sc_X)$, indexed by a stochastic node X in $\mathcal{G}_{\mathcal{X}}$, a scope denoted as Sc_X and a cost source [4]. We represent a Backprop-Q network as $(\mathcal{Q}, \mathcal{G}_{\mathcal{Q}}, \mathcal{R})$, where $\mathcal{Q}$ is the set of Q-functions, $\mathcal{G}_{\mathcal{Q}}$ the directed acyclic graph on $\mathcal{Q}$ and $\mathcal{R} = \{\mathcal{R}_x \mid \mathcal{R}_x Q_X(Sc_X) := \mathbb{E}[Q_Y(Sc_Y)], X \in Pa_Y, \forall X \in \mathcal{X}\}$ the set of update-rule operators. If Q_X has multiple equivalent update rules, we pick any one or take the average. In multi-cost SCGs, we will meet multiple Q_X with different scopes and cost sources at the same node X, making $\mathcal{G}_{\mathcal{Q}}$ no more a reversed $\mathcal{G}_{\mathcal{X}}$.

Learning local surrogate cost. If a local surrogate cost is exactly a true expected cost, we can obtain an unbiased gradient estimator by Eq.1. However, computing a sweep of expectation updates is usually intractable. We thus turn to sample updates. For each Q-function, we sample one step forward, use this sample to query the next Q-function and then update it as: $Q_X(Sc_X) \leftarrow Q_X(Sc_X) + \alpha[Q_Y(y, Sc_Y^{-y}) - Q_X(Sc_X)]$, where $y \sim p_Y(\cdot|Pa_Y; \theta_Y)$ is the drawn sample, assuming $X \in Pa_Y$ and other parents known, and α is a step size. We can also run an ancestral sampling pass and generate a full set of samples to then update each Q-function backward. It is a graph version of on-policy TD-style learning. The downside is sampling error and accumulated incorrectness of downstream Q-functions due to lack of exact expectation computation. Is there a convergence guarantee? Would these Q-functions converge to the true expected costs? In a tabular setting, the answer is yes as in reinforcement learning [19]. When Q-functions are estimated by function approximators, denoted as Q_w, especially in a nonlinear form like neural networks, we have the convergence guarantee as well, so long as each Q-function is independently parameterized and trained sufficiently, as opposed to what we know in reinforcement learning. When learning Q_{w_X} from Q_{w_Y}, for example, applying sample updates is actually doing one-step stochastic gradient descent to reduce the expected squared errors by optimizing w_X:

$$\begin{aligned} Err(w_X) &:= \mathbb{E}_{An_Y,Y}[(Q_{w_X}(Sc_X) - Q_{w_Y}(Sc_Y))^2] \\ &\geq \mathbb{E}_{An_X,X}[(Q_{w_X}(Sc_X) - \mathbb{E}_{Sc_Y - An_X \cup \{X\}|Sc_X}[Q_{w_Y}(Sc_Y)])^2] \end{aligned} \tag{3}$$

---

[2] In a multi-cost SCG, a cost f must be specified for a frontier, denoted as Fr_V^f

[3] We consider a cost f a special Q-function, deonted as $Q_f(\cdot) := f(\cdot)$ with the same scope as f.

[4] In the multi-cost setting, we need to label a cost source for Q-functions, e.g., $Q_X^f(Sc_X)$
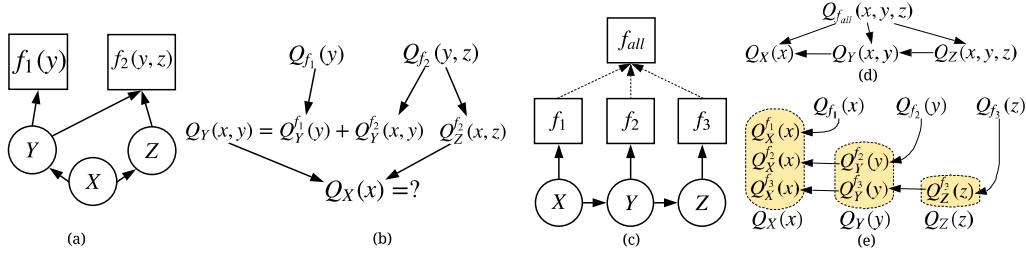
Figure 2: Multi-cost SCGs and their Backprop-Q networks

The one-step update on w_X is: $w_X \leftarrow w_X + \alpha(Q_{w_Y}(Sc_Y) - Q_{w_X}(Sc_X))\nabla Q_{w_X}(Sc_X)$.

Theorem 2. *(Convergence of learned Q-functions) Given a Backprop-Q network with one cost as the root, if the expected squared error between each learned Q_{w_X} and its parent Q_{w_Y} can be bounded by ϵ ($\epsilon > 0$) such that $\mathbb{E}_{An_Y, Y}[(Q_{w_X}(Sc_X) - Q_{w_Y}(Sc_Y))^2] \leq \epsilon$, then we have:*

$$\mathbb{E}_{An_X, X}\left[\left(Q_{w_X}(Sc_X) - Q_X(Sc_X)\right)^2\right] \leq (3 \cdot 2^{l_{Q_X} - 1} - 2)\epsilon \quad for\ l_{Q_X} \geq 1 \tag{4}$$

where $Q_X(Sc_X)$ represents the true expected cost and l_{Q_X} the length of the path from Q_X to the root.

The above shows the deviations from true Q-functions accumulate as l_{Q_X} increases. As a Backprop-Q network has a finite size, the deviations can go infinitely small when each Q_{w_X} is sufficiently trained to fit Q_{w_Y}. Due to independent parameterization, optimizing w_X will not affect Q_{w_Y}'s convergence.

SCGs with a multivariate cost. For a cost defined on multiple random variables, e.g., $f(X_1, X_2, X_3)$, we can assume a virtual node prepended to the cost, which collects all the random variables in f's scope into one big random variable $\mathcal{Z} = (X_1, X_2, X_3)$, following a deterministic conditional distribution $\mathcal{Z} \sim p_z(\cdot | X_1, X_2, X_3)$. The rest procedure is the same as the above.

SCGs with shared parameters. Consider a case with parameter θ shared by all distributions and even the cost. We replace θ with local parameters, e.g., θ_X, each only corresponding to one random variable but constrained by identity mapping $\theta_X = \theta$. To compute $\nabla_\theta J$, we compute the gradients w.r.t. each local parameter and then take the sum of them as $\nabla_\theta J = \sum_X \nabla_{\theta_X} J$.

Remarks. Standard backpropagation transports gradients, the first-order signals, while we propagate the zero-order signals of function outputs through stochastic nodes, with cumulative effect by past updates. When the approximate Q-functions get close to the true ones, we can expect their first-order derivatives also get close to the true gradients in some sense, which means we can utilize the gradients of the approximate Q-functions as well. The theoretic analysis can be found in Appendix.

3.2 Multi-Cost SCGs

Trouble caused by multiple costs. A stochastic node leading to multiple costs, e.g., Y in Figure 2(a), may have Q-functions of different scopes and different cost sources as shown in Figure 2(b). The expected cost $Q_Y(x, y)$ at node Y is the sum of those from two costs respectively. However, it is confusing to update $Q_X(x)$ based on $Q_Y(x, y)$ and $Q_Z^{f_2}(x, z)$, because summing will double f_2 and averaging will halve f_1. We can treat two costs separately to build a Backprop-Q network for each, so that we can track cost sources and take the update target for Q_X as: $Q_Y^{f_1}(y) + (Q_Y^{f_2}(x, y) + Q_Z^{f_2}(x, z))/2$. However, it is expensive to build and store a separate Backprop-Q network for each cost, and maintain probably multiple Q-functions at one stochastic node. An alternative way is to wrap all costs into one and treat it as a one-cost case as shown in Figure 2(c), but the scopes of Q-functions can be lengthy as in Figure 2(d).

Multi-cost Backprop-Q networks. In many cases, per-cost Backprop-Q networks can be merged and reduced. For example, in Figure 2(e), we sum Q-functions at each stochastic node into one, i.e., $Q_X(x) := Q_X^{f_1}(x) + Q_X^{f_2}(x) + Q_X^{f_3}(x)$, thus requiring only one Q-function at node X. The process resembles the one-step TD method in reinforcement learning, except that Q-functions are parameterized independently.
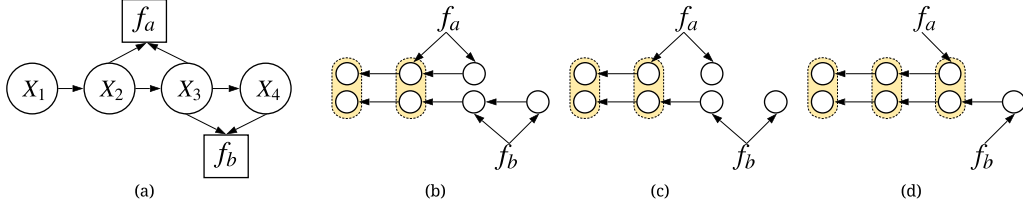
Figure 3: Merging two Backprop-Q networks

Theorem 3. *(Merging Backprop-Q networks) Two Backprop-Q networks can be merged at stochastic node X and its ancestors, if the two are fully matched from X through X's ancestors, that is, the set of the incoming edges to each ancestor in a Backprop-Q network is exactly matched to the other.*

In Figure 3 two costs provide separate Backprop-Q networks as in Figure 3(b). We can merge them at the last two nodes according to the above theorem. The update rules are always averaging all or picking one over incoming edges with the same cost source, and then summing those from different cost sources. Furthermore, we can reduce each Backprop-Q network into a directed rooted spanning tree, ensuring that each node receives exactly one copy of the cost. Many ways exist to construct a tree. Figure 3(c) shows a version with shorter paths but no benefit for merging, while Figure 3(d) constructs a chain version so that we can get a much more simplified Backprop-Q network.

Some complex cases. The above merging guideline can apply to more complex SCGs, and result in a surprisingly reduced Backprop-Q network. In Appendix, we consider a stack of fully-connected stochastic layers, with costs defined on each stochastic node.

4 Enhanced Backprop-Q

4.1 Using Techniques from Reinforcement Learning

λ**-return updates.** λ-return provides a way of moving smoothly between Monte Carlo and one-step TD methods [19]. It offers a return-based update, averaging all the n-step updates, each weighted proportional to λ^{n-1}. If $\lambda = 1$, it gives a Monte Carlo return; if $\lambda = 0$, it reduces to the one-step return. Therefore, λ trades off estimation bias with sample variance. We borrow the idea from [19, 26] to derive a graph-based λ-return method. For each node, we collect upstream errors, multiplied by λ and the discount factor γ, and add it to the current TD error. The combined error then propagates downstream. It follows the same pattern (averaging or summing) as the update rules defined by Backprop-Q networks. Cost propagation turns into propagation of TD errors. The limitation is that the updating must run in a backward pass synchronously. Some cases can be found in Appendix.

Experience replay. This technique is used to avoid divergence when training large neural networks [22, 23, 24]. It keeps the recent N experiences in a replay buffer, and applies TD updates to experience samples drawn uniformly at random. It breaks up a forward pass of ancestral sampling and may lose a full return. However, by reusing off-policy data, it breaks the correlation between consecutive updates and increases sample efficiency. It also allows asynchronous updating, which means cost propagation over a Backprop-Q network can be implemented at each node asynchronously. In a MDP, an experience tuple is (s_t, a_t, s_{t+1}) and then a sample a_{t+1} is drawn by a target policy. In the setting of SCGs, we develop a graph-based experience replay that an experience tuple for node Q_X is represented as $(X, \mathcal{A}, \mathcal{B}^1, \mathcal{A}^1 \ldots, \mathcal{B}^K, \mathcal{A}^K)$, where $\mathcal{A}$ is X's ancestors in Q_X's scope, $\mathcal{B}^k$ represents other potential parents affecting a common child Y_k with X, and $\mathcal{A}^k$ is Y_k's ancestors in Q_{Y_k}'s scope. Here, we assume that X has K children, which means that Q_X probably has K upstream Q-functions to combine. The updates are based on the optimization given below:

$$\min_{w_X} \mathbb{E}_{(X,\mathcal{A},\mathcal{B}^1,\mathcal{A}^1,\ldots,\mathcal{B}^K,\mathcal{A}^K)\sim\text{Uniform}(RB)} \left[\sum_{k=1}^{K} \mathbb{E}_{Y_k \sim p(\cdot|X,\mathcal{B}^k;\theta_{Y_k})} \left(\left[Q_{w_{Y_k}}(Y_k, \mathcal{A}^k) \right] - Q_{w_X}(X, \mathcal{A}) \right)^2 \right]$$

where RB means a replay buffer. A case can be found in Appendix as an illustration.

Other techniques. (1) To improve stability and avoid divergence, we borrow the ideas from [22, 23] to develop a *slow-tracking target* network. (2) We study graph-based *advantage functions* and use

them to replace Q-functions in the gradient estimator to reduce variance. (3) We apply *controlled policy optimization* to distribution parameters in SCGs, using the ideas from [27, 28]. See Appendix.

4.2 Using Techniques from Variational Bayesian Methods

In the framework of generalized backpropagation, after learning local surrogate costs for stochastic nodes, we need to train distribution parameters of the SCG, that is, we should continue the back-propagation process to transport gradients of local costs through underlying differentiable subgraphs. However, there is still one obstacle we must overcome. The objective function, $\mathbb{E}_{z \sim p(\cdot; \theta)}[f(Z)]$ where $f(z) := Q_{w_Z}(z)$, is an expectation in terms of a distribution we need to optimize.

Stochastic optimization has been widely used to solve the optimization problem. The key is to obtain a low-variance and unbiased gradient estimator applicable to both continuous and discrete random variables. The simplest and most general method is the REINFORCE estimator [20], but it is usually impractical due to high variance. Recently, to solve the backpropagation through stochastic operations in variational inference and learning, several advanced methods have been proposed, including the reparameterization trick [8, 9, 10], control variates [11, 12, 13, 16], continuous relaxation [14, 15] and some hybrid methods like Rebar [17] and RELAX [18] to further reduce variance and keep the gradient estimator unbiased. In Appendix, we illustrate these methods in SCGs. We find that the crux of the matter is to open up a differentiable path from parameters to costs or surrogate objectives. It is better to utilize gradient information, even approximate, rather than a function output. All the mentioned techniques can be applied to our learned Q-functions.

5 The Big Picture of Backpropagation

Looking over the panorama of learning in an SCG, we see that the Backprop-Q framework extends backpropagation to a more general level, propagating learning signals not only across deterministic nodes but also stochastic nodes. The stochastic nodes act like repeaters, sending expected costs back through all random variables. Then, each local parameterized distribution, which is a computation subgraph consisting of many deterministic and differentiable operations, takes over the job of backpropgation and then the standard backpropgation starts. Note that these computation subgraphs can overlap by sharing parameters with each other. See an illustration in Appendix.

6 Experimental Suggestions and Concerns

SCGs can express a wide range of models in stochastic neural networks, VB and RL, which differ significantly. We provide experimental suggestions and concerns from three aspects listed below:

(1) *Choose a model to train by Backprop-Q with awareness of properties of the cost, graph structure, and types of random variables.* i) Is the cost differentiable? Does it involve SCG's distribution functions or parameters? Can it be decoupled and split into smaller costs? For example, think of the ELBO optimized in variational inference, and compare it with the discrete metric BLEU used in machine translation. ii) Does the graph contain long statistical dependencies? Does it hold only long-delayed costs, or have immediate costs? If the graph structure is flat and the delayed effect is weak, it might be better to use the MC-based actual cost value rather than that bootstrapped from learned Q-functions. iii) Is a random variable continuous or discrete? We suggest using the reparameterization trick for continuous variables if the probability is computable after transformation.

(2) *Consider the way to learn Q-functions and how the trained SCG model might be impacted by the bias and inaccuracy of learned Q-functions.* i) Linear approximators converge fast and behave consistently, but cannot fit highly nonlinear functions, resulting in large bias. Nonlinear approximators based on neural networks can be unstable and hard to train, probably with higher sample complexity than using actual returns. ii) The policy gradient theorem [21] suggests using compatible features shared by policy and critic. We speculate that this might be related to the underlying factor of how Q-functions impact the SCG model, that is, how good of teaching signals Q-functions can offer might be more important than how well they fit the exact expected costs. iii) The sample updates to fit Q-functions may be correlated similarly to RL. We consider using experience replay and separate target networks to smooth data distribution for training Q-functions.

(3) *Consider the way to utilize Q-functions.* i) A simple implementation is to treat a Q-function as a local cost, yielding a low-variance gradient estimator by applying one of the methods proposed in VB. However, the estimator is always biased, relying on how well the Q-function approximates to the exact expected cost. ii) We can treat a Q-function as a control variate to reduce the variance caused by actual returns, and correct the bias by a differentiable term based on this Q-function. See Appendix.

7 Related Work

Schulman [30] introduced a framework of automatic differentiation through standard backpropagation in the context of SCGs. Inspired by this work, we conduct a comprehensive survey from three areas.

Backpropagation-related learning: The backpropagation algorithm, proposed in [1], can be viewed as a way to address the credit assignment problem, where the "credit" is represented by a signal of back-propagated gradient. Instead of gradients, people studied other forms of learning signals and other ways to assign them. [2, 3] compute targets rather than gradients using a local denoising auto-encoder at each layer and building both feedforward and feedback pathways. [4, 5] show that even random feedback weights can deliver useful learning signals to preceding layers, offering a biologically plausible learning mechanism. [6] uses synthetic gradients as error signals to work with backpropagation and update independently and asynchronously.

Policy gradient and critic learning in RL: Policy gradient methods offer stability but suffer from high variance and slow learning, while TD-style critic (or value function) learning are sample-efficient but biased and sometimes nonconvergent. Much work has been put to address such issues. For policy gradient, people introduced a variety of approaches, including controlled policy optimization (TRPO,PPO) [27, 28], deterministic policy gradient (DPG,DDPG) [33, 23], generalized advantage estimation (GAE) [26] and control variates (Q-Prop) [24]. TRPO and PPO constrain the change in the policy to avoid an excessively large policy update. DPG and DDPG, based on off-policy actor-critic, enable the policy to utilize gradient information from action-value function for continuous cases. GAE generalizes the advantage policy gradient estimator, analogous to TD(λ). For critic learning, people focused mainly on the use of value function approximators [21] instead of actual returns [20]. The TD learning [19], aided by past experience, is used widely. When using large neural networks to train action-value functions, DQN [22] uses experience replay and a separate target network to break correlation of updates. A3C [25] proposes an asynchronous variant of actor-critic using a shared and slow-changing target network without experience replay. DDPG and Q-Prop inherit these two techniques to exploit off-policy samples fully and gain sample efficiency and model consistency.

Gradient estimators in VB: The problem of optimizing over distribution parameters has been studied a lot in VB. The goal is to obtain an unbiased and lower-variance gradient estimator. The basic estimator is the REINFORCE [20], also known as the score-function [34] or likelihood-ratio estimator [35]. With the widest applicability, it is unbiased but suffer from high variance. The main approaches for variance reduction are the reparameterization trick [8, 9, 10] and control variates [11, 12, 13]. The former is applicable to continuous variables with a differentiable cost. It is typically used with Gaussian distribution [8, 9]. [10] proposed a generalized reparameterization gradient for a wider class of distributions. To reparameterize discrete variables, [14, 15] introduced the Concrete and the Gumbel-Softmax distribution respectively to build relaxed models but bringing in bias. The latter is suitable for both continuous and discrete variables. A control variate can be an input-dependent term, known as baseline [11], or a sample-dependent term with an analytic expectation [12, 13]. It may obtain higher variance than the former in practice, intuitively because it cannot utilize gradient information of the cost but an outcome. Other variance reduction methods include local expectation gradients [36] and straight-through estimator [7]. Recently, new advanced estimators have been proposed with lower variance and being unbiased. MuProp [16] uses the first-order Taylor expansion as a control variate, leaving the deterministic term computed by a mean-field network. Its model-free version for RL, Q-Prop [24], uses the similar technique combined with off-policy critic-learning by experience replay. Rebar [17] and RELAX [18] aim at deriving estimators for discrete variables. Unlike Rebar, RELAX learns a free-form control variate parameterized by a neural network.

8 Conclusion

In this paper, we propose a framework of generalized backpropagation for arbitrary stochastic computation graphs, enabling propagated signals to go across stochasticity and beyond gradients.

References

[1] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[2] Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, Thomas Mesnard, and Zhouhan Lin. Towards biologically plausible deep learning. *arXiv preprint arXiv:1502.04156*, 2015.

[3] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[4] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

[5] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 1037–1045, 2016.

[6] Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, and Koray Kavukcuoglu. Decoupled neural interfaces using synthetic gradients. *arXiv preprint arXiv:1608.05343*, 2016.

[7] Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

[8] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[9] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014.

[10] Francisco R Ruiz, Michalis Titsias RC AUEB, and David Blei. The generalized reparameterization gradient. In *Advances in Neural Information Processing Systems*, pages 460–468, 2016.

[11] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*, 2014.

[12] John Paisley, David Blei, and Michael Jordan. Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*, 2012.

[13] Rajesh Ranganath, Sean Gerrish, and David Blei. Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822, 2014.

[14] 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.

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

[16] Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. Muprop: Unbiased backpropagation for stochastic neural networks. *arXiv preprint arXiv:1511.05176*, 2015.

[17] George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *Advances in Neural Information Processing Systems*, pages 2624–2633, 2017.

[18] Will Grathwohl, Dami Choi, Yuhuai Wu, Geoff Roeder, and David Duvenaud. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *arXiv preprint arXiv:1711.00123*, 2017.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction (2nd Edition, in preparation)*. MIT Press, 2017.

[20] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Reinforcement Learning*, pages 5–32. Springer, 1992.

[21] 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*, pages 1057–1063, 2000.

[22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[23] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[24] 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.

[25] 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*, pages 1928–1937, 2016.

[26] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[27] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International Conference on Machine Learning*, pages 1889–1897, 2015.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] R Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, PhD thesis, University of Massachusetts, Amherst, MA Google Scholar, 1984.

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

[31] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. *arXiv preprint arXiv:1511.06732*, 2015.

[32] George Casella and Christian P Robert. Rao-blackwellisation of sampling schemes. *Biometrika*, 83(1):81–94, 1996.

[33] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *ICML*, 2014.

[34] Michael C Fu. Gradient estimation. *Handbooks in operations research and management science*, 13:575–616, 2006.

[35] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

[36] Michalis K Titsias. Local expectation gradients for doubly stochastic variational inference. *arXiv preprint arXiv:1503.01494*, 2015.

[37] Robert "Dr. Bob" Gardner. Bernstein inequalities for polynomials. 2013.

[38] Z. Ditzian. Multivariate bernstein and markov inequalities. *J. Approx. Theory*, 70(3):273–283, September 1992.

Appendix

1 Proofs

Theorem 4. *(SCG's gradient estimators) Given an SCG with a cost function f defined on $\mathcal{Z} \subseteq \mathcal{X}$, and each random variable associated with its own distribution parameter such that $X \sim p(\cdot|Pa_X; \theta_X)$, the gradient of the expected total cost J with respect to θ_X can be written as:*

$$\nabla_{\theta_X} J = \mathbb{E}_{An_X, X} \left[\nabla_{\theta_X} \log p(X|Pa_X; \theta_X) \cdot Q_X(Fr_{An_X \cup \{X\}}) \right] \quad (5)$$

where Pa_X is the set of X's parents, An_X the set of X's ancestors and $Fr_V \subseteq V$ the frontier [5] of a set of random variables V, defined as: a subset of random variables from which the cost is reachable through random variables not in V. We also define a Q-function for each stochastic node, representing the expected cost depending on this random variable and its necessary ancestors such that:

$$Q_X(Fr_{An_X \cup \{X\}}) := \mathbb{E}_{Z|Fr_{An_X \cup \{X\}}}[f(\mathcal{Z})] \quad (6)$$

Proof. First, we rewrite the objective function for the SCG $(\mathcal{X}, \mathcal{G}_\mathcal{X}, \mathcal{P}, \Theta, \mathcal{F}, \Phi)$ by unfolding the whole expectation computation and splitting it into three parts, with the expectation on X in the middle, as:

$$\begin{aligned} J(\Theta, \Phi) &= \mathbb{E}_{An_X} \left[\mathbb{E}_{X|Pa_X} \left[\mathbb{E}_{\mathcal{Z}|An_X \cup \{X\}}[f(\mathcal{Z})] \right] \right] \\ &= \mathbb{E}_{An_X} \left[\mathbb{E}_{X|Pa_X} \left[\mathbb{E}_{\mathcal{Z}|Fr_{An_X \cup \{X\}}}[f(\mathcal{Z})] \right] \right] \end{aligned} \quad (7)$$

The second line follows the Markov property that given the frontier set $Fr_{An_X \cup \{X\}}$ the rest ancestors of X have no impact on the cost. Then, we write the conditional distribution function $p(x|Pa_X; \theta_X)$ explicitly in J:

$$J(\Theta, \Phi) = \mathbb{E}_{An_X} \left[\sum_x p(x|Pa_X; \theta_X) \cdot \left[\mathbb{E}_{\mathcal{Z}|Fr_{An_X \cup \{x\}}}[f(\mathcal{Z})] \right] \right] \quad (8)$$

Note that we can change the sum into an integral for the continuous case. Then, we can derive the gradient of J with respect to the distribution parameter θ_X as follows:

$$\begin{aligned} \nabla_{\theta_X} J(\Theta, \Phi) &= \mathbb{E}_{An_X} \left[\sum_x \nabla_{\theta_X} p(x|Pa_X; \theta_X) \cdot \mathbb{E}_{\mathcal{Z}|Fr_{An_X \cup \{x\}}}[f(\mathcal{Z})] \right] \\ &= \mathbb{E}_{An_X} \left[\sum_x p(x|Pa_X; \theta_X) \nabla_{\theta_X} \log p(x|Pa_X; \theta_X) \cdot \mathbb{E}_{\mathcal{Z}|Fr_{An_X \cup \{x\}}}[f(\mathcal{Z})] \right] \\ &= \mathbb{E}_{An_X} \left[\mathbb{E}_{X|Pa_X} \left[\nabla_{\theta_X} \log p(x|Pa_X; \theta_X) \cdot \mathbb{E}_{\mathcal{Z}|Fr_{An_X \cup \{X\}}}[f(\mathcal{Z})] \right] \right] \\ &= \mathbb{E}_{An_X, X} \left[\nabla_{\theta_X} \log p(x|Pa_X; \theta_X) \cdot \mathbb{E}_{\mathcal{Z}|Fr_{An_X \cup \{X\}}}[f(\mathcal{Z})] \right] \end{aligned} \quad (9)$$

The above result is an instance of the REINFORCE [20] when we apply it to stochastic nodes in an SCG. $\qquad\square$

Theorem 5. *(Convergence of learned Q-functions) Given a Backprop-Q network with one cost as the root, if the expected squared error between each learned Q_{w_X} and its parent Q_{w_Y} can be bounded by ϵ ($\epsilon > 0$) such that $\mathbb{E}_{An_Y, Y}[(Q_{w_X}(Sc_X) - Q_{w_Y}(Sc_Y))^2] \le \epsilon$, then we have:*

$$\mathbb{E}_{An_X, X} \left[(Q_{w_X}(Sc_X) - Q_X(Sc_X))^2 \right] \le (3 \cdot 2^{l_{Q_X} - 1} - 2)\epsilon \quad for\ l_{Q_X} \ge 1 \quad (10)$$

where $Q_X(Sc_X)$ represents the true expected cost and l_{Q_X} the length of the path from Q_X to the root.

Proof. Let $Q_f(\mathcal{Z}) := f(\mathcal{Z})$ be the root, where $f(\mathcal{Z})$ is a cost function defined on random variables $\mathcal{Z}$. For node Q_X, we can find a path from it to Q_f, denoted as $(Q_{X^l}, Q_{X^{l-1}}, \ldots, Q_{X^0})$ where l is the length of the path, $Q_{X^l} := Q_X$ and $Q_{X^0} := Q_f$. Since we know

$$Q_{X^i}(Sc_{X^i}) = \mathbb{E}_{V^{i-1}|Sc_{X^i}}\left[Q_{X^{i-1}}(Sc_{X^{i-1}})\right] = \mathbb{E}_{V^{i-1}|An_{X^i}, X^i}\left[Q_{X^{i-1}}(Sc_{X^{i-1}})\right] \quad (11)$$

---

[5]In a multi-cost SCG, a cost f must be specified for a frontier, denoted as Fr_V^f

where $V^{i-1} = Sc_{X^{i-1}} - An_{X^i} \cup \{X^i\}$, we thus can derive the following inequalities:

$$
\begin{aligned}
&\mathbb{E}_{An_{X^i},X^i}\Big[\big(Q_{w_{X^i}}(Sc_{X^i}) - Q_{X^i}(Sc_{X^i})\big)^2\Big] \\
=&\mathbb{E}_{An_{X^i},X^i}\Big[\big(Q_{w_{X^i}}(Sc_{X^i}) - \mathbb{E}_{V^{i-1}|An_{X^i},X^i}\big[Q_{w_{X^{i-1}}}(Sc_{X^{i-1}})\big] \\
&\qquad\qquad + \mathbb{E}_{V^{i-1}|An_{X^i},X^i}\big[Q_{w_{X^{i-1}}}(Sc_{X^{i-1}})\big] - \mathbb{E}_{V^{i-1}|An_{X^i},X^i}\big[Q_{X^{i-1}}(Sc_{X^{i-1}})\big]\big)^2\Big] \\
\leq&\mathbb{E}_{An_{X^i},X^i}\Big[2\big(Q_{w_{X^i}}(Sc_{X^i}) - \mathbb{E}_{V^{i-1}|An_{X^i},X^i}\big[Q_{w_{X^{i-1}}}(Sc_{X^{i-1}})\big]\big)^2 \\
&\qquad\qquad + 2\big(\mathbb{E}_{V^{i-1}|An_{X^i},X^i}\big[Q_{w_{X^{i-1}}}(Sc_{X^{i-1}}) - Q_{X^{i-1}}(Sc_{X^{i-1}})\big]\big)^2\Big] \\
\leq&2\mathbb{E}_{An_{X^{i-1}},X^{i-1}}\Big[\big(Q_{w_{X^i}}(Sc_{X^i}) - Q_{w_{X^{i-1}}}(Sc_{X^{i-1}})\big)^2\Big] \\
&+ 2\mathbb{E}_{An_{X^{i-1}},X^{i-1}}\Big[\big(Q_{w_{X^{i-1}}}(Sc_{X^{i-1}}) - Q_{X^{i-1}}(Sc_{X^{i-1}})\big)^2\Big] \\
\leq&2\epsilon + 2\mathbb{E}_{An_{X^{i-1}},X^{i-1}}\Big[\big(Q_{w_{X^{i-1}}}(Sc_{X^{i-1}}) - Q_{X^{i-1}}(Sc_{X^{i-1}})\big)^2\Big]
\end{aligned}
\tag{12}
$$

Let $h_i = \mathbb{E}_{An_{X^i},X^i}\big[\big(Q_{w_{X^i}}(Sc_{X^i}) - Q_{X^i}(Sc_{X^i})\big)^2\big]$, indicating the deviation of a learned Q-function from the true one , and then we have the recursion inequality: $h_i \leq 2h_{i-1} + 2\epsilon$. Because we do not need to approximate the root Q_f which is explicitly known as f, we have $h_0 = 0$. Therefore, we can eventually obtain: $h_i \leq (3 \cdot 2^{i-1} - 2)\epsilon$ for $i \geq 1$. $\qquad\square$

Theorem 6. *(Merging Backprop-Q networks) Two Backprop-Q networks can be merged at stochastic node X and its ancestors, if the two are fully matched from X through X's ancestors, that is, the set of the incoming edges to each ancestor in a Backprop-Q network is exactly matched to the other.*

Proof. This theorem is not difficult to prove. In a Backprop-Q network, the incoming edges to a Q-function node define its update rule. For a stochastic node, e.g., Y, if there are two Backprop-Q networks with respect to cost f_1 and f_2 respectively, both of which go through Y, node Y will hold two Q-functions, denoted as $Q_Y^{f_1}$ and $Q_Y^{f_2}$. If the incoming edges to $Q_Y^{f_1}$ and those to $Q_Y^{f_2}$ are exactly the same, then the update rule of $Q_Y^{f_1}$, whether to take the sum or the average, would follow the same pattern as that of $Q_Y^{f_2}$. Therefore, we can treat the two Q-functions as one by summing them, denoted as $Q_Y^{f_1,f_2}$. We wish to propagate the new $Q_Y^{f_1,f_2}$ instead of the two functions $Q_Y^{f_1}$ and $Q_Y^{f_2}$, so that we need to make sure that all Y's ancestor nodes could combine Q-functions in the same way. If the two Backprop-Q networks are fully matched over these ancestors, the merging for the downstream parts of the two networks, starting at Y, can occur through all Y's ancestors. Actually, we can start the merging a little bit earlier, from the stochastic node X one step after Y in the SCG. $\qquad\square$

2 Gradient Difference Between Two Locally Fitted Functions

When using a neural-network-based function approximator $Q_w(x)$ to fit $Q(x)$, we wish to know to what degree $Q_w(x)$ also preserves the first-order derivative information of $Q(x)$. If we could bound the difference of the gradients w.r.t. input x between them, such that $\|\partial Q_w/\partial x - \partial Q/\partial x\| \leq \epsilon$ for all x, and $\epsilon \to 0$ when $Q_w(x) \to Q(x)$ for all x, we can utilize the gradient of $Q_w(x)$ as well as its function value, and treat $\partial Q_w/\partial x$ the same as $\partial Q/\partial x$. If we consider using the reparameterization trick for continuous or relaxed discrete random variables in some cases, the approximate gradient $\partial Q_w/\partial x$ can provide useful information even if the true $\partial Q/\partial x$ is unknown. However, this is not true universally. For example, a zigzag line, which is a piecewise linear function, can be infinitely close to a straight line but still keep its slope staying constant almost everywhere. Therefore, we need to impose some conditions to make the bounded gradient difference converge to zero. First, we propose a reasonable hypothesis on the function behavior of a neural network around a point.

Hypothesis 1. *The functionality of a neural network $f(x)$ in a local region $x \in \Omega$ can be expressed fully by a family of polynomials P of a finite degree n.*
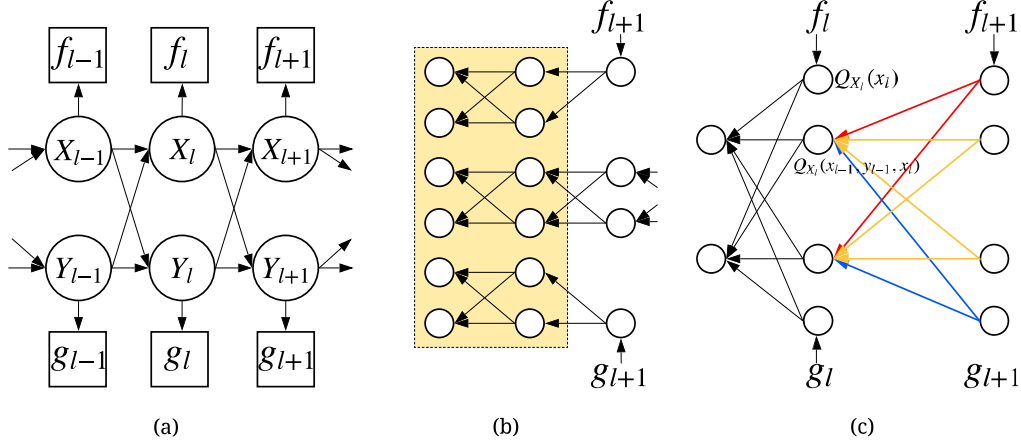
Figure 4: Reducing Backprop-Q networks for full-connected-layered SCGs

This hypothesis assumes that the degree of the non-linearity of a neural network can be bounded locally. Then, we introduce the Bernstein's inequality [37], which works as the theory basis to bound the gradient difference.

Theorem 7. *(Bernstein's inequality) Let P be a polynomial of degree n with derivative P'. Then,*

$$\max_{|z| \leq 1}(|P'(z)|) \leq n \cdot \max_{|z| \leq 1}(|P(z)|) \tag{13}$$

The above polynomial is defined on a scalar variable. The multivariate version of Bernstein's inequality can be found in [38]. It shows that the magnitude (defined by L_p norm) of the first-order derivative of a polynomial of degree n, in a bounded convex region such as $|z| \leq 1$, can be bounded by the product of a constant relying on n and the magnitude of the polynomial's value.

For simplicity, we consider a univariate case. Let $f(x) := Q_w(x) - Q(x)$ and we wish to bound $|f'(x)| = |Q'_w(x) - Q'(x)|$ by $|f(x)| = |Q_w(x) - Q(x)|$ for all $x \in [x_0 - \Delta x, x_0 + \Delta x]$. According to the above hypothesis, we can express $f(x)$ in a form of a polynomial of a finite degree n: $P(z) := f(x)$ where $z = \frac{x - x_0}{\Delta x}$ and $|z| \leq 1$, and then have:

$$|f'(x)| = \left| \frac{1}{\Delta x} P'(z) \right|_{z = \frac{x - x_0}{\Delta x}} \leq \frac{n}{\Delta x} \cdot \max_{|z| \leq 1}(|P(z)|) = \frac{n}{\Delta x} \cdot \max_{x \in [x_0 - \nabla x, x_0 + \nabla x]}(|f(x)|) \tag{14}$$

We view $n/\Delta x$ as a constant C. Therefore, if we fit $Q_w(x)$ to $Q(x)$ well enough in a local region, that is $|Q_w(x) - Q(x)| \leq \epsilon$ for all $x \in \Omega$, we can bound their gradient difference by $C\epsilon$ everywhere within this local region, converging to zero when $\epsilon \to 0$.

3 Reduced Backprop-Q networks for fully-connected-layered SCGs

Figure 4(a) shows a multi-layer stochastic system, with each random variable relying on the entire previous stochastic layer, e.g., $X_l \sim p_{x_l}(\cdot | X_{l-1}, Y_{l-1})$, and also associated with a cost $f_l(X_l)$. Suppose there are N layers and M nodes per layer, so that $N \cdot M$ costs will provide $N \cdot M$ separate Backprop-Q networks, and a node in layer t needs to maintain $(N - t)M + 1$ Q-functions. However, from Figure 4(b), we can see that all the Backprop-Q networks rooted in layer $l + 1$ and higher layers share the exactly same downstream subgraphs from layer l. This means, at each node in layer l, we can combine all the Q-functions provided by upstream Backprop-Q networks, into one Q-function like $Q_{X_l}(x_{l-1}, y_{l-1}, x_l)$. It takes red, yellow and blue incoming edges as shown in Figure 4(c), representing three different cost sources. Therefore, for each stochastic node X, we only need to maintain two Q-functions, one for the immediate cost defined on itself, one for the combined Q-functions from upstream. Further, we do not have to construct a new Q-function for the immediate cost but use it directly. As a result, each stochastic node X only stores one Q-function that is $Q_{X_l}(x_{l-1}, y_{l-1}, x_l)$.
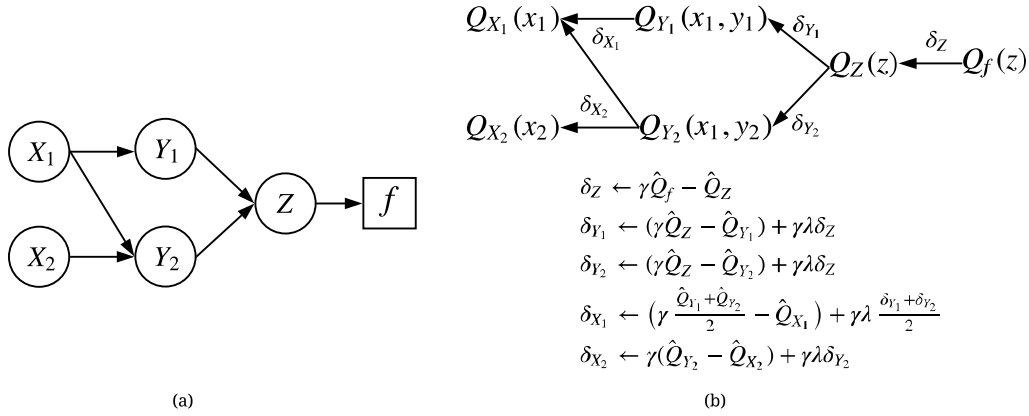
13

Figure 5 diagram (a): nodes $X_1 \rightarrow Y_1$, $X_2 \rightarrow Y_2$, $X_1 \rightarrow Y_2$, $Y_1, Y_2 \rightarrow z \rightarrow f$

Figure 5 diagram (b): Backprop-Q network with propagated errors

$Q_{X_1}(x_1) \xleftarrow{\delta_{X_1}} Q_{Y_1}(x_1, y_1) \xleftarrow{\delta_{Y_1}} Q_Z(z) \xleftarrow{\delta_Z} Q_f(z)$

$Q_{X_2}(x_2) \xleftarrow{\delta_{X_2}} Q_{Y_2}(x_1, y_2) \xleftarrow{\delta_{Y_2}}$

$$\delta_Z \leftarrow \gamma\hat{Q}_f - \hat{Q}_Z$$

$$\delta_{Y_1} \leftarrow (\gamma\hat{Q}_Z - \hat{Q}_{Y_1}) + \gamma\lambda\delta_Z$$

$$\delta_{Y_2} \leftarrow (\gamma\hat{Q}_Z - \hat{Q}_{Y_2}) + \gamma\lambda\delta_Z$$

$$\delta_{X_1} \leftarrow \left(\gamma\,\frac{\hat{Q}_{Y_1}+\hat{Q}_{Y_2}}{2} - \hat{Q}_{X_1}\right) + \gamma\lambda\,\frac{\delta_{Y_1}+\delta_{Y_2}}{2}$$

$$\delta_{X_2} \leftarrow \gamma(\hat{Q}_{Y_2} - \hat{Q}_{X_2}) + \gamma\lambda\delta_{Y_2}$$

(a)   (b)

Figure 5: λ-return updates on an SCG with one cost

Figure 6 diagram (a): f_a above; $X_1 \rightarrow X_2 \rightarrow X_3 \rightarrow X_4$; f_b below.

Figure 6 diagram (b):

$Q_{f_a}(x_2, x_3)$

$Q_{X_1}(x_1) \xleftarrow{\delta_{X_1}} Q_{X_2}(x_2) \xleftarrow{\delta_{X_2}} Q_{X_3}(x_2, x_3) \xleftarrow{\delta_{X_3}} Q_{X_4}(x_3, x_4)$

$\xleftarrow{\delta_{X_4}} Q_{f_b}(x_3, x_4)$

$$\delta_{X_4} \leftarrow \gamma\hat{Q}_{f_b} - \hat{Q}_{X_4}$$

$$\delta_{X_3} \leftarrow \left(\gamma(\hat{Q}_{f_a} + \hat{Q}_{X_4}) - \hat{Q}_{X_3}\right) + \gamma\lambda\delta_{X_4}$$

$$\delta_{X_2} \leftarrow (\gamma\hat{Q}_{X_3} - \hat{Q}_{X_2}) + \gamma\lambda\delta_{X_3}$$

$$\delta_{X_1} \leftarrow (\gamma\hat{Q}_{X_2} - \hat{Q}_{X_1}) + \gamma\lambda\delta_{X_2}$$
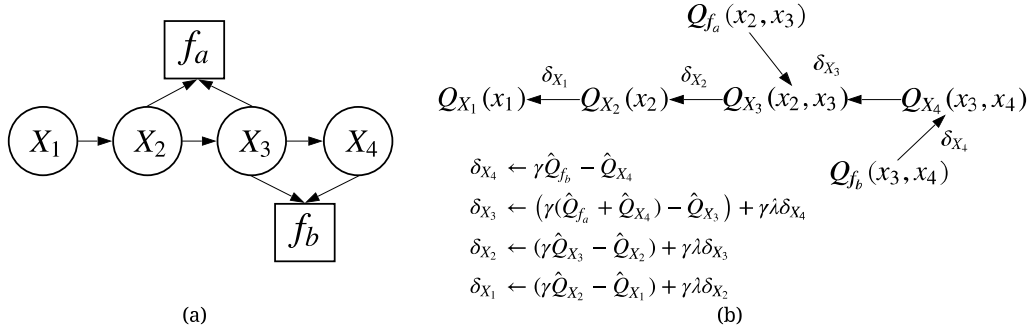
(a)   (b)

Figure 6: λ-return updates on an SCG with two costs

4 Using Techniques from RL for Backprop-Q

4.1 Cases for λ-return Updates

We show two cases to illustrate our graph-based λ-return method. Figure 5(a) is an SCG with one cost. Its corresponding Backprop-Q network is given in Figure 5(b), with propagated errors pointing to each node. The start error δ_z is computed by $\gamma Q_f(z) - Q_z(z)$ based on a sample z, where γ is the discount factor. For simplicity of notation, we use $\hat{Q}_f := Q_f(z)$ and $\hat{Q}_z := Q_z(z)$. Then, each of the following errors is a sum of the current TD error and its upstream errors, like $\delta_{Y_1} \leftarrow (\gamma\hat{Q}_Z - \hat{Q}_{Y_1}) + \gamma\lambda\delta_z$, where the second term is weighted by $\gamma\lambda$. For node Q_{X_1}, it has two upstream nodes Q_{Y_1} and Q_{Y_2} belonging to the same cost source. Therefore, we compute its TD error based on the averaged update target $(\hat{Q}_{Y_1} + \hat{Q}_{Y_2})/2$, and also average the two upstream errors $(\delta_{Y_1} + \delta_{Y_2})/2$.

The second case is a two-cost SCG in Figure 6(a). Before applying λ-return, we reduce its Backprop-Q networks into a simpler one, by removing edges $Q_{f_a} \rightarrow Q_{X_2}^{f_a}$ and $Q_{f_b} \rightarrow Q_{X_3}^{f_b}$ and merging the rest, as shown in Figure 6(b). The procedure is much like the first case, except that at node Q_{X_3} we sum the two upstream Q-function values instead of averaging them due to different cost sources.

Taking δ_{X_1} as an example, in the first case, $\delta_{X_1} = \gamma^3\hat{Q}_f - \hat{Q}_{X_1}$ if $\lambda = 1$, and $\delta_{X_1} = \gamma(\hat{Q}_{Y_1} + \hat{Q}_{Y_2})/2 - \hat{Q}_{X_1}$ if $\lambda = 0$; in the second case, $\delta_{X_1} = \gamma^3\hat{Q}_{f_a} + \gamma^4\hat{Q}_{f_b} - \hat{Q}_{X_1}$ if $\lambda = 1$, and $\delta_{X_1} = \gamma\hat{Q}_{X_2} - \hat{Q}_{X_1}$ if $\lambda = 0$. This gives us a more flexible way to make a compromise between bias and variance. When being at an early phase of training, we set λ and γ close to 1, so that the remote cost signal can propagate backward faster; after training Q-functions for a while, we decrease λ a little bit to reduce variance by relying more on learned Q-functions and thus cumulative effect of past experience.
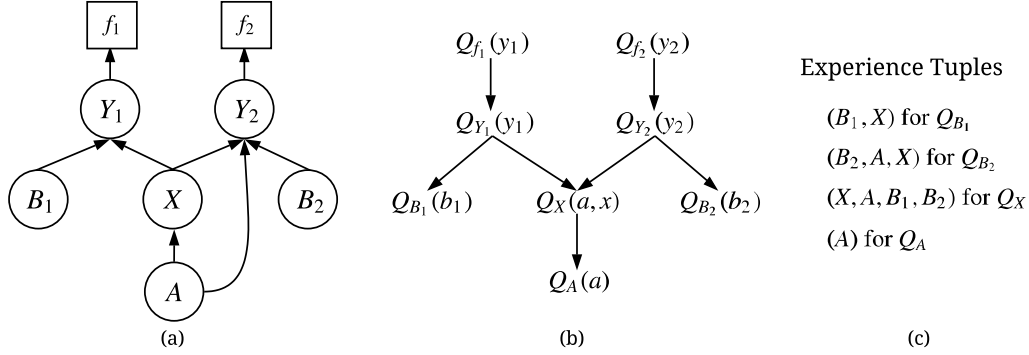
14

Figure 7: Graph-based experience replay in an SCG

4.2 Cases for Experience Replay

For the purpose of illustration, we consider an SCG with two costs shown in Figure 7(a). For simplicity, we remove edge $Q_{Y_2}^{f_2} \rightarrow Q_A^{f_2}$ to get a simplified Backprop-Q network in Figure 7(b). We generate and store four types of experience tuples for nodes Q_{B_1}, Q_{B_2}, Q_X and Q_A respectively, as shown in Figure 7(c). Taking Q_X as an example, its experience tuple should contain X and A for the scope of Q_X, and also include B_1 and B_2 to generate Y_1 and Y_2 respectively, together with X. Given an experience sample (x, a, b_1, b_2), a sample y_1 should be drawn from $p_{Y_1}(\cdot|x, b_1; \theta_{Y_1})$, and y_2 drawn from $p_{Y_2}(\cdot|x, b_2; \theta_{Y_2})$, both based on the current policy parameters. This sampling process can be performed multiple times to generate many (y_1, y_2) for training Q_{w_X} by taking gradient steps to minimize:

$$L(w_X) = \frac{1}{n} \sum_{i=1}^{n} \left(Q_{w_{Y_1}}(y_1^{(i)}) + Q_{w_{Y_2}}(y_2^{(i)}) - Q_{w_X}(a, x) \right)^2 \tag{15}$$

4.3 Details for Slow-tracking Target.

DQN [22] uses a separate network for generating the targets for the Q-learning updates, which takes parameters from some previous iteration and updates periodically. DDPG [23] uses "soft" target updates rather than directly coping the weights. Our solution, called slow-tracking target, is similar to target updates in DDPG by having the target network slowly track the learned network. This can be applied to both policy parameters and critic parameters when performing experience replay. For each parameter, we maintain θ_t and $\Delta\theta_t$, where θ_t represents the parameter of the target network, and $\theta_t + \Delta\theta_t$ represents the parameter of the current learned network. We suppose that the varying of θ_t is slow while $\Delta\theta_t$ can change drastically. Each time we obtain a new Δ, we add it to $\Delta\theta_t$ as: $\Delta\theta_{t+1} \leftarrow \Delta\theta_t + \Delta$, and then we let θ_t slowly track the new $\Delta\theta_{t+1}$ as: $\theta_{t+1} \leftarrow \theta_t + \alpha\Delta\theta_{t+1}$ and $\Delta\theta_{t+1} \leftarrow (1 - \alpha)\Delta\theta_{t+1}$ with a positive $\alpha \ll 1$.

4.4 Cases for Advantage Functions.

In reinforcement learning, the action-value function summarizes the performance of each action from a given state, assuming it follows π thereafter, while the advantage function provides a measure of how each action compares to the average performance at state s_t given by the state-value function. The advantage function is often used to replace the action-value function in the gradient estimator to yield much lower variance [25, 26, 24]. It is viewed as an instance of the baseline method, centering the learning signal and reducing variance significantly. To derive the SCG version of advantage functions, we take Figure 8(a) as an example. Here, suppose that Y is action and X is state, so that $Q_Y(y)$ represents the performance for taking action y, and $Q_X(x) = \mathbb{E}_{Y|x}[Q_Y(Y)]$ represents the average performance for taking all actions at state x. Therefore, the advantage function at Y should be $A_Y(x, y) = Q_Y(y) - Q_X(x)$. If Y has two states as in Figure 8(b), each of $Q_{X_1}(x_1)$ and $Q_{X_2}(x_2)$ gives an evaluation of the average performance at its own state. We thus subtract the two's average from $Q_Y(y)$ to compute the advantage function $A_Y(x_1, x_2, y)$ at Y. In Figure 8(c)(d), the advantage functions become more complex, requiring us to consider other branches. For example, in Figure 8(c), as $Q_X(x)$ takes the sum of $Q_{Y_1}(y_1)$ and $Q_{Y_2}(y_2)$ as its update target, when computing $A_{Y_1}(x, y_1)$ at
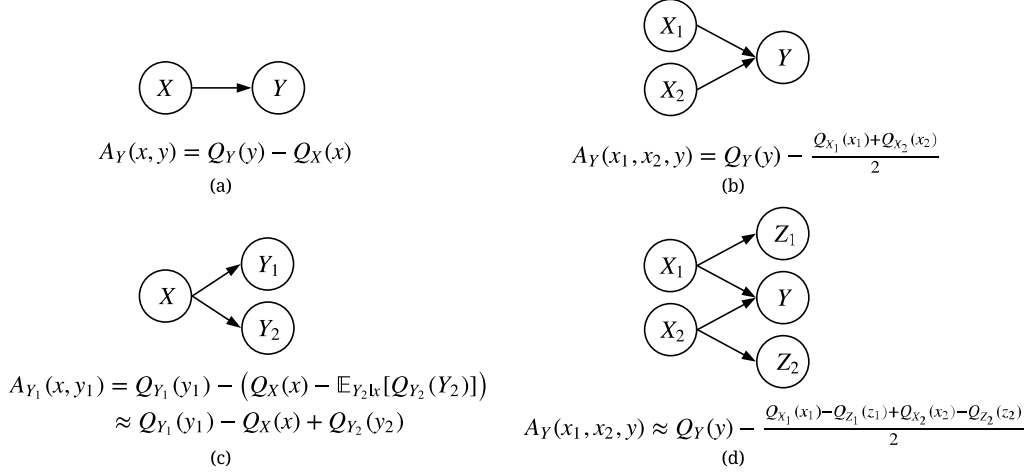
(a)
$$A_Y(x, y) = Q_Y(y) - Q_X(x)$$

(b)
$$A_Y(x_1, x_2, y) = Q_Y(y) - \frac{Q_{X_1}(x_1) + Q_{X_2}(x_2)}{2}$$

(c)
$$A_{Y_1}(x, y_1) = Q_{Y_1}(y_1) - \left(Q_X(x) - \mathbb{E}_{Y_2|x}[Q_{Y_2}(Y_2)]\right)$$
$$\approx Q_{Y_1}(y_1) - Q_X(x) + Q_{Y_2}(y_2)$$

(d)
$$A_Y(x_1, x_2, y) \approx Q_Y(y) - \frac{Q_{X_1}(x_1) - Q_{Z_1}(z_1) + Q_{X_2}(x_2) - Q_{Z_2}(z_2)}{2}$$

Figure 8: Computing advantage functions in SCGs

$$A_{X_t}(x_{t-1}, x_t) \approx \delta_{t-1} \overset{\text{if } \gamma\lambda = 1}{=} R - Q_{w_{X_{t-1}}}(x_{t-1})$$
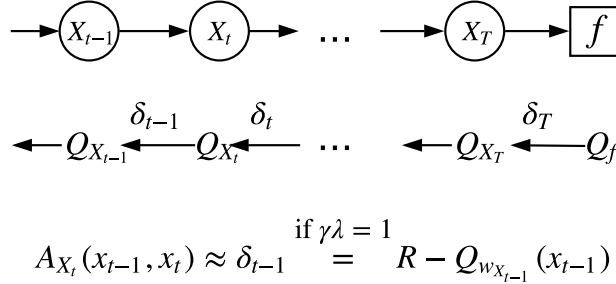
Figure 9: Approximating advantage functions using λ-return errors

Y_1, we need to subtract $\mathbb{E}_{Y_2|x}[Q_{Y_2}(Y_2)]$ from $Q_X(x)$. Here, we approximate the advantage function $A_{Y_1}(x, y_1)$ by using a sample $Q_{Y_2}(y_2)$ instead of the expectation computation. In practice, the above advantage functions are not known and must be estimated as we estimate the Q-functions by Q_w. One way is to build the approximate advantage functions directly based on Q_w, such as $A_{w_Y}(x, y) := Q_{w_Y}(y) - Q_{w_X}(x)$ for the case in Figure 8(a). Another way is to use λ-return to estimate the first term so that we can utilize the remote signal in case that Q_w is not accurate yet. Figure 9 shows that the advantage function at X_t can be approximated by the error δ_{t-1}. In the extreme case when $\gamma\lambda = 1$, $A_{X_t}(x_{t-1}, x_t)$ reduces to $R - Q_{w_{X_{t-1}}}(x_{t-1})$ where R represents the actual return.

4.5 Details for Controlled Policy Optimization.

To avoid an excessively large policy update, TRPO [27] puts a constraint on the change in the policy at each update, and guarantees policy monotonic improvement with controlled step sizes. It solves a constrained optimization problem on the policy parameters. In the context of SCGs, consider a pair (X, Pa_X) where $X \sim p(\cdot|Pa_X; \theta)$. The constrained optimization problem is:

$$
\begin{aligned}
\min_\theta \quad & \mathbb{E}_{X \sim p(\cdot|Pa_X; \theta_{\text{old}})} \left[\frac{p(X|Pa_X; \theta)}{p(X|Pa_X; \theta_{\text{old}})} Q_X(Sc_X) \right] \\
\text{s.t.} \quad & D_{KL}\left(p(\cdot|Pa_X; \theta) \| p(\cdot|Pa_X; \theta_{\text{old}}) \right) \le \delta
\end{aligned}
\tag{16}
$$

which can be solved by the conjugate gradient algorithm followed by a line search. The objective function is approximated by drawing multiple samples, while the constraint is approximated by a quadratic approximation using the Fisher information matrix.

PPO [28] introduces a much simpler way to implement the controlled policy optimization. In contrast to TRPO, it uses a clipped surrogate objective without any constraint. Given a pair (X, Pa_X) in a

16

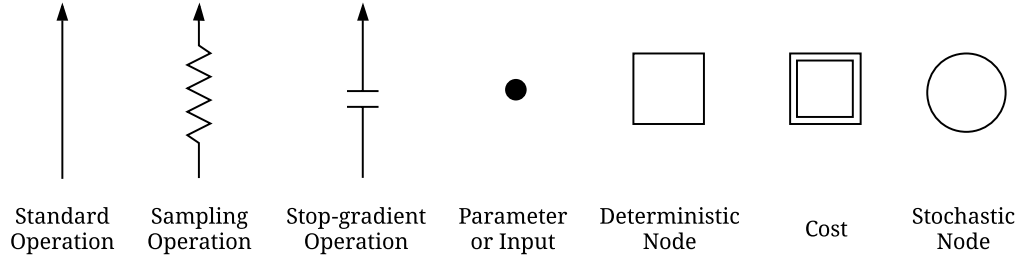| Standard Operation | Sampling Operation | Stop-gradient Operation | Parameter or Input | Deterministic Node | Cost | Stochastic Node |

Figure 10: Graphical Notation

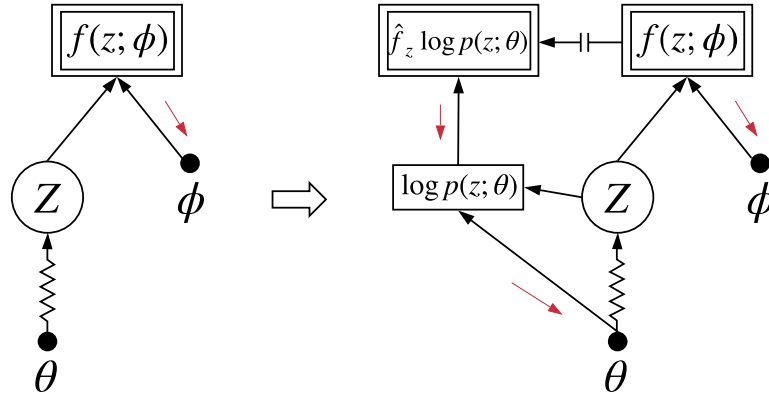
Figure 11: REINFOCE / Score-Function / Likelihood-Ratio Estimators

SCG, we write the objective as:

$$\min_{\theta} \mathbb{E}_x \Big[\max \Big(r_x(\theta) Q_x(Sc_x), \ \text{clip}\big(r_x(\theta), 1 - \epsilon, 1 + \epsilon\big) Q_x(Sc_x) \Big) \Big] \tag{17}$$

where $r_x(\theta) = \frac{p(X|Pa_X;\theta)}{p(X|Pa_X;\theta_{\text{old}})}$. The idea behind this is to remove the incentive for moving $r_x(\theta)$ outside of the interval $[1 - \epsilon, 1 + \epsilon]$.

5 Using Techniques from VB for Backprop-Q

5.1 Graphical Notation

Here, we list some gradient estimation methods, illustrated in SCGs. To explain it well, we use a new graphical notation that differentiates three types of arrows as shown in Figure 10. The arrow, called *standard operation*, represents a normal deterministic computation, producing an output when given an input. The arrow, called *sampling operation*, represents a process of drawing a sample from a distribution. The arrow, called *stop-gradient operation*, is also a deterministic mapping but with no flowing-back gradients permitted. Gradients can only be propagated backward through standard operations if not specified. For the notation of nodes, we use a double-line square to denote a cost node, and the rest follows [30].

5.2 REINFORCE / Score-Function / Likelihood-Ratio Estimators

The REINFORCE estimator [20], also known as the score-function or the likelihood-ratio estimator, has the widest applicability for almost all cases. It does not require $f(z)$ to be differentiable, and only needs a function output to estimate the gradient. Also, it is applicable to both continuous and discrete random variables. We illustrate the REINFORCE estimator using an SCG in Figure 11. The left graph shows an optimization problem containing a sampling operation, where only ϕ can
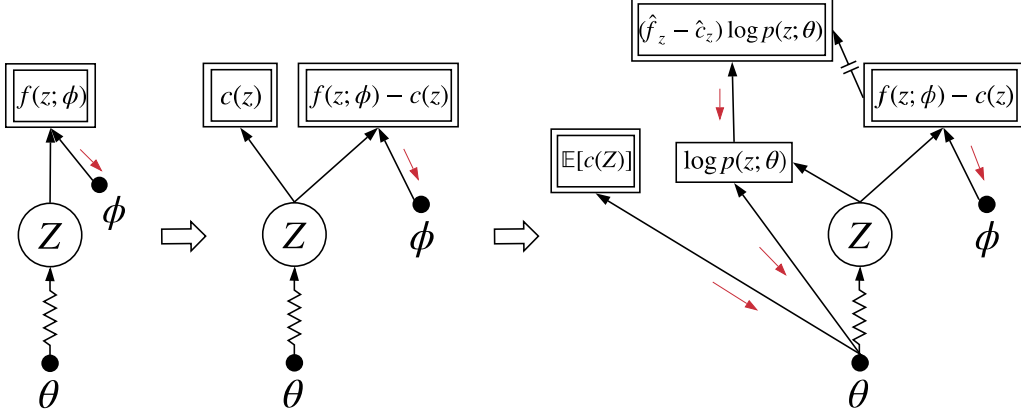
Figure 12: Control Variates

receive a signal of the cost gradient. To send a gradient signal to θ, we have to create a surrogate objective function, $\hat{f}_z \log p(z; \theta)$, taking an output of function $f(z)$, denoted as $\hat{f}_z$, with no gradients allowed to send back to $f(z)$. Then, we build a differentiable path from θ to the surrogate objective that can propagate the gradient back to θ. Note that as a Monte-Carlo-based method, a forward pass for computing $\hat{f}_z \log p(z; \theta)$ involves a sampling operation to draw z from θ, which is the source of stochasticity and thus brings variance. The REINFORCE gradient estimator w.r.t θ is written as:

$$\hat{g} := f(z) \frac{\partial}{\partial \theta} \log p(z; \theta) \tag{18}$$

5.3 Control Variates

Control variates is a variance reduction technique that helps reduce the high variance in the REINFORCE estimator. A carefully designed control variate should be a term highly correlated to $f(z)$, with a closed-form or tractable expectation to correct the bias. Figure 12 shows how a control variate works to yield an unbiased and low-variance gradient estimator. First, we subtract a term $c(z)$ from $f(z)$ and add the same one aside simultaneously to keep the total cost unbiased. Then, we create a surrogate objective $(\hat{f}_z - \hat{c}_z) \log p(z; \theta)$ the same way as in the previous subsection, reachable from θ via a differentiable path. Since $\mathbb{E}_z[c(Z)]$ can be computed analytically, we avoid the operation of sampling z to connect θ to $\mathbb{E}_z[c(Z)]$ directly, resulting in no variance when estimating the gradient of this part. To reduce variance, we wish $c(z)$ to be closely correlated to $f(z)$ so that the magnitude of $f(z) - c(z)$ could be as small as possible. There are several ways to design $c(z)$. (1) Let c be a constant, a moving average, or a function that does not rely on z [11]. Due to $\frac{\partial}{\partial \theta} \mathbb{E}_z[c(Z)] = 0$, we can remove the edge from θ to $\mathbb{E}_z[c(Z)]$. This method is often called *baseline*. (2) Let $c(z)$ be the linear Taylor expansion of $f(z)$ around $z = \mathbb{E}_z[Z]$ [16]:

$$c(z) = f(\mathbb{E}_z[Z]) + f'(z)\Big|_{z=\mathbb{E}_z[Z]} (z - \mathbb{E}_z[Z]) \tag{19}$$

Then we have:

$$\frac{\partial}{\partial \theta} \mathbb{E}_z[c(Z)] = f'(z)\Big|_{z=\mathbb{E}_z[Z]} \frac{\partial}{\partial \theta} \mathbb{E}_z[Z] \tag{20}$$

where $f'(z)\big|_{z=\mathbb{E}_z[Z]}$ is computed through a deterministic and differentiable mean-field network. Furthermore, to learn a good control variate, we minimize the expected square of the centered learning signal by $\min_c \mathbb{E}_z[(f(Z) - c)^2]$, or maximize the variance reduction by learning the best scale factor a in $\hat{f}(z) = f(z) - a \cdot c(z)$, so that $\text{Var}(\hat{f})$ has the minimal value when $a = \text{Cov}(f, c)/\text{Var}(c)$. The gradient estimator with a control variate is written as:

$$\hat{g} := (f(z) - c(z)) \frac{\partial}{\partial \theta} \log p(z; \theta) + \frac{\partial}{\partial \theta} \mathbb{E}_z[c(Z)] \tag{21}$$
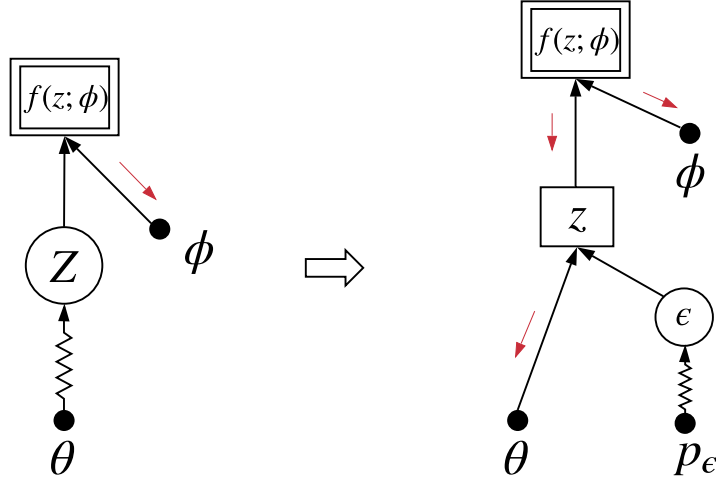
18

Figure 13: Reparameterization Trick

5.4 Reparameterization Trick

The reparameterization trick is thought to yield an unbiased gradient estimator with lower variance than that of control variates in practice. The intuition is that it takes advantage of the first-order derivative $\partial f/\partial z$, while the control variates method only uses an outcome like $\hat{f}_z$. However, the reparameterization estimator requires z to be continuous and $f(z)$ to be differentiable. Most importantly, we need to find a transformation function $z(\epsilon;\theta)$ where ϵ is a sample from a fixed known distribution and θ is the distribution parameter such that $z \sim p(\cdot;\theta)$ can follow exactly the same distribution as before. Therefore, it is typically used with Gaussian distribution [8, 9]. [10] proposed a generalized reparameterization gradient for a wider class of distributions, but it demands a sophisticated invertible transformation that it is not easy to define. The right graph in Figure 13 shows a deterministic node of z in place of the stochastic node Z, as z is computed by a function of θ and ϵ but not sampled directly. Therefore, we can propagate the gradient signal through z to θ. The reparameterization gradient estimator is written as:

$$\hat{g} := \frac{\partial f}{\partial z} \frac{\partial}{\partial \theta} z(\epsilon; \theta) \tag{22}$$

5.5 Continuous Relaxation + Reparameterization Trick

How can we apply the reparameterization trick to discrete random variables, so that we can utilize the gradient information of a cost function to further reduce variance? [14, 15] introduced the Concrete distribution and the Gumbel-Softmax distribution respectively to build relaxed discrete models. A discrete random variable can be a binary variable $B \sim$ Bernoulli(θ), or a categorical variable $B \sim$ Categorical(θ) represented by an one-hot vector. Instead of sampling b directly, we draw a continuous sample z from the Gumbel distribution as shown in Figure 14. The Gumbel random variable Z can be reparameterized by a transformation function of θ and a noise ϵ from the uniform distribution. Then b can be computed through a hard threshold function, $b = H(z)$. However, the threshold function provides zero gradients almost everywhere, blocking any upstream gradient signal. To solve it, we introduce a sigmoid function $\sigma_\lambda(z)$ with a temperature hyperparametr λ to produce a relaxed $\tilde{b}$. Instead of minimizing cost $f(b)$, we minimize $f(\tilde{b})$ and open up a differentiable path from θ to $f(\tilde{b})$, which absolutely brings in biases due to $\tilde{b}$ not being b. However, in the low temperature limit when $\lambda \to 0$, we have $\tilde{b} \to b$ and thus obtain an unbiased estimator. The gradient estimator is written as:

$$\hat{g} := \frac{\partial f}{\partial \tilde{b}} \Big|_{\tilde{b} = \sigma_\lambda(z)} \frac{\partial \sigma_\lambda}{\partial z} \frac{\partial}{\partial \theta} z(\epsilon; \theta) \tag{23}$$
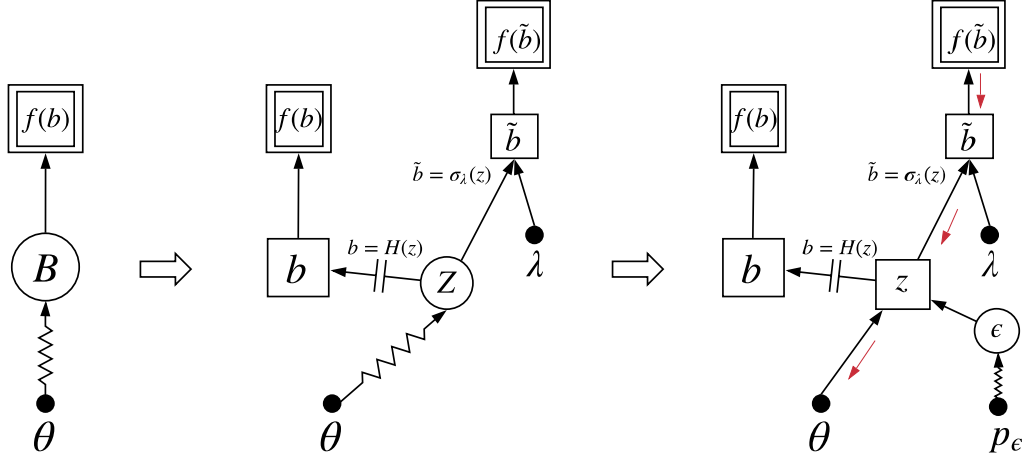
19

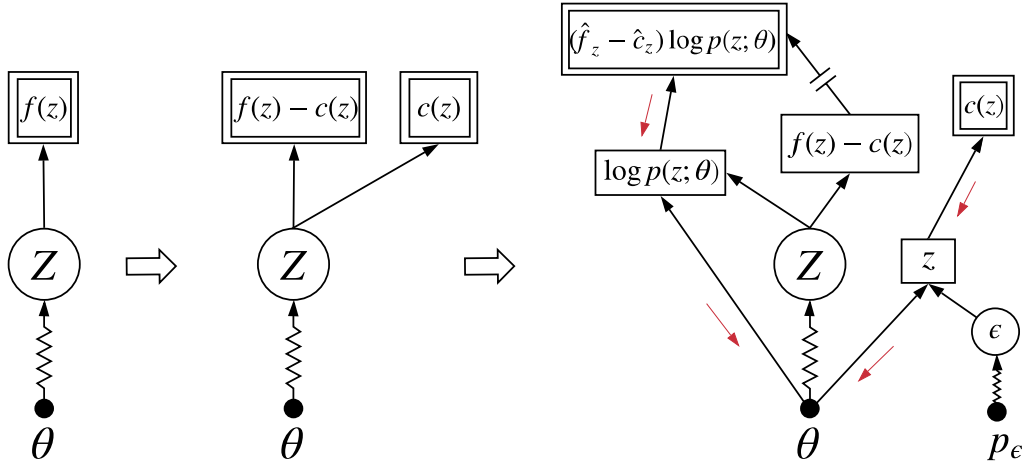Figure 14: Combining continuous relaxation with reparameterization trick.

Figure 15: Combining control variates with reparameterization trick.

5.6   Control Variates + Reparameterization Trick

For the control variates method, how well it reduces the variance depends on how correlated to $f(z)$ the control variate is. The effective way is to design a sample-dependent control variate like $c(z)$ rather than a constant or a baseline, so that it can change the value as z is changing, keeping closely correlated to $f(z)$. However, it may introduce bias, so we need a known mean of $c(z)$ to correct the bias. Unfortunately, that limits the possible forms $c(z)$ can take. Inspired by the reparameterizaton trick, we take a compromise solution that a reparameterization gradient estimator is used in place of the gradient of the true mean. That only requires $c(z)$ to be differentiable and z to be continuous. In practice, the reparameterization estimator usually yields lower variance compared to control variates. Therefore, we provide an unbiased and lower-variance gradient estimator by combining control variates with the repameterization trick [18].

In Figure 15, we suppose $f(z)$ a non-differentiable or even unknown cost function, treated as a black-box function. We can acquire no more information about f than a function output queried by an input z. We design a differentiable surrogate $c(z)$ to approximate $f(z)$ and apply it from two aspects: (1) Let $c(z)$ be a control variate, subtracted from $f(z)$ to reduce its variance. (2) Consider that $c(z)$ has its first-order derivative approximate well to that of $f(z)$, so that we can utilize the
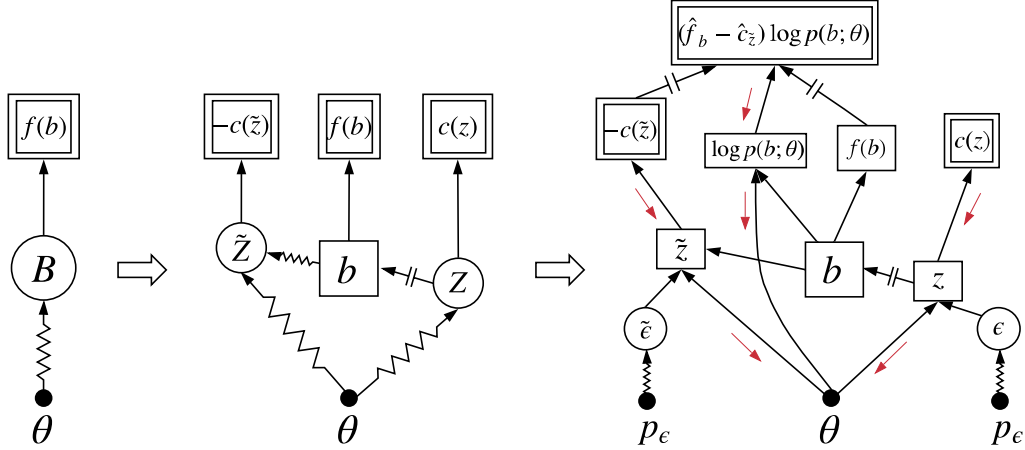
Figure 16: Combining control variates, reparameterization trick with continuous relaxation.

gradient information with the reparameterization trick, transporting the signal of $\partial c/\partial z$ from the bias-correction term $c(z)$ through z to θ. Thus, we build two paths from θ to costs via which the gradient signals can be sent back. The gradient estimator w.r.t. θ is written as:

$$\hat{g} := (f(z) - c(z))\frac{\partial}{\partial\theta}\log p(z;\theta) + \frac{\partial c}{\partial z}\frac{\partial}{\partial\theta}z(\epsilon;\theta) \tag{24}$$

Generally, $c(z;w)$ is parameterized by a neural network with weights w that should be learned as well. We usually turn it into an optimization problem to get a variance-minimizing solution, minimizing $\text{Var}(\hat{g}) = \mathbb{E}[\hat{g}^2] - \mathbb{E}[\hat{g}]^2$. Since $\hat{g}$ is unbiased, we minimize $\mathbb{E}[\hat{g}^2]$ instead, which can be further approximated by $\min_w \mathbb{E}[(f(z) - c(z;w))^2]$, indicating that the best $c(z;w)$ should be learned by fitting $f(z)$.

5.7 Control Variates + Reparameterization Trick + Continuous Relaxation

The technique mentioned in the previous subsection, combining control variates with the reparameterization trick, can also be applied to discrete random variables under continuous relaxation [18, 17]. We have already seen the way to reparameterize a discrete distribution under continuous relaxation, with a temperature hyperparameter λ tuned to control the bias. Here, we can derive an unbiased gradient estimator without the need to tune λ. The unbiasedness is guaranteed by subtracting and adding the same function as shown in Figure 16. Note that the one used as a control variate, $c(\tilde{z})$, does not have to rely on the same z as in the bias-correction term $c(z)$, because we keep the total cost unbiased in the expectation level as follow:

$$\mathbb{E}_B[f(B)] = \mathbb{E}_Z\Big[\mathbb{E}_{B|Z}\big[f(B) - \mathbb{E}_{\tilde{Z}|B}[c(\tilde{Z})]\big] + c(Z)\Big] \tag{25}$$

It shows that z is sampled before knowing b, while $\tilde{z}$ is sampled after b is given. In this way, we construct a relaxed $\tilde{z}$ conditioned on b, so that $c(\tilde{z})$ can correlate with $f(b)$ more closely to reduce the variance. Here, $Z \sim p(\cdot;\theta)$ follows a prior distribution while $\tilde{Z} \sim p(\cdot|b;\theta)$ follows a posterior distribution, each of which is reparameterized by using a different transformation. Finally, we open up three paths to transport gradient signals back. The gradient estimator is written as:

$$\hat{g} := (f(b) - c(\tilde{z}))\frac{\partial}{\partial\theta}\log p(b;\theta) - \frac{\partial c}{\partial\tilde{z}}\frac{\partial}{\partial\theta}\tilde{z}(\tilde{\epsilon},b;\theta) + \frac{\partial c}{\partial z}\frac{\partial}{\partial\theta}z(\epsilon;\theta) \tag{26}$$

5.8 Gradient Estimators with Q-functions

We have introduced several advanced approaches for gradient estimation previously. There are two ways to apply these techniques to our learned Q-functions $Q_{w_Z}(z)$ at each stochastic node Z in an SCG.
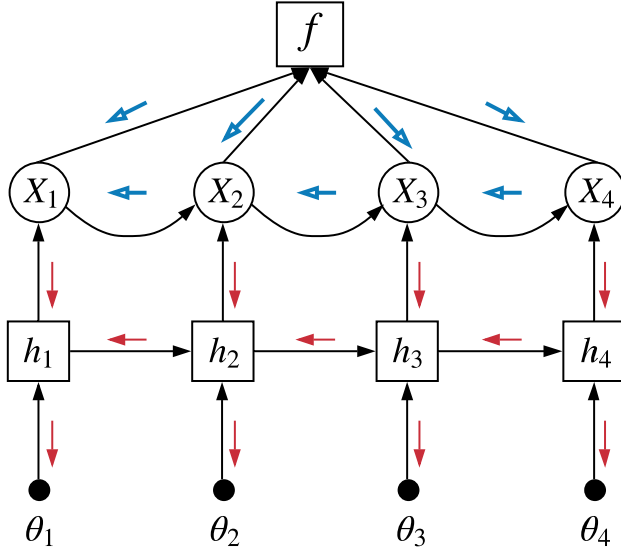
21

Figure 17: An illustration for generalized backpropagation. Blue arrows carry the signals of sample-based expected costs, and red arrows carry the signals of gradients.

(1) We treat $Q_{w_z}(z)$ as a local cost $f(z)$ and apply the previously introduced approaches directly. If z is continuous, we use the reparameterization trick and define a transformation function $z(\epsilon; \theta)$ where ϵ is a noise from a fixed distribution. If z is discrete, we introduce a relaxed random variable z' following the Concrete or the Gumbel-Softmax distribution such that $z = H(z')$, and then apply control variates and the reparameterization trick, with $Q_{w_z}(z')$ as the control variate. We also get a bias-correction term $Q_{w_z}(\tilde{z}')$ with a different $\tilde{z}'$. However, the gradient estimator here is still biased, as $Q_{w_z}(z)$ is an approximation to the true Q-function.

(2) From the previous two subsections, we find that $f(z)$ does not have to be a local cost. In fact, we can use an actual return from the remote cost as f though very stochastic with high variance. Then, we change the role of being a local surrogate cost played by $Q_{w_z}(z)$, and let it act as a control variate to reduce the high variance. Because $Q_{w_z}(z)$ is learned by fitting the expectation of the remote cost, it is an ideal choice for control variates. We can also define a control variate based on $Q_{w_z}(z)$, with a scale factor a and a baseline b, as $c(z) = a \cdot Q_{w_z}(z) + b$, where a and b are acquired by minimizing the variance. With a second term to correct the bias, the unbiased gradient estimator is written as:

$$\hat{g}_z := \left(R - aQ_{w_z}(z) - b\right)\frac{\partial}{\partial \theta}\log p(z|Pa_z; \theta) + a\frac{\partial}{\partial z}Q_{w_z}(z)\frac{\partial}{\partial \theta}z(\epsilon; \theta) \qquad (27)$$

where R represents an actual return. If z is discrete, we apply continuous relaxation the way as (1).

6 The Big Picture of Backpropagation

Looking over the panorama of learning in a SCG, we see that the Backprop-Q framework extends backpropagation to a more general level, propagating learning signals not only across deterministic nodes but also stochastic nodes. The stochastic nodes act like repeaters, sending expected costs back over all random variables. Then, each local parameterized distribution, which is a computation subgraph consisting of many deterministic and differentiable operations, takes over the job of backpropagation and then the standard backpropagation starts. Note that these computation subgraphs can overlap by sharing the common parameters with each other. See Figure 17.

7 Algorithms

Algorithm 1 Basic Framework of Backprop-Q (BPQ)

Input: SCG $(\mathcal{X}, \mathcal{G}_\mathcal{X}, \mathcal{P}, \Theta, \mathcal{F}, \Phi)$, Backprop-Q network $(\mathcal{Q}, \mathcal{G}_\mathcal{Q}, \mathcal{R})$, a set of approximators $\{Q_{w_X} \mid \forall Q_X \in \mathcal{Q}\}$

1: Initialize (Θ, Φ) and all w_X
2: **repeat**
3:     // A forward pass
4:     **for** each $X \in \mathcal{X}$ in a topological order of $\mathcal{G}_\mathcal{X}$ **do**
5:         Sample $x \sim p_x(\cdot|Pa_x; \theta_x)$
6:         Store x and values computed on deterministic nodes in this forward pass
7:     **end for**
8:     Compute and store values on each cost node $f \in \mathcal{F}$
9:     // Backpropagation across stochastic nodes
10:    **for** each $Q_X \in \mathcal{Q}$ (excluding Q_f) in a topological order of $\mathcal{G}_\mathcal{Q}$ **do**
11:        Get sample update target $Q^{\text{tar}} = R_X^{\text{sample}} Q_{w_X}(Sc_X)$ by applying the sample-update version of operator $R_X \in \mathcal{R}$ to approxiamtor Q_{w_X} based on current samples
12:        Take one-step SGD update on w_X by: $w_X \leftarrow w_X + \alpha(Q^{\text{tar}} - Q_{w_X}(Sc_X))\nabla Q_{w_X}(Sc_X)$
13:    **end for**
14:    // Backpropgation across deterministic nodes
15:    **for** each $X \in \mathcal{X}$ **do**
16:        Sum over all $Q_{w_X}(Sc_X)$ to get a total local cost on X
17:        Construct a local differentiable surrogate objective on X using one of the gradient estimation techniques
18:    **end for**
19:    Combine all surrogate objectives with cost functions in $\mathcal{F}$ into one
20:    Run standard backpropagation and take one-step SCG update on (Θ, Φ)
21: **until** (Θ, Φ) converges