Hindsight Network Credit Assignment: Efficient Credit Assignment in Networks of Discrete Stochastic Units

Kenny Young

Department of Computing Science, University of Alberta, Edmonton, Canada
Alberta Machine Intelligence Institute (Amii), Edmonton, Canada
kjyoung@ualberta.ca

Abstract

Training neural networks with discrete stochastic variables presents a unique challenge. Backpropagation is not directly applicable, nor are the reparameterization tricks used in networks with continuous stochastic variables. To address this challenge, we present Hindsight Network Credit Assignment (HNCA), a novel gradient estimation algorithm for networks of discrete stochastic units. HNCA works by assigning credit to each unit based on the degree to which its output influences its immediate children in the network. We prove that HNCA produces unbiased gradient estimates with reduced variance compared to the REINFORCE estimator, while the computational cost is similar to that of backpropagation. We first apply HNCA in a contextual bandit setting to optimize a reward function that is unknown to the agent. In this setting, we empirically demonstrate that HNCA significantly outperforms REINFORCE, indicating that the variance reduction implied by our theoretical analysis is significant and impactful. We then show how HNCA can be extended to optimize a more general function of the outputs of a network of stochastic units, where the function is known to the agent. We apply this extended version of HNCA to train a discrete variational auto-encoder and empirically show it compares favourably to other strong methods. We believe that the ideas underlying HNCA can help stimulate new ways of thinking about efficient credit assignment in stochastic compute graphs.

Introduction

Using discrete stochastic units within neural networks is appealing for a number of reasons, including representing multimodal distributions, modeling discrete choices, providing regularization and facilitating exploration. However, training such units efficiently and accurately presents challenges, as backpropagation is not directly applicable, nor are the reparameterization tricks (Kingma and Welling 2014; Rezende, Mohamed, and Wierstra 2014) that are typically used with continuous stochastic units. Despite these challenges, discrete stochastic units have played an important role in recent empirical successes in both text-to-image generation (Ramesh et al. 2021) and model based reinforcement learning (Hafner et al. 2021). Hence, techniques for efficiently training networks of discrete stochastic units have the potential to be of significant practical interest.

Prior work has proposed a number of techniques for producing either biased, or unbiased estimates of gradients for discrete stochastic units. Bengio, Léonard, and Courville (2013) propose an unbiased REINFORCE (Williams 1992) style estimator, as well as a biased but low variance estimator which replaces a random variable with its expectation during backpropagation. Tang and Salakhutdinov (2013) propose an EM procedure which maximizes a variational lower bound on the loss. Mnih and Gregor (2014) propose several techniques to reduce the variance of a REINFORCE style estimator, including subtracting a learned baseline and normalizing by a moving average standard deviation. Maddison, Mnih, and Teh (2017) and Jang, Gu, and Poole (2017) each propose a biased estimator based on a continuous relaxation of discrete outputs. Tucker et al. (2017) use such a continuous relaxation to derive a control variate for a REINFORCE style estimator, resulting in a variance reduced *unbiased* gradient estimator. Grathwohl et al. (2018) and Gu et al. (2018) also explore the use of control variates with discrete random variables. Yin and Zhou (2019) provide a variance reduced unbiased estimator, called ARM, based on a particular reparameterization and antithetic sampling. Dong, Mnih, and Tucker (2020) further reduce the variance of ARM by marginalizing over the reparameterization step.

We introduce an unbiased, and computationally efficient estimator for the gradients of stochastic units which provably reduces gradient estimate variance compared to REINFORCE. Our estimator works by assigning credit to each unit based on how much it impacts the outputs of its immediate children. Our approach is inspired by Hindsight Credit Assignment (HCA; Harutyunyan et al. (2019)) for reinforcement learning (RL), hence we call it Hindsight Network Credit Assignment (HNCA).

Aside from HCA, perhaps the most closely related work is the Local Expectation Gradients (LEG) approach of Titsias and Lázaro-Gredilla (2015). In fact, the gradient estimator used in HNCA can be seen as an instance of the LEG estimator. However, the generic expression for the LEG estimator makes it unclear when and how it can be efficiently computed. This has led to suggestions in the literature that LEG tends to be too computationally expensive to be practi-

Code to reproduce the experiments is available at: https://github.com/kenjyoung/HNCA_code_supplement.

cal (Tucker et al. 2017; Mnih and Rezende 2016).

The present work extends the work of Titsias and Lázaro-Gredilla (2015) in several ways. First, while LEG may be computational expensive in the general case, for the common case of a network of Bernoulli units, with firing probability parameterized by a linear transformation of their inputs followed by a nonlinear activation, HNCA provides an efficient message passing procedure.[1] In this case, the resulting computational cost is similar to that of Backpropagation. This efficiency allows us to straightforwardly apply HNCA to multi-layer Bernoulli networks, while the analysis and experiments of Titsias and Lázaro-Gredilla (2015) focus on single-layer (fully factorized) stochastic networks. We further demonstrate that a simple baseline subtraction, similar to that employed by Mnih and Gregor (2014), drastically improves performance when applying HNCA to multi-layer networks. While Titsias and Lázaro-Gredilla (2015) focus on the case where the agent has access to the function being optimized, we also present HNCA in a contextual bandit setting where an agent operates online, outputting an action at each time-step and observing a single sampled reward as a result. Interestingly, in the contextual bandit setting, we can still compute local expectations for each hidden unit without the need to resample the reward. Finally, we prove that HNCA provides a variance reduction over REINFORCE.

In taking inspiration from RL to train networks of stochastic units, our work is related to work on CoAgent Networks (Thomas and Barto 2011; Kostas, Nota, and Thomas 2020) that formalizes framing stochastic networks as collectives of interacting RL agents.

In addition to the immediate application to stochastic neural networks, we believe the insights presented in this work can help pave the way for new ways of thinking about efficient credit assignment in stochastic compute graphs, including perhaps the RL setting.

HNCA in a Contextual Bandit Setting

We first formulate HNCA in a contextual bandit setting. In this setting, an agent interacts with an environment in a series of time-steps.[2] At each time-step, the environment provides an i.i.d. random context $X \in \mathcal{X}$ (for example the pixels of an image). The agent then selects an action from a discrete set $A \in \mathcal{A}$ (for example a guess of what class the image belongs to). Finally, the environment responds to with a reward $R = R(X, A)$, where $R : \mathcal{X}, \mathcal{A} \mapsto \mathbb{R}$ is an unknown reward function (for example a reward of 1 for guessing the correct class and 0 otherwise). The agent's goal is to select actions which result in as much reward as possible.

In our case, the agent consists of a network of stochastic computational units. Let Φ be a random variable corresponding to the output of a particular unit. For each unit, Φ is drawn from a parameterized *policy* $\pi_\Phi(\phi|b) \doteq \mathbb{P}(\Phi = \phi|\,\mathrm{pa}(\Phi) = b)$ conditioned on $\mathrm{pa}(\Phi) = b$, its parents in the network.[3] Each unit's policy is differentiably parameterized by a unique set of parameters $\theta_\Phi \in \mathbb{R}^d$. A unit's parents $\mathrm{pa}(\Phi)$ may include the output of other units, as well as the context X. We focus on the case where Φ takes values from a discrete set. We will use $\mathrm{ch}(\Phi)$ to refer to the children of Φ, that is, the set of outputs of all units for which Φ is an input.[4] We assume the network has a single output unit, which selects the action A sent to the environment.

The goal is to tune the network parameters to increase $\mathbb{E}[R]$. Towards this, we will construct an unbiased estimator of the gradient $\frac{\partial\,\mathbb{E}[R]}{\partial\theta_\Phi}$ for the parameters of each unit, and update the parameters according to the estimator.

Directly computing the gradient of the output probability with respect to the parameters for a given input, as we might do with backpropagation for a deterministic network, is generally intractable for discrete stochastic networks. Instead, we can define a local REINFORCE estimator, $\hat{G}^{\mathrm{RE}}_\Phi \doteq \frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial\theta_\Phi} R$. It is well known that $\hat{G}^{\mathrm{RE}}_\Phi$ is an unbiased estimator of $\frac{\partial\,\mathbb{E}[R]}{\partial\theta_\Phi}$ (see Appendix A for a proof). However, $\hat{G}^{\mathrm{RE}}_\Phi$ tends to have high variance.

HNCA Gradient Estimator

HNCA exploits the causal structure of the network to assign credit to each unit's output based on how it impacts the output of its immediate children. Assume Φ is a nonoutput unit and define $\mathrm{mb}(\Phi) \doteq \{\mathrm{ch}(\Phi), \mathrm{pa}(\Phi), \mathrm{pa}(\mathrm{ch}(\Phi)) \setminus \Phi\}$ as a notational shorthand. Note that $\mathrm{mb}(\Phi)$ is a Markov blanket (Pearl 1988) for Φ, meaning that conditioned on $\mathrm{mb}(\Phi)$, Φ is independent of all other variables in the network as well as the reward R. Beginning from the expression for $\hat{G}^{\mathrm{RE}}_\Phi$, we can rewrite $\frac{\partial\,\mathbb{E}[R]}{\partial\theta_\Phi}$ as follows:

$$\begin{aligned}
\frac{\partial\,\mathbb{E}[R]}{\partial\theta_\Phi} &\overset{(a)}{=} \mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial\theta_\Phi} R\right] \\
&\overset{(b)}{=} \mathbb{E}\left[\mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial\theta_\Phi} R \middle| \mathrm{mb}(\Phi), R\right]\right] \\
&\overset{(c)}{=} \mathbb{E}\left[\mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial\theta_\Phi} \middle| \mathrm{mb}(\Phi)\right] R\right] \\
&\overset{(d)}{=} \mathbb{E}\left[\sum_\phi \frac{\mathbb{P}(\Phi = \phi|\,\mathrm{mb}(\Phi))}{\pi_\Phi(\phi|\,\mathrm{pa}(\Phi))} \frac{\partial \pi_\Phi(\phi|\,\mathrm{pa}(\Phi))}{\partial\theta_\Phi} R\right],
\end{aligned} \tag{1}$$

where (a) follows from the unbiasedness of $\hat{G}^{\mathrm{RE}}$, (b) applies the law of total expectation, (c) pulls R out of the expectation and then uses the fact that $\mathrm{mb}(\Phi)$ forms a Markov blanket for Φ, thus we can drop the conditioning on R without loosing anything, and (d) expands the inner expectation over Φ and rewrites the log gradient. This idea of taking a local expectation conditioned on a Markov blanket is similar to the LEG estimator proposed by Titsias and Lázaro-Gredilla

[1] A similarly procedure applies to units with softmax activation, though we do not explore this empirically in this work.

[2] We suppress the time-step in notation, for example writing the context as X instead of X_t.

[3] Expectations and probabilities are taken with respect to all random variables in the network, and the context.

[4] We may also apply $\mathrm{ch}(\cdot)$ or $\mathrm{pa}(\cdot)$ to sets, in which case it has the obvious meaning of the union of the elementwise applications.

(2015). However, it is not immediately obvious how to compute this estimator efficiently. Titsias and Lázaro-Gredilla (2015) provide a more explicit expression and empirical results for a fully factorized variational distribution. Here, we will go beyond this case to provide a computationally efficient way to compute the inner expression for more general networks of stochastic units. To begin, we apply Theorem 1 from Chapter 4 of the probabilistic reasoning textbook of Pearl (1988), which implies that

$$\mathbb{P}(\Phi = \phi | \operatorname{mb}(\Phi)) = \rho_\Phi(\phi)\pi_\Phi(\phi | \operatorname{pa}(\Phi)). \tag{2}$$

where $\rho_\Phi(\phi) = \frac{\prod_{C \in \operatorname{ch}(\Phi)} \pi_C(C | \operatorname{pa}(C) \setminus \Phi, \Phi = \phi)}{\sum_{\phi'} \pi_\Phi(\phi' | \operatorname{pa}(\Phi)) \prod_{C \in \operatorname{ch}(\Phi)} \pi_C(C | \operatorname{pa}(C) \setminus \Phi, \Phi = \phi')}$.
Intuitively, $\rho_\Phi(\phi)$ is the relative counterfactual probability of the children of Φ taking the value they did had Φ been fixed to ϕ. See Appendix B for a full proof. Substituting this result into the expression within the expectation in Equation 1, we get that the following is an unbiased estimator of $\frac{\partial \mathbb{E}[R]}{\partial \theta_\Phi}$:

$$\hat{G}_\Phi^{\text{HNCA}} \doteq \sum_\phi \rho_\Phi(\phi) \frac{\partial \pi_\Phi(\phi | \operatorname{pa}(\Phi))}{\partial \theta_\Phi} R, \tag{3}$$

which we call the HNCA estimator. Equation 3 applies only to Φ for which $\operatorname{ch}(\Phi) \neq \emptyset$, which excludes the output unit A. In our bandit experiments, we use the REINFORCE estimator $\hat{G}_\Phi^{\text{RE}}(\phi)$ for the output unit. Later, we will show how to improve upon this if we have access to the reward function.

HNCA assigns credit to a particular output choice ϕ based on the relative counterfactual probability of its children's outputs had ϕ been chosen, independent of the actual value of Φ. Intuitively, this reduces variance, because each potential output choice of a given unit will get credit proportional to the difference it makes further downstream. On the other hand, REINFORCE credits whatever output happens to be selected, whether it makes a difference or not. This intuition is formalized in the following theorem:

Theorem 1. $\mathbb{V}(\hat{G}_\Phi^{HNCA}) \leq \mathbb{V}(\hat{G}_\Phi^{RE})$*, where* $\mathbb{V}(\vec{X})$ *stand for the elementwise variance of random vector* $\vec{X}$*, and the inequality holds elementwise.*

Theorem 1 follows from the law of total variance by the proof available in Appendix C.

Efficient Implementation of HNCA

We implement HNCA as a message-passing procedure. A forward pass propagates information from parents to children to compute the network output. A backward pass passes information from children to parents to compute the HNCA estimator. The computational complexity of this procedure depends on how difficult it is to compute the numerators of $\rho_\Phi(\phi)$. We could naively recompute $\pi_C(C | \operatorname{pa}(C) \setminus \Phi, \Phi = \phi)$ from scratch for each possible ϕ. When C corresponds to a Bernoulli unit, which computes its output probability as a linear function of its inputs followed by sigmoid activation, this would require time $\mathcal{O}(|\operatorname{pa}(C)| N_\Phi)$, where N_Φ is the number of possible values Φ can take (2 if Φ is also Bernoulli). To do this for every parent of every unit in a Bernoulli network would thus require $\mathcal{O}(2 \sum_\Phi |\operatorname{pa}(\Phi)|^2)$. This is much greater than the cost of a forward pass, which takes on the order of the total number of edges in the network, or $\mathcal{O}(\sum_\Phi |\operatorname{pa}(\Phi)|)$. This contrasts with backpropagation where the cost of the backward pass is on the same order as the forward pass, an appealing property, which implies that learning is not a bottleneck.

Luckily, we can improve on this for cases where $\pi_C(C | \operatorname{pa}(C) \setminus \Phi, \Phi = \phi)$ can be computed from $\pi_C(C | \operatorname{pa}(C))$ in less time than computing $\pi_C(C | \operatorname{pa}(C))$ from scratch. This is indeed the case for linear Bernoulli units, for which the policy can be written $\pi_\Phi(\phi | \vec{x}) = \sigma(\vec{\theta} \cdot \vec{x} + b)$ where $\vec{x}$ is the binary vector consisting of all parent outputs, b is a scalar bias, $\vec{\theta}$ is the parameter vector for the unit, and σ is the sigmoid function. Say we wish to compute the counterfactual probability of $\Phi = 1$ given $\vec{x}[i] = 1$, if we already have $\pi_\Phi(1 | \vec{x})$. Regardless of the actual value of

Algorithm 1: HNCA (Bernoulli unit)

1: Receive $\vec{x}$ from parents
2: $l = \vec{\theta} \cdot \vec{x} + b$
3: $p = \sigma(l)$
4: $\phi \sim$ *Bernoulli*(p)
5: Pass ϕ to children
6: Receive $\vec{q}_1, \vec{q}_0, R$ from children
7: $q_1 = \prod_i \vec{q}_1[i]$
8: $q_0 = \prod_i \vec{q}_0[i]$
9: $\bar{q} = pq_1 + (1-p)q_o$
10: $\vec{l}_1 = l + \vec{\theta} \odot (1 - \vec{x})$
11: $\vec{l}_0 = l - \vec{\theta} \odot \vec{x}$
12: $\vec{p}_1 = (1-\phi)(1-\sigma(\vec{l}_1)) + \phi\sigma(\vec{l}_1)$
13: $\vec{p}_0 = (1-\phi)(1-\sigma(\vec{l}_0)) + \phi\sigma(\vec{l}_0)$
14: Pass $\vec{p}_1, \vec{p}_0, R$ to parents
15: $\vec{\theta} = \vec{\theta} + \alpha\sigma'(l)\vec{x}\left(\frac{q_1 - q_0}{\bar{q}}\right) R$
16: $b = b + \alpha\sigma'(l)\left(\frac{q_1 - q_0}{\bar{q}}\right) R$

Algorithm 1: The forward pass in lines 1-5 takes input from the parents, uses it to compute the fire probability p and samples $\phi \in \{0, 1\}$. The backward pass receives two vectors of probabilities $\vec{q}_1$ and $\vec{q}_0$, each with one element for each child. Each element represents $\vec{q}_{0/1}[i] = \mathbb{P}(C_i | \operatorname{pa}(C_i) \setminus \Phi, \Phi = 0/1)$ for a given child $C_i \in \operatorname{ch}(\Phi)$. Lines 7 and 8 take the product of child probabilities to compute $\prod_i \pi_{C_i}(C_i | \operatorname{pa}(C_i) \setminus \Phi, \Phi = 0/1)$. Line 9 computes the associated normalizing factor. Line 10-13 use the logit l to efficiently compute a vector of probabilities $\vec{p}_1$ and $\vec{p}_0$. Each element corresponds to a counterfactual probability of ϕ if a given parent's value was fixed to 1 or 0. Here $\odot$ represents the elementwise product. Line 14 passes information to the unit's children. Lines 15 and 16 finally update the parameter using $\hat{G}_\Phi^{\text{HNCA}}$ with learning-rate hyperparameter α.

$\vec{x}_i$ we can use the following identity:

$$\pi_\Phi(1|\vec{x}\setminus\vec{x}[i], \vec{x}[i]=1) = \sigma(\sigma^{-1}(\pi_\Phi(1|\vec{x})) + \vec{\theta}[i](1-\vec{x}[i])).$$

This requires only constant time, whereas computing $\pi_\Phi(\phi|\vec{x})$ requires time proportional to the length of $\vec{x}$. This simple idea is crucial for implementing HNCA efficiently. In this case, we can compute the numerator terms for every unit in a Bernoulli network in $\mathcal{O}(\sum_\Phi |\operatorname{pa}(\Phi)|)$ time. This is now on the same order as computing a forward pass through the network. Computing $\hat{G}^{\text{HNCA}}_\Phi$ for a given Φ from these numerator terms requires multiplying a scalar by a gradient vector with the same size as θ_Φ. For a Bernoulli unit, θ_Φ has $\mathcal{O}(|\operatorname{pa}(\Phi)|)$ elements, so this operation adds another $\mathcal{O}(\sum_\Phi |\operatorname{pa}(\Phi)|)$, maintaining the same order of complexity.

Algorithm 1 shows an efficient implementation of HNCA for Bernoulli units. Note that, for ease of illustration, the pseudocode is implemented for a single unit and a single training example at a time. In practice, we use a vectorized version which works with vectors of units that constitute a layer, and with minibatches of training data.

In Section , we will apply HNCA to a model consisting of a number of hidden layers of Bernoulli units followed by a softmax output layer. Appendix D provides an implementation and discussion of HNCA for a softmax output unit. Note that the output unit itself uses the REINFORCE estimator in its update, as it has no children, which precludes the use of HNCA. Nonetheless, the output unit still needs to provide information to its parents, which do use HNCA. Using a softmax unit at the output, we can still maintain the property that the time required for the backward pass is on the same order as the time required for the forward pass. If, on the other hand, the entire network consisted of softmax nodes with N choices each, the HNCA backward pass would require a factor of N more computation than the forward pass, we discuss this in Appendix D as well.

Contextual Bandit Experiments

We evaluate HNCA against REINFORCE in terms of gradient variance and performance on a contextual bandit version of MNIST (LeCun, Cortes, and Burges 2010), with the standard train test split. Following Dong, Mnih, and Tucker (2020), input pixels are dynamically binarized, meaning that at each epoch they are randomly fixed to 0 or 1 with probability proportional to their intensity. For each training example, the model outputs a prediction and receives a reward of 1 if correct and 0 otherwise. We use a fully connected, feed-forward network with 1, 2 or 3 hidden layers, each with 200 Bernoulli units, followed by a softmax output layer. We train using ADAM optimizer (Kingma and Ba 2014) with a learning rate fixed to 10^{-4} and batch-size of 50 for 100 epochs. Learning rate and layer size hyperparameters follow Dong, Mnih, and Tucker (2020) for simplicity. We map the output of the Bernoulli units to one or negative one, instead of one or zero, as we found this greatly improved performance in preliminary experiments. We report results for HNCA and REINFORCE, both with and without an exponential moving average baseline subtracted from the reward. We use a discount rate of 0.99 for the moving average.

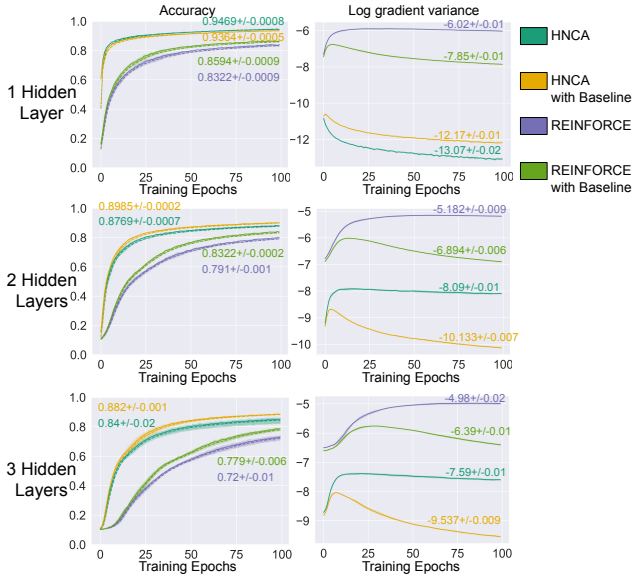

Figure 1: Training stochastic networks on a contextual bandit version of MNIST. Each line represents the average of 5 random seeds with error bars showing 95% confidence interval. Final values (train accuracy for the left plots) at the end of training are written beside each line. The left column shows the online training accuracy (or equivalently the average reward) as a dotted line, and the test accuracy as a solid line (though they essentially overlap). The right column shows the natural logarithm of the mean gradient variance. Mean gradient variance is computed as the mean of the per-parameter empirical variance over examples in a training batch of 50. We find that, for each network depth, HNCA drastically reduces gradient variance, resulting in significantly improved performance on this task.

Figure 1 shows the results, in terms of performance and gradient variance, for gradient estimates generated by HNCA and REINFORCE. We find that HNCA provides drastic improvement in terms of both gradient variance and performance over REINFORCE. Note that performance degrades with number of layers for both estimators, reflecting the increasing challenge of credit assignment. Subtracting a moving average baseline generally improves performance of both algorithms, except for HNCA in the single hidden layer case. The comparison between the two algorithms is qualitatively similar whether or not a baseline is used.

In Appendix E, we demonstrate that HNCA can also be used to efficiently train a stochastic layer as the final hidden layer of an otherwise deterministic network, this could be useful, for example, for learning a binary representation.

Optimizing a Known Function

We introduced HNCA in a setting where the reward function was unknown, and dependent only on the input context and the output of the network as a whole. Here, we extend HNCA to optimize the expectation of a known function f, which may have direct dependence on every unit. We refer

to this extension as f-HNCA. This setting is similar to the setting explored by Titsias and Lázaro-Gredilla (2015), and f-HNCA differs from LEG mainly in its computationally efficient message passing implementation, which in turn facilitates its application to multi-layer stochastic networks.

We assume the function $f = \sum_i f^i$ is factored into a number of *function components* f^i, which we index by i for convenience. This factored structure has two benefits, the first is computational. In particular, it will allow us to compute counterfactual values for each component with respect to changes to its input separately. The second is for variance reduction by realizing that we only need to assign credit to function components that lie downstream of the unit being credited. A similar variance reduction approach is also used by the NVIL algorithm of Mnih and Gregor (2014).

Each function component f^i is a deterministic function of a subset of the outputs of units in the network, as well as possibly depending directly on some parameters. Thus, $f^i = f^i(\widetilde{\mathrm{pa}}(f^i); \theta^i)$, where θ^i is a set of real valued parameters which may overlap with the parameters θ_Φ for some subset of units in the network, and $\widetilde{\mathrm{pa}}(f^i)$ is the set of nodes in the network which act as input to f^i. Formally, f^i without arguments will refer to the random variable corresponding to the output of the associated function. We use the notation $\widetilde{\mathrm{pa}}$, distinct from pa, to make it clear that function components are not considered nodes in the network.

The goal in this setting is to estimate the gradient of $\mathbb{E}[f]$ in order to maximize it by gradient ascent. By linearity of expectation, we can define unbiased estimators for $\frac{\partial \mathbb{E}[f^i]}{\partial \theta_\Phi}$ and sum over i to get an unbiased estimator of the full gradient.

HNCA with a Known Function

We now discuss how to extend the HNCA estimator to construct an estimator of $\frac{\partial \mathbb{E}[f]}{\partial \theta_\Phi}$ for a particular unit Φ and function component in this setting. We begin by considering the gradient for a single function component $\frac{\partial \mathbb{E}[f^i]}{\partial \theta_\Phi}$. First, note that we can break the gradient into indirect and direct dependence on θ_Φ:

$$\frac{\partial \mathbb{E}[f^i]}{\partial \theta_\Phi} = \mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial \theta_\Phi} f^i\right] + \mathbb{E}\left[\frac{\partial f^i}{\partial \theta_\Phi}\right]. \quad (4)$$

The direct gradient $\frac{\partial f^i}{\partial \theta_\Phi}$ is zero unless $\theta_\Phi \in \theta^i$, in which case it can be computed directly given we assume access to f^i. From this point on, we will focus on the left expectation.

The main added complexity in estimating $\mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial \theta_\Phi} f^i\right]$, compared to the contextual bandit case, arises if f^i has a direct functional dependence on Φ. In this case we can no longer assume that f^i is separated from Φ by $\mathrm{mb}(\Phi)$. Luckily, this is straightforward to patch. Let $f^i_\Phi(\phi)$ be the random variable defined by taking the function $f^i(\widetilde{\mathrm{pa}}(f^i); \theta^i)$ and substituting the specific value ϕ instead of the random variable Φ into the arguments while keeping all other $\widetilde{\mathrm{pa}}(f^i)$ equal to the associated random variables. By design, $f^i_\Phi(\phi)$ is independent of Φ given $\mathrm{mb}(\Phi)$, which allows us to define the following unbiased estimator for $\mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial \theta_\Phi} f^i\right]$ (see Appendix F for the full derivation):

$$\hat{G}^{f\text{-HNCA},i}_\Phi(\phi) \doteq \sum_\phi \rho_\Phi(\phi) \frac{\partial \pi_\Phi(\Phi|\,\mathrm{pa}(\Phi))}{\partial \theta_\Phi} f^i_\Phi(\phi), \quad (5)$$

where $\rho_\Phi(\phi)$ is as in Equation 2. As $\rho_\Phi(\phi)$ is defined with respect to $\mathrm{ch}(\Phi)$, this estimator is only applicable if Φ has children (i.e. $\mathrm{ch}(\Phi) \neq \emptyset$). In fact, even if Φ has children, we can ignore them if they have no downstream connection[5] to f^i, as such children cannot influence f^i. Thus if $\mathrm{ch}(\Phi) \cap \widetilde{\mathrm{an}}(f^i) = \emptyset$ we instead define $\hat{G}^{f\text{-HNCA},i}_\Phi(\phi) \doteq \sum_\phi \frac{\partial \pi_\Phi(\Phi|\,\mathrm{pa}(\Phi))}{\partial \theta_\Phi} f^i_\Phi(\phi)$. In Appendix H, we extend Theorem 1 to apply to f-HNCA, showing that using $\hat{G}^{f\text{-HNCA},i}_\Phi$ results in a variance reduced estimator for $\mathbb{E}\left[\frac{\partial \log(\pi_\Phi(\Phi|\,\mathrm{pa}(\Phi)))}{\partial \theta_\Phi} f^i\right]$ compared to REINFORCE. The full f-HNCA estimator is defined by summing up these components and accounting for any direct functional dependence of f on network parameters:

$$\hat{G}^{f\text{-HNCA}}_\Phi \doteq \sum_\phi \frac{\partial \pi_\Phi(\phi|\,\mathrm{pa}(\Phi))}{\partial \theta_\Phi} \left(\rho_\Phi(\phi) \sum_{i:\mathrm{ch}(\Phi)\cap\widetilde{\mathrm{an}}(f^i)\neq\emptyset} f^i_\Phi(\phi) \quad + \sum_{i:\mathrm{ch}(\Phi)\cap\widetilde{\mathrm{an}}(f^i)=\emptyset} f^i_\Phi(\phi) \right) + \sum_i \frac{\partial f^i}{\partial \theta_\Phi}. \quad (6)$$

If $\Phi \notin \widetilde{\mathrm{an}}(f^i)$ then $\frac{\partial \mathbb{E}[f^i]}{\partial \theta_\Phi} = \mathbb{E}\left[\frac{\partial f^i}{\partial \theta_\Phi}\right]$ as Φ cannot influence something with no downstream connection. Hence, in the two leftmost sums over i in Equation 6, we implicitly only sum over i such that $\Phi \in \widetilde{\mathrm{an}}(f^i)$.

In addition to the efficiency of computing counterfactual probabilities, for f-HNCA, we have to consider the efficiency of computing counterfactual function components $f^i_\Phi(\phi)$ given f^i. For function components with no direct connection to a unit Φ, this is trivial as $f^i_\Phi(\phi) = f^i$. If f^i is directly connected, then implementing f-HNCA with efficiency similar to HNCA will require that we are able to compute $f^i_\Phi(\phi)$ from f^i in constant time. This is the case if f^i is a linear function followed by some activation. For example, functions of the form $f^i = \log(\sigma(\vec{\theta} \cdot \vec{x} + b))$ which will appear in the ELBO function used in our variational auto-encoder (VAE; Kingma and Welling (2014); Rezende, Mohamed, and Wierstra (2014)) experiments. More algorithmic details can be found in Appendix G.

Variational Auto-encoder Experiment

Here, we demonstrate how the f-HNCA approach described in Section can be applied to the challenging task of training a discrete hierarchical VAE. Consider a VAE consisting of a generative model (decoder) p and an approximate posterior (encoder) q, each of which consist of L discrete stochastic layers. Samples $\vec{X}$ are generated by p as

$$\vec{X} \sim p_0(\vec{X}|\vec{\Phi}_1), \vec{\Phi}_1 \sim p_1(\vec{\Phi}_1|\vec{\Phi}_2), ..., \vec{\Phi}_L \sim p_L(\vec{\Phi}_L),$$

[5] More generally, if only a subset of $\mathrm{ch}(\Phi)$ lies in $\widetilde{\mathrm{an}}(f^i)$ we can replace $\mathrm{ch}(\Phi)$ in $\rho_\Phi(\phi)$ with $\mathrm{ch}^i(\Phi) = (\mathrm{ch}(\Phi) \cap \widetilde{\mathrm{an}}(f^i))$, but, we will not use this in our experiments in this work.

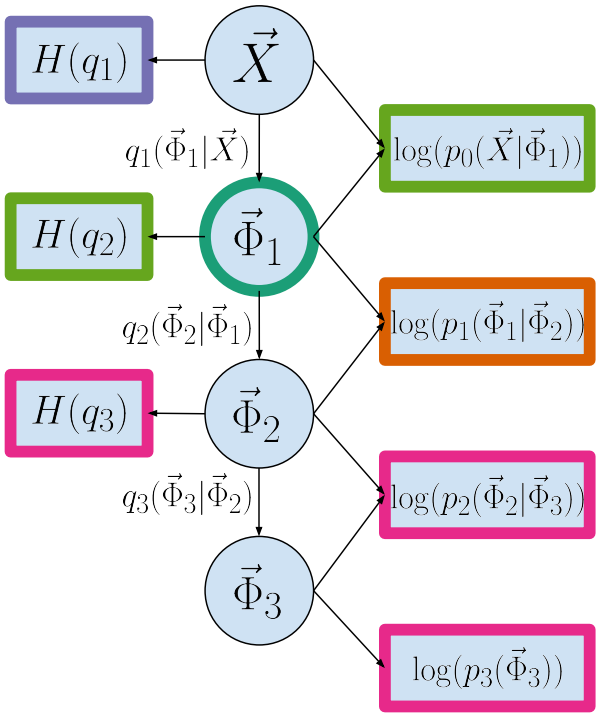

Figure 2: An illustration of the ELBO for a 3 layer discrete hierarchical VAE broken down into function components for f-HNCA. $\vec{X}$ is the input to be encoded, each additional circle is the latent state from a layer of the encoder network. Each rectangle is a set of function components which contribute to the ELBO. The parameters of the encoder are trained to maximize the ELBO by f-HNCA. Consider the f-HNCA estimator for $\vec{\Phi}_1$. The function components $H(q_1)$, marked in purple are upstream of $\vec{\Phi}_1$, however, $H(q_1)$ depends directly on θ_{q_1} and thus $\frac{\partial H(q_1)}{\partial \theta_{q_1}}$ is nonzero, so the entire contribution of $H(q_1)$ to the gradient estimate $\hat{G}_{\Phi}^{f\text{-HNCA}}$ will come from this gradient. The function components marked in green have only direct connection with $\vec{\Phi}_1$, so they will receive credit via $\hat{G}_{\Phi}^{f\text{-HNCA},i}(\phi) \doteq \sum_{\phi} \frac{\partial \pi_{\Phi}(\Phi|\,\text{pa}(\Phi))}{\partial \theta_{\Phi}} f_{\Phi}^{i}(\phi)$. The function components marked in orange have both direct connections and downstream connections mediated by $\vec{\Phi}_2$, so they will receive credit via Equation 5. Finally, the variables marked in pink have only mediated connections to $\vec{\Phi}_1$ through $\vec{\Phi}_2$, so $f_{\Phi}^{i}(\phi) = f^i$, the estimator for these variables essentially reduces to the original HNCA estimator defined in Equation 3.

while q approximates the posterior $\mathbb{P}(\vec{\Phi}_L|X)$ as a distribution which can be sampled as

$$\Phi_L \sim q_L(\vec{\Phi}_L|\vec{\Phi}_{L-1}), \vec{\Phi}_{L-1} \sim q_{L-1}(\vec{\Phi}_{L-1}|\vec{\Phi}_{L-2}), \ldots , \vec{\Phi}_1 \sim q_1(\vec{\Phi}_1|\vec{X}),$$

where, each p_i and q_i represents a vector of Bernoulli distributions, each parameterized as a linear function of their input (except the prior $p_L(\vec{\Phi}_L)$ which takes no input, and is simply a vector of Bernoulli variables with learned means). Call the associated parameters θ_{p_i} and θ_{q_i} We can train such a VAE by maximizing a lower bound on the log-likelihood of the training data, usually referred to as the evidence lower bound (ELBO) which we can write as $\mathbb{E}[f_E]$ where

$$f_E \doteq \log(p_0(\vec{X}|\vec{\Phi}_1)) + \sum_{l=1}^{L-1} \log(p_l(\vec{\Phi}_l|\vec{\Phi}_{l-1})) + \log(p_L(\vec{\Phi}_L)) + H(q_1(\cdot|\vec{X})) + \sum_{l=1}^{L-1} H(q_{l+1}(\cdot|\vec{\Phi}_l)), \quad (7)$$

where H is the entropy of the distribution, and the expectation is taken with respect to the encoder q and random samples $\vec{X}$. Each $\vec{\Phi}_i$ is sampled from the associated encoder q_i. Note that each term in Equation 7 is a sum over elements in the associated output vector, we can view each element as a particular function component f^i. The resulting compute graph is illustrated in Figure 2.

We compare f-HNCA with REINFORCE and several stronger methods for optimizing an ELBO of a VAE trained to generate MNIST digits. We focus on strong, unbiased, variance reduction techniques from the literature that do not require modifying the architecture or introduce significant additional hyperparameters. Since HNCA falls into this category, this allows for straightforward comparison without the additional nuance of architectural and hyperparameter choices. Specifically, we compare HNCA with REINFORCE leave one out (REINFORCE LOO; Kool, van Hoof, and Welling (2019)) and DisARM (Dong, Mnih, and Tucker 2020). Note that in the multi-layer case, both DisARM and REINFORCE LOO require sampling an additional partial forward pass beginning from each layer, which gives them a quadratic scaling in compute cost with the number of layers. By contrast, HNCA requires only a single forward pass and a backward pass of similar complexity.

Initially, we found that f-HNCA outperformed the other tested methods in the single layer discrete VAE case, but fell short in the multi-layer case. However, we found that a simple modification that subtracts a layer specific scalar baseline, similar to that used by Mnih and Gregor (2014), significantly improved the performance of f-HNCA in the multi-layer case. Specifically, for each layer, we maintain a scalar running average of the sum of those components of f with mediated connections (those highlighted in pink and orange in Figure 2) and subtract it from the leftmost sum over i in Equation 6 to produce a centered learning signal.[6] We use a discount rate of 0.99 for the moving average.[7] We refer to this variant as f-HNCA with Baseline. We also tested subtracting a moving average of all downstream function components in REINFORCE to understand how much this change helps on its own. It's not obvious how to apply a running average baseline to the other tested methods, as they already use alternative means to center the learning signal a naive moving average baseline would have expectation zero.

[6] Using such a baseline for components without mediated connections would analytically cancel.

[7] We used the first value we tried, we did not tune it.

As in Section , we use dynamic binarization and train using ADAM optimizer with learning rate 10^{-4} and batch-size 50. Following Dong, Mnih, and Tucker (2020), our decoder and encoder each consist of a fully connected, stochastic feedforward neural network with 1, 2 or 3 layers, each hidden layer has 200 Bernoulli units. We train for 840 epochs, approximately equivalent to the 10^6 updates used by Dong, Mnih, and Tucker (2020). For consistency with prior work, we use Bernoulli units with a zero-one output. For all methods, we train each unit based on downstream function components, as opposed to using the full function f. See Appendix I for more implementation details.

Figure 3, shows the results in terms of ELBO and gradient variance, for gradient estimates generated by f-HNCA and the other methods tested. As in the contextual bandit case, we find that f-HNCA provides drastic improvement over REINFORCE. f-HNCA also provides a significant improvement over all other methods for the single-layer discrete VAE, but underperforms the other strong methods in the multi-layer case. On the other hand, f-HNCA with Baseline significantly improves on the other tested methods in all cases. REINFORCE with baseline outperforms ordinary f-HNCA in the multi-layer cases. Hence, this baseline subtraction is a fairly powerful variance reduction technique for REINFORCE, with strong complementary benefits with f-HNCA. In Appendix J, we additionally report multi-sample test-set ELBOs for the final trained networks, which reflect the same performance ordering as the training set ELBOs. In Appendix K, we perform an ablation experiments on f-HNCA with Baseline and find that the choice of whether to exclude children when $\mathrm{ch}(\Phi) \cap \widetilde{\mathrm{an}}(f^i) = \emptyset$ has a significant performance impact, while the additional impact of excluding upstream function components is fairly minimal.

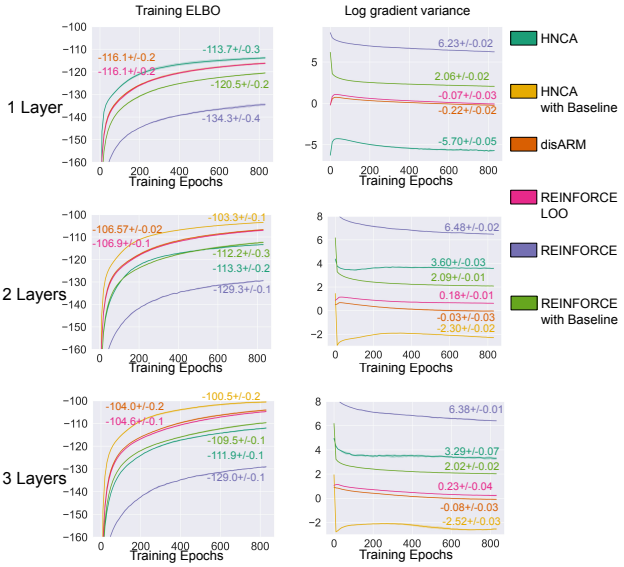

Figure 3: Training discrete VAEs to generate MNIST digits. Each line represents the average of 5 random seeds with error bars showing 95% confidence interval. Final values at the end of training are written near each line in matching color. The left column shows the online training ELBO. The right column shows the natural logarithm of the mean encoder gradient variance. Mean gradient variance is computed as the mean over parameters and batches of the per-parameter empirical variance over examples in a training batch of 50. f-HNCA outperforms all other tested methods in the single-layer case, but underperforms in the multi-layer cases. f-HNCA with Baseline outperforms the other methods in the multi-layer case. f-HNCA with baseline is excluded from the single layer results as there are no mediated connections.

Discussion and Conclusion

We introduced HNCA, an algorithm for gradient estimation in networks of discrete stochastic units. HNCA is inspired by Hindsight Credit Assignment (Harutyunyan et al. 2019), and can be seen as an instance of Local Expectation Gradients, extending the work of Titsias and Lázaro-Gredilla (2015) by providing a computationally efficient message passing algorithm and extension to multi-layer networks of stochastic units. Our computational efficient approach directly addresses concerns in the literature that LEG is inherently computationally expensive (Tucker et al. 2017; Mnih and Rezende 2016). We prove that HNCA is unbiased, and that it reduces variance compared to REINFORCE. Empirically, we show that HNCA outperforms strong methods for training a single-layer Bernoulli VAE, and when subtracting a simple moving average baseline also outperforms the same methods for the case of a multi-layer Hierarchical VAE.

It's worth highlighting that efficient implementation of HNCA is predicated on the ability to efficiently compute counterfactual probabilities or function components when a single input is changed. This is not always possible, for example, if f is the result of a multi-layer deterministic network. An example of this situation is the nonlinear discrete VAE architecture explored by Dong, Mnih, and Tucker (2020) and Yin and Zhou (2019) where the encoder and decoder are nonlinear networks with a single stochastic Bernoulli layer at the outputs. However, as we show in Appendix E, HNCA can be used to train a final Bernoulli hidden layer at the end of a nonlinear network.

In addition to optimizing a known function of the output of a stochastic network, we show that HNCA can be applied to train the hidden layers of a multi-layer discrete network in an online learning setting with unknown reward function. REINFORCE LOO and DisARM, which rely on the ability to evaluate the reward function multiple times for a single training example, cannot.

Future work could explore combining HNCA with other methods for complimentary benefits. One could also explore extending HNCA to propagate credit multiple steps which would presumably allow further variance reduction, but presents challenges as the relationships between more distant nodes in the network becomes increasingly complex.

HNCA provides insight into the challenges of credit assignment in discrete stochastic compute graphs, which has the potential to have an impact on future approaches.

Acknowledgments

The author thanks Rich Sutton, Matt Taylor and Tian Tian for useful conversations, and anonymous reviewers for useful feedback. I also thank the Natural Sciences and Engineering Research Council of Canada and Alberta Innovates for providing funding for this work.

References

Bengio, Y.; Léonard, N.; and Courville, A. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*.

Dong, Z.; Mnih, A.; and Tucker, G. 2020. DisARM: An antithetic gradient estimator for binary latent variables. *Advances in neural information processing systems*, 33.

Grathwohl, W.; Choi, D.; Wu, Y.; Roeder, G.; and Duvenaud, D. 2018. Backpropagation through the void: Optimizing control variates for black-box gradient estimation. *International Conference on Learning Representations*.

Gu, S.; Levine, S.; Sutskever, I.; and Mnih, A. 2018. Muprop: Unbiased backpropagation for stochastic neural networks. *International Conference on Learning Representations*.

Hafner, D.; Lillicrap, T. P.; Norouzi, M.; and Ba, J. 2021. Mastering Atari with Discrete World Models. In *International Conference on Learning Representations*.

Harutyunyan, A.; Dabney, W.; Mesnard, T.; Azar, M. G.; Piot, B.; Heess, N.; van Hasselt, H. P.; Wayne, G.; Singh, S.; Precup, D.; et al. 2019. Hindsight credit assignment. *Advances in neural information processing systems*, 32: 12488–12497.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical reparameterization with gumbel-softmax. *International Conference on Learning Represenations*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kingma, D. P.; and Welling, M. 2014. Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kool, W.; van Hoof, H.; and Welling, M. 2019. Buy 4 REINFORCE Samples, Get a Baseline for Free! In *ICLR Deep Reinforcement Learning Meets Structured Prediction Workshop*.

Kostas, J.; Nota, C.; and Thomas, P. 2020. Asynchronous Coagent Networks. *Proceedings of the 37th International Conference on Machine learning*, 5426–5435.

LeCun, Y.; Cortes, C.; and Burges, C. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The concrete distribution: A continuous relaxation of discrete random variables. *International Conference on Learning Represenations*.

Mnih, A.; and Gregor, K. 2014. Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*, 1791–1799. PMLR.

Mnih, A.; and Rezende, D. J. 2016. Variational Inference for Monte Carlo Objectives. *Proceedings of the 33rd International Conference on Machine Learning*, 2188–2196.

Pearl, J. 1988. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann.

Ramesh, A.; Pavlov, M.; Goh, G.; Gray, S.; Voss, C.; Radford, A.; Chen, M.; and Sutskever, I. 2021. Zero-shot text-to-image generation. *arXiv preprint arXiv:2102.12092*.

Rezende, D. J.; Mohamed, S.; and Wierstra, D. 2014. Stochastic backpropagation and approximate inference in deep generative models. *Proceedings of the 31st International Conference on Machine learning*, 1278–1286.

Tang, C.; and Salakhutdinov, R. R. 2013. Learning stochastic feedforward neural networks. *Advances in Neural Information Processing Systems*, 26: 530–538.

Thomas, P. S.; and Barto, A. G. 2011. Conjugate Markov Decision Processes. *Proceedings of the 28th International Conference on Machine learning*, 137–144.

Titsias, M. K.; and Lázaro-Gredilla, M. 2015. Local expectation gradients for black box variational inference. *Advances in Neural Information Processing Systems*, 28: 2638–2646.

Tucker, G.; Mnih, A.; Maddison, C. J.; Lawson, J.; and Sohl-Dickstein, J. 2017. Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. *Advances in Neural Information Processing Systems*, 30: 2627–2636.

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

Yin, M.; and Zhou, M. 2019. ARM: Augment-REINFORCE-merge gradient for stochastic binary networks. *International Conference on Learning Representations*.