# The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables

**Chris J. Maddison**[1,2], **Andriy Mnih**[1], **& Yee Whye Teh**[1,2]
DeepMind[1]
University of Oxford[2]
`{cmaddis, y.w.teh}@stats.ox.ac.uk`, `amnih@google.com`

## Abstract

The reparameterization trick enables optimizing large scale stochastic computation graphs via gradient descent. The essence of the trick is to refactor each stochastic node into a differentiable function of its parameters and a random variable with fixed distribution. After refactoring, the gradients of the loss propagated by the chain rule through the graph are low variance unbiased estimators of the gradients of the expected loss. While many continuous random variables have such reparameterizations, discrete random variables lack continuous reparameterizations due to the discontinuous nature of discrete states. In this work we introduce *concrete random variables* – continuous relaxations of discrete random variables. The concrete distribution is a new family of distributions with closed form densities and a simple reparameterization. Whenever a discrete stochastic node of a computation graph can be refactored into a one-hot bit representation that is treated continuously, concrete stochastic nodes can be used with automatic differentiation to produce low-variance biased gradients of objectives (including objectives that depend on the log-probability of latent stochastic nodes) on the corresponding discrete graph. We demonstrate effectiveness of concrete relaxations on density estimation and structured prediction tasks using neural networks.

## 1 Introduction

Software libraries for automatic differentiation (AD) [1, 40] are enjoying broad use spurred on by the success of neural networks on some of the most challenging problems of machine learning. The dominant mode of research in these environments is to define a forward parametric computation, in the form of a directed acyclic graph, that computes the desired objective. If the components of the graph are continuous, then a backwards computation for the gradient of the objective can be derived automatically with the chain rule. The ease of use and unreasonable effectiveness of gradient descent has lead to an explosion in the diversity of architectures and objective functions. Thus, expanding the range of useful continuous operations can have an outsized impact on the development of new models. For example, a topic of recent attention has been the optimization of stochastic computation graphs from samples of their states. Here, the observation that AD "just works" when stochastic nodes[1] can be reparameterized into deterministic functions of their parameters and a fixed noise distribution [23, 35], has liberated researchers in the development of large complex stochastic architectures [e.g. 16].

[1]For our purposes a stochastic node of a computation graph is just a random variable whose distribution depends in some deterministic way on the values of the parent nodes.

Computing with discrete stochastic nodes still poses a significant challenge for AD libraries. Deterministic discreteness can be relaxed and approximated reasonably well with sigmoidal functions or the softmax [15, 13], but, if a distribution over discrete states is needed, there is no clear solution. There are well known unbiased estimators for the gradients of the parameters of a discrete stochastic node from samples. While these can be made to work with AD, they involve special casing and defining surrogate objectives [39], and even then they can have high variance. Still, reasoning about discrete computation comes naturally to humans, and so, despite the difficulty associated, many modern architectures incorporate discrete stochasticity [44, 25].

This work is inspired by the observation that many architectures treat discrete nodes continuously, and gradients rich with counterfactual information are available for each of their possible states. We introduce a *con*tinuous relaxation of dis*crete* random variables, concrete for short, which allow gradients to flow through their states. The *concrete distribution* is a new parametric family of continuous distributions on the simplex with closed form densities. Sampling from the concrete distribution is as simple as taking the softmax of logits perturbed by fixed additive noise. This reparameterization means that concrete stochastic nodes are quick to implement in a way that "just works" with AD. Crucially, every discrete random variable corresponds to the zero temperature limit of a concrete one. In this view optimizing an architecture with discrete stochastic nodes can be accomplished by gradient descent on the samples of the corresponding concrete relaxation. When the objective function on the graph depends, as in variational inference, on the log-probability of certain nodes, the concrete density is used in place of the discrete mass. The graph with discrete nodes is evaluated.

The paper is organized as follows. We provide a background on stochastic computation graphs and their optimization in Section 2. Section 3 reviews a reparameterization for discrete random variables, introduces the concrete distribution, and discusses its application as a relaxation. Section 4 reviews related work. In Section 5 we present results on a density estimation task and a structured prediction task on the MNIST and Omniglot datasets. In the Appendix we provide more details on the practical implementation and use of concrete random variables. When comparing the effectiveness of gradients obtained via concrete relaxations to a state-of-the-art-method [VIMCO, 29], we find that they are competitive — occasionally outperforming and occasionally underperforming — all the while being implemented in an AD library without special casing.

## 2 Background

### 2.1 Optimizing Stochastic Computation Graphs

Stochastic computation graphs (SCGs) provide a formalism for specifying, potentially stochastic, input-output mappings with learnable parameters using directed acyclic graphs (see [39] for a review). The state of each non-input node in such a graph is obtained from the states of its parent nodes by either evaluating a deterministic function or sampling from a conditional distribution. Many training objectives in supervised, unsupervised, and reinforcement learning can be be expressed in terms of SCGs.

To optimize an objective represented as a SCG, we need estimates of its parameter gradients. We will concentrate on graphs with some stochastic nodes (backpropagation covers the rest). For simplicity, we restrict our attention to graphs with a single stochastic node $X$. We can interpret the forward pass in the graph as first sampling $X$ from the conditional distribution $p_\phi(x)$ of the stochastic node given its parents, then evaluating a deterministic function $f_\theta(x)$ at $X$. We can think of $f_\theta(X)$ as a noisy objective, and we are interested in optimizing its expected value $L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)}[f_\theta(X)]$ w.r.t. parameters $\theta, \phi$.

In general, both the objective and its gradients are intractable. We will side-step this issue by estimating them with samples from $p_\phi(x)$. The gradient w.r.t. to the parameters $\theta$ has the form

$$\nabla_\theta L(\theta, \phi) = \nabla_\theta \mathbb{E}_{X \sim p_\phi(x)}[f_\theta(X)] = \mathbb{E}_{X \sim p_\phi(x)}[\nabla_\theta f_\theta(X)] \tag{1}$$

and can be easily estimated using Monte Carlo sampling:

$$\nabla_\theta L(\theta, \phi) \simeq \frac{1}{S} \sum\nolimits_{s=1}^{S} \nabla_\theta f_\theta(X^s), \tag{2}$$

where $X^s \sim p_\phi(x)$ i.i.d. The more challenging task is to compute the gradient w.r.t. the parameters $\phi$ of $p_\phi(x)$. The expression obtained by differentiating the expected objective,

$$\nabla_\phi L(\theta, \phi) = \nabla_\phi \int p_\phi(x) f_\theta(x)\,\mathrm{d}x = \int f_\theta(x) \nabla_\phi p_\phi(x)\,\mathrm{d}x, \tag{3}$$

does not have the form of an expectation w.r.t. $x$ and thus does not directly lead to a Monte Carlo gradient estimator. However, there are two ways of getting around this difficulty which lead to the two classes of estimators we will now discuss.

### 2.2 Score Function Estimators

The *score function estimator* [SFE, 10], also known as the REINFORCE [43] or likelihood-ratio estimator [12], is based on the identity $\nabla_\phi p_\phi(x) = p_\phi(x) \nabla_\phi \log p_\phi(x)$, which allows the gradient in Eq. 3 to be written as an expectation:

$$\nabla_\phi L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)} \left[ f_\theta(X) \nabla_\phi \log p_\phi(X) \right]. \tag{4}$$

Estimating this expectation using naive Monte Carlo gives the estimator

$$\nabla_\phi L(\theta, \phi) \simeq \frac{1}{S} \sum_{s=1}^{S} f_\theta(X^s) \nabla_\phi \log p_\phi(X^s), \tag{5}$$

where $X^s \sim p_\phi(x)$ i.i.d. This is a very general estimator that is applicable whenever $\log p_\phi(x)$ is differentiable w.r.t. $\phi$. As it does not require $f_\theta(x)$ to be differentiable or even continuous as a function of $x$, the SFE can be used with both discrete and continuous random variables.

Though the basic version of the estimator can suffer from high variance, various variance reduction techniques can be used to make the estimator much more effective [14]. Baselines are the most important and widely used of these techniques [43].

### 2.3 Reparameterization Trick

In many cases we can sample from $p_\phi(x)$ by first sampling $Z$ from some fixed distribution $q(z)$ and then transforming the sample using some function $g_\phi(z)$. For example, a sample from $\mathrm{Normal}(\mu, \sigma^2)$ can be obtained by sampling $Z$ from the standard form of the distribution $\mathrm{Normal}(0, 1)$ and then transforming it using $g_{\mu,\sigma}(Z) = \mu + \sigma Z$. This two-stage reformulation of the sampling process, called the *reparameterization trick*, allows us to transfer the dependence on $\phi$ from $p$ into $f$ by writing $f_\theta(x) = f_\theta(g_\phi(z))$ for $x = g_\phi(z)$, making it possible to reduce the problem of estimating the gradient w.r.t. parameters of a distribution to the simpler problem of estimating the gradient w.r.t. parameters of a deterministic function.

Having reparameterized $p_\phi(x)$, we can now express the objective as an expectation w.r.t. $q(z)$:

$$L(\theta, \phi) = \mathbb{E}_{X \sim p_\phi(x)}[f_\theta(X)] = \mathbb{E}_{Z \sim q(z)}[f_\theta(g_\phi(Z))]. \tag{6}$$

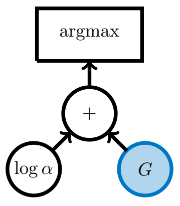
As $q(z)$ does not depend on $\phi$, we can estimate the gradient w.r.t. $\phi$ in exactly the same way we estimated the gradient w.r.t. $\theta$ in Eq. 1. Assuming differentiability of $f_\theta(x)$ w.r.t. $x$ and of $g_\phi(z)$ w.r.t. $\phi$ and using the chain rule gives

$$\nabla_\phi L(\theta, \phi) = \mathbb{E}_{Z \sim q(z)}[\nabla_\phi f_\theta(g_\phi(Z))] = \mathbb{E}_{Z \sim q(z)} \left[ f'_\theta(g_\phi(Z)) \nabla_\phi g_\phi(Z) \right]. \tag{7}$$

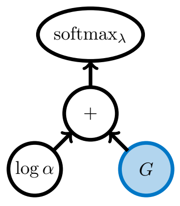
The reparameterization trick can be applied to many continuous random variables [24] and is usually the estimator of choice when it is applicable. It is in large part responsible for the wide adoption of variational autoencoders and related models. Unfortunately, it cannot be applied to discrete latent variables or in cases for which $f$ is not differentiable.

### 2.4 Application: Variational Training of Latent Variable Models

We will now see how the task of training latent variable models can be formulated in the SCG framework. Such models assume that each observation $x$ is obtained by first sampling a vector of latent variables $Z$ from the prior $p_\theta(z)$ before sampling the observation itself from $p_\theta(x \mid z)$. Thus the probability of observation $x$ is $p_\theta(x) = \sum_z p_\theta(z) p_\theta(x \mid z)$. Maximum likelihood training of such models



(a) $\mathrm{Discrete}(\alpha)$



(b) $\mathrm{Concrete}(\alpha, \lambda)$

Figure 1: Visualization of sampling graphs for a discrete random variable $D \sim \mathrm{Discrete}(\alpha)$ and a concrete random variable $X \sim \mathrm{Concrete}(\alpha, \lambda)$. White operations are deterministic, blue are stochastic, circles are continuous, squares discrete. For a single forward pass of the computation every blue node is sampled once; each $G_k$ is sampled via $-\log(-\log U_k)$ where $U_k \sim \mathrm{Uniform}(0, 1)$ i.i.d.; $\mathrm{softmax}_\lambda(x) = \exp(x/\lambda)/\sum_i \exp(x_i/\lambda)$ componentwise for $\lambda \in (0, \infty)$.

is infeasible, because the log-likelihood (LL) objective $L(\theta) = \log p_\theta(x) = \log \mathbb{E}_{Z\sim p_\theta(z)}[p_\theta(x \mid Z)]$ is typically intractable and does not fit into the above framework due to the expectation being inside the $\log$. The multi-sample variational objective, however,

$$\mathcal{L}_m(\theta, \phi) = \mathop{\mathbb{E}}_{Z^i \sim q_\phi(z|x)} \left[ \log \left( \frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(Z^i,\, x)}{q_\phi(Z^i \mid x)} \right) \right]. \tag{8}$$

provides a convenient alternative which has precisely the form we considered in Section 2.1. This approach relies on introducing an auxiliary distribution $q_\phi(z \mid x)$ with its own parameters, which serves as approximation to the intractable posterior $p_\theta(z \mid x)$. The model is trained by jointly maximizing the objective w.r.t. to the parameters of $p$ and $q$. The number of samples used inside the objective $m$ allows trading off the computational cost against the tightness of the bound. For $m = 1$, $\mathcal{L}_m(\theta, \phi)$ becomes is the widely used evidence lower bound [ELBO, 20] on $\log p_\theta(x)$, while for $m > 1$, it is known as the importance weighted bound [6].

The reparameterization trick, introduced in the context of variational inference independently by [24], [35], and [41], is the method of choice for training variational autoencoders and related models with continuous latent variables. For models with discrete latent variables, the discontinuous nature of which makes reparameterization not useful, a number of score function estimators have been developed [31, 17, 33, 28, 42, 18], which differ primarily in the variance reduction techniques used. Recently, new hybrid estimators have also been developed for continuous latent variables which are not directly reparameterizable, by combining partial reparameterizations with score function estimators [36, 30].

## 3 The Concrete Distribution

### 3.1 Discrete Random Variables and the Gumbel-Max Trick

To motivate the construction of concrete random variables, we review a method for sampling from discrete distributions called the Gumbel-Max trick [19, 27, 26]. We restrict ourselves to a representation of discrete states as vectors $d \in \{0, 1\}^n$ of bits that are one-hot, or $\sum_{k=1}^{n} d_k = 1$. In the context of a computation graph, this is a flexible representation; to achieve an integral representation take the inner product of $d$ with $(1, \dots, n)$, and to achieve a point mass representation in $\mathbb{R}^m$ take $Wd$ where $W \in \mathbb{R}^{m\times n}$.

Consider an unnormalized parameterization $(\alpha_1, \dots, \alpha_n)$ where $\alpha_k \in (0, \infty)$ of a discrete distribution $D \sim \mathrm{Discrete}(\alpha)$ — we can assume that states with 0 probability are excluded. The Gumbel-Max trick proceeds as follows: sample $U_k \sim \mathrm{Uniform}(0, 1)$ i.i.d. for each $k$, find $k$ that maximizes $\{\log \alpha_k - \log(-\log U_k)\}$, set $D_k = 1$ and the remaining $D_i = 0$ for $i \neq k$. Then

$$\mathbb{P}(D_k = 1) = \frac{\alpha_k}{\sum_{i=1}^{n} \alpha_i}. \tag{9}$$

In other words, the sampling of a discrete random variable can be refactored into a deterministic function — componentwise addition followed by $\mathrm{argmax}$ — of the parameters $\log \alpha_k$ and fixed distribution $-\log(-\log U_k)$. See Figure 1a for a visualization.

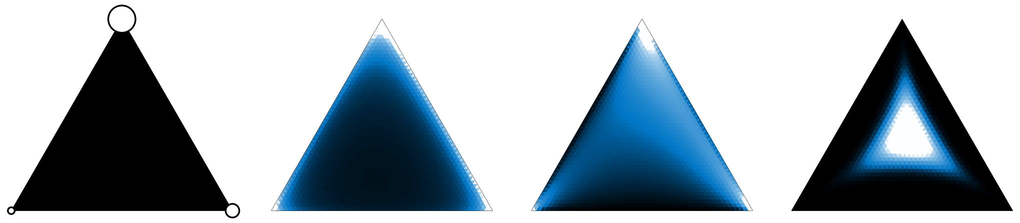

(a) $\lambda = 0$ (b) $\lambda = 1/2$ (c) $\lambda = 1$ (d) $\lambda = 2$

Figure 2: A discrete distribution with unnormalized probabilities $(\alpha_1, \alpha_2, \alpha_3) = (2, 0.5, 1)$ and three corresponding concrete densities at increasing temperatures $\lambda$. Each triangle represents the set of points $(y_1, y_2, y_3)$ in the simplex $\Delta^2 = \{(y_1, y_2, y_3) \mid y_k \in (0, 1), y_1 + y_2 + y_3 = 1\}$. For $\lambda = 0$ the size of white circles represents the mass assigned to each vertex of the simplex under the discrete distribution. For $\lambda \in \{2, 1, 0.5\}$ the intensity of the shading represents the value of $p_{\alpha,\lambda}(y)$.

The apparently arbitrary choice of noise gives the trick its name, as $-\log(-\log U)$ has a Gumbel distribution. This distribution features in extreme value theory [19] where it plays a central role similar to the Normal distribution: the Gumbel distribution is stable under $\max$ operations, and for some distributions, the order statistics (suitably normalized) of i.i.d. draws approach the Gumbel in distribution. The Gumbel can also be recognized as a $-\log$-transformed exponential random variable. So, the correctness of (9) also reduces to a well known result regarding the $\operatorname{argmin}$ of exponential random variables; see [26] for a proof and general treatment of this trick.

## 3.2 Concrete Random Variables

The derivative of the $\operatorname{argmax}$ is 0 everywhere except at the boundary of state changes, where it is undefined. For this reason the Gumbel-Max trick is not a suitable reparameterization for use in SCGs with AD. Here we introduce the concrete distribution motivated by considering a graph, which is the same as Figure 1a up to a continuous relaxation of the $\operatorname{argmax}$ computation, see Figure 1b. This will ultimately allow the optimization of parameters $\alpha_k$ via gradients.

The $\operatorname{argmax}$ computation returns states on the vertices of the simplex $\Delta^{n-1} = \{x \in \mathbb{R}^n \mid x_k \in [0, 1], \sum_{k=1}^n x_k = 1\}$. The idea behind concrete random variables is to relax the state of a discrete variable from the vertices into the interior where it is a random probability vector—a vector of numbers between 0 and 1 that sum to 1. To sample a concrete random variable $X \in \Delta^{n-1}$ at temperature $\lambda \in (0, \infty)$ with parameters $\alpha_k \in (0, \infty)$, sample $G_k \sim \operatorname{Gumbel}$ i.i.d. and set

$$X_k = \frac{\exp((\log \alpha_k + G_k)/\lambda)}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\lambda)}. \tag{10}$$

The softmax computation of (10) smoothly approaches the discrete $\operatorname{argmax}$ computation as $\lambda \to 0$ while preserving the relative order of the Gumbels $\log \alpha_k + G_k$. So, imagine making a series of forward passes on the graphs of Figure 1. Both graphs return a stochastic value for each forward pass, but for smaller temperatures the outputs of Figure 1b become more discrete and eventually indistinguishable from a typical forward pass of Figure 1a.

The distribution of $X$ sampled via (10) has a closed form density on the simplex. Because there may be other ways to sample a concrete random variable, we take the density to be its definition.

**Definition 1** (Concrete Random Variables)**.** Let $\alpha \in (0, \infty)^n$ and $\lambda \in (0, \infty)$. $X \in \Delta^{n-1}$ has a concrete distribution $X \sim \operatorname{Concrete}(\alpha, \lambda)$ with location $\alpha$ and temperature $\lambda$, if its density is:

$$p_{\alpha,\lambda}(x) = (n-1)!\, \lambda^{n-1} \prod_{k=1}^n \left( \frac{\alpha_k x_k^{-\lambda-1}}{\sum_{i=1}^n \alpha_i x_i^{-\lambda}} \right). \tag{11}$$

Proposition 1 lists a few properties of the concrete distribution. 1 is confirmation that our definition corresponds to the sampling routine (10). 2 and 3 are discretization properties. Finally, 4 is a convexity result on the density.

**Proposition 1** (Some Properties of Concrete Random Variables)**.** *Let $X \sim \operatorname{Concrete}(\alpha, \lambda)$ with location parameters $\alpha \in (0, \infty)^n$ and temperature $\lambda \in (0, \infty)$, then*

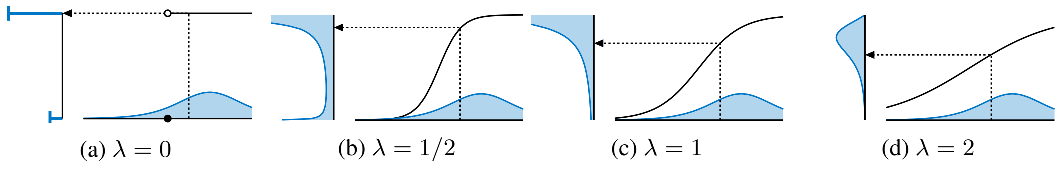


Figure 3: A visualization of a Gumbel-Max reparameterization for Bernoulli random variables and some corresponding concrete relaxations. (a) shows the discrete trick, which works by passing a noisy logit through the unit step function. (b), (c), (d) show concrete relaxations; the horizontal blue densities show the density of the noise distribution $L + \log \alpha$ and the vertical densities show the corresponding concrete density over the interval $(0, 1)$ of the random variable $\sigma((L + \log \alpha)/\lambda)$ for varying temperatures $\lambda$.

1. *(Reparameterization) If* $G_k \sim \text{Gumbel}$ *i.i.d., then* $X_k \overset{d}{=} \frac{\exp((\log \alpha_k + G_k)/\lambda)}{\sum_{i=1}^{n} \exp((\log \alpha_i + G_i)/\lambda)}$,
2. *(Rounding)* $\mathbb{P}\left(X_k > X_i \text{ for } i \neq k\right) = \alpha_k / (\sum_{i=1}^{n} \alpha_i)$,
3. *(Zero temperature)* $\mathbb{P}\left(\lim_{\lambda \to 0} X_k = 1\right) = \alpha_k / (\sum_{i=1}^{n} \alpha_i)$,
4. *(Convex eventually) If* $\lambda \leq (n-1)^{-1}$*, then* $p_{\alpha,\lambda}(x)$ *is log-convex in* $x$.

We prove these results in the Appendix. There is an obvious question whether the Gumbel noise is a necessary feature of this idea. In the binary case, see Figure 3, it can be generalized to accommodate any additive noise distribution with infinite support (we cover this in the Appendix). In contrast, for the general $n$-ary case the Gumbel is a crucial 1 and the Gumbel-Max trick cannot be generalized for other additive noise distributions [45]. In the Appendix we include a cheat sheet for all of the random variables we include in this work along with details on their implementation.

### 3.3 Concrete Random Variables as Relaxations

Concrete random variables may have some intrinsic value, but we investigate them simply as surrogates for optimizing a SCG with discrete nodes. When it is computationally feasible to integrate over the discreteness, that will always be a better choice. Otherwise the basic idea is to optimize a graph with every discrete stochastic node replaced by a concrete stochastic node at some fixed temperature (or with an annealing schedule). Because the graphs are identical up to $\text{softmax}$ / $\text{argmax}$ computations, the parameters of the relaxed graph and discrete graph are the same. The success of concrete relaxations will depend heavily on the choice of temperature during training. It is important that the relaxed nodes are not able to represent a precise real valued mode in the interior of the simplex as in Figure 2d. If this is the case, it is possible for the relaxed random variable to communicate much more than $\log_2(n)$ bits of information about its $\alpha$ parameters. This might lead the relaxation to prefer the interior of the simplex to the vertices, and as a result there will be a large integrality gap in the overall performance of the discrete graph. Therefore 4 of Proposition 1 is a conservative guideline for generic $n$-ary concrete relaxations; at temperatures lower than $(n-1)^{-1}$ we are guaranteed not to have any modes in the interior for any $\alpha \in (0, \infty)^n$. We discuss the subtleties of choosing the temperature in more detail in the Appendix. Ultimately the best choice of $\lambda$ and the performance of the relaxation for any specific $n$ will be an empirical question.

When an objective depends on the log-probability of discrete variables in the SCG, as the variational lowerbound does, we must consider how to modify it in the context of a relaxation. To preserve the property that the variational objective bounds the log-probability of the observed data, the log-probability terms for the latent variables must match their sampling distribution. Thus, in addition to relaxing the sampling pass of a SCG the log-probability terms are also "relaxed" to represent the true distribution of the relaxed node. One could treat the Gumbels $G_k$ as the stochastic node, and the softmax as downstream computation. However, this is a looser bound than the bound depending on log-probability of the concrete states in the simplex, because the softmax is many-to-one. Thus, the concrete-discrete pairing satisfies this valuable property: the discretization of any concrete distribution has a closed form mass function, and the relaxation of any discrete distribution into a concrete distribution has a closed form density. It is generally easy to go from a continuous process to a discrete one by quantizing and backwards by relaxing, but maintaining analytic tractability both ways is not always possible. For example, there is no closed form for the mass function of the multinomial probit model — the Gumbel-Max trick but with Gaussians replacing Gumbels.

## 4 Related Work

Perhaps the most common distribution over the simplex is the Dirichlet with density $p_\alpha(x) \propto \prod_{k=1}^{n} x_k^{\alpha_k - 1}$ on $x \in \Delta^{n-1}$. The Dirichlet can be characterized by strong independence properties, and a great deal of work has been done to generalize it [7, 2, 34, 8]. Of note is the logistic Normal distribution [3], which can be simulated by taking the softmax of $n-1$ normal random variables and an $n$th logit that is deterministically zero. The logistic Normal is an important distribution, because it can effectively model correlations within the simplex [5]. To our knowledge the concrete distribution does not fall completely into any family of distributions previously described. For $\lambda \leq 1$ the concrete is a member of a class of normalized infinitely divisible distributions (S. Favaro, personal communication), and the results of [8] apply.

The idea of using a softmax of Gumbels as a relaxation for a discrete random variable was concurrently considered by [21], where it was called the Gumbel-Softmax. They do not mention whether they used the density in the relaxed objective, opting instead to work with a vector of Gumbels passed through a softmax would result in a looser bound. The idea of using sigmoidal functions with additive input noise to approximate discreteness is also not a new idea. [9] introduced nonlinear Gaussian units which computed their activation by passing Gaussian noise with the mean and variance specified by the input to the unit through a nonlinearity, such as the logistic function. **(author?)** [37] binarized real-valued codes of an autoencoder by adding (Gaussian) noise to the logits before passing them through the logistic function. Most recently, to avoid the difficulty associated with likelihood-ratio methods [25] relaxed the discrete sampling operation by sampling a vector of Gaussians and passing it through a softmax instead.

There is another family of gradient estimators that have been studied in the context of training neural networks with discrete units. These are usually collected under the umbrella of straight-through estimators [4, 32]. The basic idea they use is passing forward discrete values, but taking gradients through the expected value. They have good empirical performance, but have not been shown to be the estimators of any loss function. This is in contrast to gradients from concrete relaxations, which are biased with respect to the discrete graph, but unbiased with respect to the continuous one.

## 5 Experiments

### 5.1 Protocol

The aim of these experiments is to evaluate the effectiveness of the gradients of concrete relaxations for optimizing SCGs with discrete nodes. We considered the tasks in [29]: structured output prediction and density estimation. Both tasks are difficult optimization problems involving fitting probability distributions with hundreds of latent discrete nodes. We compare the performance of concrete reparameterizations to two state-of-the-art score function estimators: VIMCO [29] for optimizing the multisample variational objective ($m > 1$) and NVIL [28] for optimizing the single-sample one ($m = 1$). We report the negative log-likelihood (NLL) of the discrete graph on the test data as the performance metric. Appendix contains more experimental details.

All our models are neural networks with layers of $n$-ary discrete stochastic nodes with $\log_2(n)$-dimensional states on the corners of the hypercube $\{-1, 1\}^{\log_2(n)}$. The distribution of the nodes are parameterized by $n$ real values $\log \alpha_k \in \mathbb{R}$, which we take to be the logits of a discrete random variable $D \sim \text{Discrete}(\alpha)$ with $n$ states. Model descriptions are of the form "(200V–200H∼784V)", read from left to right. This describes the order of conditional sampling, again from left to right, with each integer representing the number of stochastic units in a layer. The letters V and H represent observed and latent variables, respectively. If the leftmost layer is H, then it is sampled unconditionally from some parameters. Conditioning functions are taken from {–, ∼}, where "–" means a linear function of the previous layer and "∼" means a non-linear function. A "layer" of these units is simply the concatenation of some number of independent nodes whose parameters are determined as a function the previous layer. For example a 240 binary layer is a factored distribution over the $\{-1, 1\}^{240}$ hypercube. Whereas a 240 8-ary layer can be seen as a distribution over the same

| binary model | $m$ | MNIST NLL Test Concrete | MNIST NLL Test VIMCO | MNIST NLL Train Concrete | MNIST NLL Train VIMCO | Omniglot NLL Test Concrete | Omniglot NLL Test VIMCO | Omniglot NLL Train Concrete | Omniglot NLL Train VIMCO |
|---|---|---|---|---|---|---|---|---|---|
| (200H – 784V) | 1 | 107.3 | **104.4** | 107.5 | **104.2** | 118.7 | **115.7** | 117.0 | **112.2** |
| | 5 | 104.9 | **101.9** | 104.9 | **101.5** | 118.0 | **113.5** | 115.8 | **110.8** |
| | 50 | 104.3 | **98.8** | 104.2 | **98.3** | 118.9 | **113.0** | 115.8 | **110.0** |
| (200H – 200H – 784V) | 1 | 102.1 | **92.9** | 102.3 | **91.7** | 116.3 | **109.2** | 114.4 | **104.8** |
| | 5 | 99.9 | **91.7** | 100.0 | **90.8** | 116.0 | **107.5** | 113.5 | **103.6** |
| | 50 | 99.5 | **90.7** | 99.4 | **89.7** | 117.0 | **108.1** | 113.9 | **103.6** |
| (200H $\sim$784V) | 1 | **92.1** | 93.8 | **91.2** | 91.5 | **108.4** | 116.4 | **103.6** | 110.3 |
| | 5 | **89.5** | 91.4 | **88.1** | 88.6 | **107.5** | 118.2 | **101.4** | 102.3 |
| | 50 | **88.5** | 89.3 | **86.4** | 86.5 | **108.1** | 116.0 | **100.5** | 100.8 |
| (200H $\sim$200H $\sim$784V) | 1 | **87.9** | 88.4 | 86.5 | **85.8** | **105.9** | 111.7 | **100.2** | 105.7 |
| | 5 | **86.3** | 86.4 | 84.1 | **82.5** | **105.8** | 108.2 | **98.6** | 101.1 |
| | 50 | 85.7 | **85.5** | 83.1 | **81.8** | **106.8** | 113.2 | 97.5 | **95.2** |

Table 1: Density estimation with binary latent variables and distinct models. When $m = 1$, VIMCO stands for NVIL.

hypercube where each of the 80 triples of units are sampled independently from an 8 way discrete distribution over $\{-1, 1\}^3$. These models were initialized with the heuristic of [11] and optimized using Adam [22]. The Appendix contains the details of hyperparameter selection. For concrete relaxations the temperatures were fixed throughout training.

We performed the experiments using the MNIST and Omniglot datasets. These are datasets of $28 \times 28$ images of handwritten digits (MNIST) or letters (Omniglot). For MNIST we used the fixed binarization of [38] and the standard 50,000/10,000/10,000 split into training/validation/testing sets. For Omniglot we sampled a fixed binarization and used the standard 24,345/8,070 split into training/testing sets.

### 5.2 Density Estimation

Density estimation, or generative modelling, is the problem of capturing the distribution of the data by fitting a probabilistic model to samples from this distribution . We will take the latent variable modelling approach described in Section 2.4 and will train the models by optimizing the variational objective $\mathcal{L}_m(\theta, \phi)$ given by Eq. 8. Both our generative models $p_\theta(z,\ x)$ and variational distributions $q_\phi(z \mid x)$ will be parameterized as belief networks as described above.

We trained models by optimizing $\mathcal{L}_m(\theta, \phi)$ for $m \in \{1, 5, 50\}$ and computed the log-likelihood estimates by evaluating $\mathcal{L}_{50,000}(\theta, \phi)$. The results are shown in Table 1. In general, VIMCO outperformed concrete relaxations for linear models and concrete relaxations outperformed VIMCO for non-linear models. We also tested the effectiveness of concrete relaxations on generative models with $n$-ary layers on the $\mathcal{L}_5(\theta, \phi)$ objective. The best 4-ary model achieved test/train NLL 86.7/83.3, the best 8-ary achieved 87.4/84.6 with concrete relaxations, more complete results in Appendix. The relatively poor performance of the 8-ary model may be because moving from 4 to 8 results in a more difficult objective without much added capacity. As a control we trained $n$-ary models using logistic normals as relaxations of discrete distributions (with retuned temperature hyperparameters). Because the discrete zero temperature limit of logistic Normals is a multinomial probit whose mass function is not known, to evaluate the discrete model we simply sampled from the discrete distribution parameterized in the traditional way by the logits learned during training. The best 4-ary model achieved test/train NLL of 88.7/85.0, the best 8-ary model achieved 89.1/85.1.

### 5.3 Structured Output Prediction

Structured output prediction is concerned with modelling the high-dimensional distribution of the observation given a context and can be seen as conditional density estimation. We will consider the task of predicting the bottom half $x_1$ of an image of an MNIST digit given its top half $x_2$, as introduced by [32]. As the problem is multimodal, we follow [32] in using a model with layers of discrete stochastic units between the context and the observation. Conditioned on the top half $x_2$ the

| binary model | | Test NLL | | Train NLL | |
|---|---|---|---|---|---|
| | $m$ | Concrete | VIMCO | Concrete | VIMCO |
| (392V–240H –240H–392V) | 1 | **58.5** | 61.4 | **54.2** | 59.3 |
| | 5 | **54.3** | 54.5 | **49.2** | 52.7 |
| | 50 | 53.4 | **51.8** | **48.2** | 49.6 |
| (392V–240H –240H–240H –392V) | 1 | **56.3** | 59.7 | **51.6** | 58.4 |
| | 5 | **52.7** | 53.5 | **46.9** | 51.6 |
| | 50 | 52.0 | **50.2** | **45.9** | 47.9 |

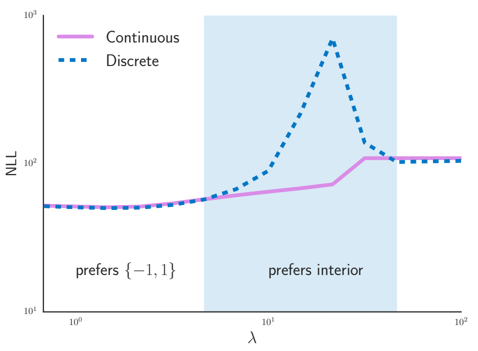


Figure 4: Results for structured prediction on MNIST comparing concrete relaxations to VIMCO. When $m = 1$ VIMCO stands for NVIL. The plot on the right shows the objective (lower is better) for the continuous and discrete graph trained at temperatures $\lambda$. In the shaded region, units prefer to communicate real values in the interior of $(-1, 1)$ and the discretization suffers an integrality gap.

network samples from a distribution $p_\phi(z \mid x_2)$ over layers of stochastic units $z$ then predicts $x_1$ by sampling from a distribution $p_\theta(x_1 \mid z)$. The training objective for a single observation/context pair $(x_1, x_2)$ is

$$\mathcal{L}_m^{SP}(\theta, \phi) = \mathop{\mathbb{E}}_{Z_i \sim p_\phi(z|x_2)} \left[ \log \left( \frac{1}{m} \sum_{i=1}^{m} p_\theta(x_1 \mid Z_i) \right) \right].$$

This objective is a special case of $\mathcal{L}_m(\theta, \phi)$ (Eq. 8) where we use the prior $p_\phi(z|x_2)$ as the variational distribution. Thus, the objective is a lower bound on $\log p_{\theta,\phi}(x_1 \mid x_2)$.

We trained the models by optimizing $\mathcal{L}_m^{SP}(\theta, \phi)$ for $m \in \{1, 5, 50\}$ and evaluated them by computing $\mathcal{L}_{100}^{SP}(\theta, \phi)$. The results are shown in Figure 4. Concrete relaxations more uniformly outperformed VIMCO in this instance. We also trained $n$-ary (392V–240H–240H–240H–392V) models on the $\mathcal{L}_1^{SP}(\theta, \phi)$ objective using the best temperature hyperparameters from density estimation. 4-ary achieved a test/train NLL of 55.4/46.0 and 8-ary achieved 54.7/44.8. As opposed to density estimation, increasing arity uniformly improved the models. We also investigated the hypothesis that for higher temperatures concrete relaxations might prefer the interior of the interval to the boundary points $\{-1, 1\}$. Figure 4 was generated with binary (392V–240H–240H–240H–392V) model trained on $\mathcal{L}_1^{SP}(\theta, \phi)$.

## 6 Conclusion

We introduced the concrete distribution, a continuous relaxation of discrete random variables. The concrete distribution is a new distribution on the simplex with a closed form density parameterized by a vector of positive location parameters and a positive temperature. Crucially, the zero temperature limit of every concrete distribution corresponds to a discrete distribution, and any discrete distribution can be seen as the discretization of a concrete one. The application we considered was training stochastic computation graphs with discrete stochastic nodes. The gradients of concrete relaxations are biased with respect to the original discrete objective, but they are low variance unbiased estimators of a continuous surrogate objective. We showed in a series of experiments that stochastic nodes with concrete distributions can be used effectively to optimize the parameters of a stochastic computation graph with discrete stochastic nodes. We did not find that annealing or automatically tuning the temperature was important for these experiments, but it remains interesting and possibly valuable future work.

### Acknowledgments

We thank Jimmy Ba for the excitement and ideas in the early days, Stefano Favarro for some analysis of the distribution. We also thank Gabriel Barth-Maron and Roger Grosse.

## References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow,

Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] J Aitchison. A general class of distributions on the simplex. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 136–146, 1985.

[3] J Atchison and Sheng M Shen. Logistic-normal distributions: Some properties and uses. *Biometrika*, 67(2):261–272, 1980.

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

[5] David Blei and John Lafferty. Correlated topic models. 2006.

[6] Yuri Burda, Roger Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *ICLR*, 2016.

[7] Robert J Connor and James E Mosimann. Concepts of independence for proportions with a generalization of the dirichlet distribution. *Journal of the American Statistical Association*, 64(325):194–206, 1969.

[8] Stefano Favaro, Georgia Hadjicharalambous, and Igor Prünster. On a class of distributions on the simplex. *Journal of Statistical Planning and Inference*, 141(9):2987 – 3004, 2011.

[9] Brendan Frey. Continuous sigmoidal belief networks trained using slice sampling. In *NIPS*, 1997.

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

[11] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

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

[13] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471–476, 2016.

[14] Evan Greensmith, Peter L. Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *JMLR*, 5, 2004.

[15] Edward Grefenstette, Karl Moritz Hermann, Mustafa Suleyman, and Phil Blunsom. Learning to transduce with unbounded memory. In *Advances in Neural Information Processing Systems*, pages 1828–1836, 2015.

[16] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

[17] Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. *arXiv preprint arXiv:1310.8499*, 2013.

[18] Shixiang Gu, Sergey Levine, Ilya Sutskever, and Andriy Mnih. MuProp: Unbiased backpropagation for stochastic neural networks. *ICLR*, 2016.

[19] Emil Julius Gumbel. *Statistical theory of extreme values and some practical applications: a series of lectures*. Number 33. US Govt. Print. Office, 1954.

[20] Matthew D Hoffman, David M Blei, Chong Wang, and John William Paisley. Stochastic variational inference. *JMLR*, 14(1):1303–1347, 2013.

[21] E. Jang, S. Gu, and B. Poole. Categorical Reparameterization with Gumbel-Softmax. *ArXiv e-prints*, November 2016.

[22] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

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

[24] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[25] Tomáš Kočiský, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. Semantic parsing with semi-supervised sequential autoencoders. In *EMNLP*, 2016.

[26] Chris J Maddison. A Poisson process model for Monte Carlo. *arXiv preprint arXiv:1602.05986*, 2016.

[27] Chris J Maddison, Daniel Tarlow, and Tom Minka. A$^*$ Sampling. In *NIPS*, 2014.

[28] Andriy Mnih and Karol Gregor. Neural variational inference and learning in belief networks. In *ICML*, 2014.

[29] Andriy Mnih and Danilo Jimenez Rezende. Variational inference for monte carlo objectives. In *ICML*, 2016.

[30] Christian A Naesseth, Francisco JR Ruiz, Scott W Linderman, and David M Blei. Rejection sampling variational inference. *arXiv preprint arXiv:1610.05683*, 2016.

[31] John William Paisley, David M. Blei, and Michael I. Jordan. Variational bayesian inference with stochastic search. In *ICML*, 2012.

[32] Tapani Raiko, Mathias Berglund, Guillaume Alain, and Laurent Dinh. Techniques for learning binary stochastic feedforward neural networks. *arXiv preprint arXiv:1406.2989*, 2014.

[33] Rajesh Ranganath, Sean Gerrish, and David M. Blei. Black box variational inference. In *AISTATS*, 2014.

[34] William S Rayens and Cidambi Srinivasan. Dependence properties of generalized liouville distributions on the simplex. *Journal of the American Statistical Association*, 89(428):1465–1470, 1994.

[35] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *ICML*, 2014.

[36] Francisco JR Ruiz, Michalis K Titsias, and David M Blei. The generalized reparameterization gradient. *arXiv preprint arXiv:1610.02287*, 2016.

[37] Ruslan Salakhutdinov and Geoffrey Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[38] Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *ICML*, 2008.

[39] John Schulman, Nicolas Heess, Theophane Weber, and Pieter Abbeel. Gradient estimation using stochastic computation graphs. In *NIPS*, 2015.

[40] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016.

[41] Michalis Titsias and Miguel Lázaro-Gredilla. Doubly stochastic variational bayes for non-conjugate inference. In Tony Jebara and Eric P. Xing, editors, *ICML*, 2014.

[42] Michalis Titsias and Miguel Lázaro-Gredilla. Local expectation gradients for black box variational inference. In *NIPS*, 2015.

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

[44] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[45] John I Yellott. The relationship between luce's choice axiom, thurstone's theory of comparative judgment, and the double exponential distribution. *Journal of Mathematical Psychology*, 15(2):109–144, 1977.

## Appendix

### Proof of Proposition 1

Let $X \sim \text{Concrete}(\alpha, \lambda)$ with location parameters $\alpha \in (0, \infty)^n$ and temperature $\lambda \in (0, \infty)$.

1. Let $G_k \sim$ Gumbel i.i.d., consider

$$Y_k = \frac{\exp((\log \alpha_k + G_k)/\lambda)}{\sum_{i=1}^n \exp((\log \alpha_i + G_i)/\lambda)}$$

Let $Z_k = \log \alpha_k + G_k$, which has density

$$\alpha_k \exp(-z_k) \exp(-\alpha_k \exp(-z_k))$$

We will consider the invertible transformation

$$F(z_1, \ldots, z_n) = (y_1, \ldots, y_{n-1}, c)$$

where

$$y_k = \exp(z_k/\lambda) c^{-1}$$
$$c = \sum_{i=1}^n \exp(z_i/\lambda)$$

then

$$F^{-1}(y_1, \ldots, y_{n-1}, c) = (\lambda(\log y_1 + \log c), \ldots, \lambda(\log y_{n-1} + \log c), \lambda(\log y_n + \log c))$$

where $y_n = 1 - \sum_{i=1}^{n-1} y_i$. This has Jacobian

$$\begin{bmatrix} \lambda y_1^{-1} & 0 & 0 & 0 & \ldots & 0 & \lambda c^{-1} \\ 0 & \lambda y_2^{-1} & 0 & 0 & \ldots & 0 & \lambda c^{-1} \\ 0 & 0 & \lambda y_3^{-1} & 0 & \ldots & 0 & \lambda c^{-1} \\ & & \vdots & & & & \\ -\lambda y_n^{-1} & -\lambda y_n^{-1} & -\lambda y_n^{-1} & -\lambda y_n^{-1} & \ldots & -\lambda y_n^{-1} & \lambda c^{-1} \end{bmatrix}$$

by adding $y_i/y_n$ times each of the top $n-1$ rows to the bottom row we see that this Jacobian has the same determinant as

$$\begin{bmatrix} \lambda y_1^{-1} & 0 & 0 & 0 & \ldots & 0 & \lambda c^{-1} \\ 0 & \lambda y_2^{-1} & 0 & 0 & \ldots & 0 & \lambda c^{-1} \\ 0 & 0 & \lambda y_3^{-1} & 0 & \ldots & 0 & \lambda c^{-1} \\ & & \vdots & & & & \\ 0 & 0 & 0 & 0 & \ldots & 0 & \lambda (c y_n)^{-1} \end{bmatrix}$$

and thus the determinant is equal to

$$\frac{\lambda^n}{c \prod_{i=1}^k y_i}$$

all together we have the density

$$\frac{\lambda^n \prod_{k=1}^n \alpha_k \exp(-\lambda \log y_k - \lambda \log c) \exp(-\alpha_k \exp(-\lambda \log y_k - \lambda \log c))}{c \prod_{i=1}^n y_i}$$

with $r = \log c$ change of variables we have density

$$\frac{\lambda^n \prod_{k=1}^n \alpha_k \exp(-\lambda r) \exp(-\alpha_k \exp(-\lambda \log y_k - \lambda r))}{\prod_{i=1}^n y_i^{\lambda+1}} =$$

$$\frac{\lambda^n \prod_{k=1}^n \alpha_k}{\prod_{i=1}^n y_i^{\lambda+1}} \exp(-n\lambda r) \exp(-\sum_{i=1}^n \alpha_i \exp(-\lambda \log y_i - \lambda r)) =$$

letting $\gamma = \log(\sum_{n=1}^n \alpha_k y_k^{-\lambda})$

$$\frac{\lambda^n \prod_{k=1}^n \alpha_k}{\prod_{i=1}^n y_i^{\lambda+1} \exp(\gamma)} \exp(-n\lambda r + \gamma) \exp(-\exp(-\lambda r + \gamma)) =$$

integrating out $r$

$$\frac{\lambda^n \prod_{k=1}^n \alpha_k}{\prod_{i=1}^n y_i^{\lambda+1} \exp(\gamma)} \left( \frac{\exp(-\gamma n + \gamma)\Gamma(n)}{\lambda} \right) =$$
$$\frac{\lambda^{n-1} \prod_{k=1}^n \alpha_k}{\prod_{i=1}^n y_i^{\lambda+1}} \left(\exp(-\gamma n)\Gamma(n)\right) =$$
$$(n-1)!\lambda^{n-1} \frac{\prod_{k=1}^n \alpha_k y_k^{-\lambda-1}}{(\sum_{n=1}^n \alpha_k y_k^{-\lambda})^n}$$

Thus $Y \overset{d}{=} X$.

2. Follows directly from 1 and the Gumbel-Max trick [26].
3. Follows directly from 1 and the Gumbel-Max trick [26].
4. Let $\lambda \leq (n-1)^{-1}$. The density of $X$ can be rewritten as

$$p_{\alpha,\lambda}(x) \propto \prod_{k=1}^n \frac{\alpha_k y^{-\lambda-1}}{\sum_{i=1}^n \alpha_i y_i^{-\lambda}}$$
$$= \prod_{k=1}^n \frac{\alpha_k y_k^{\lambda(n-1)-1}}{\sum_{i=1}^n \alpha_i \prod_{j\neq i} y_j^{\lambda}}$$

Thus, the log density is up to an additive constant $C$

$$\log p_{\alpha,\lambda}(x) = \sum_{k=1}^n (\lambda(n-1)-1)\log y_k - n \log \left( \sum_{k=1}^n \alpha_k \prod_{j\neq k} y_j^{\lambda} \right) + C$$

If $\lambda \leq (n-1)^{-1}$, then the first $n$ terms are convex, because $-\log$ is convex. For the last term, $-\log$ is convex and non-increasing and $\prod_{j\neq k} y_j^{\lambda}$ is concave for $\lambda \leq (n-1)^{-1}$. Thus, their composition is convex. The sum of convex terms is convex, finishing the proof.

**Binary Gumbel-Max Trick**

Bernoulli random variables are an important special case of discrete distributions. They take states in $\{0,1\}$. Consider the binary version of the Gumbel-Max trick from Figure 1a. Let $D \sim \text{Discrete}(\alpha)$ for $\alpha \in (0,\infty)^2$ be that two state discrete random variable on $\{0,1\}^2$ parameterized as in Figure 1a. The distribution is degenerate, because $D_1 = 1 - D_2$. Therefore we consider just $D_1$. The event that $D_1 = 1$ is the event that $\{G_1 + \log \alpha_1 > G_2 + \log \alpha_2\}$ where $G_k \sim \text{Gumbel}$. The difference of two Gumbels is a Logistic distribution. $L \sim \text{Logistic}$ has the logistic function $\sigma(x) = 1/(1+\exp(-x))$ for a CDF. Therefore, $L \overset{d}{=} \log U - \log(1-U)$ where $U \sim \text{Uniform}(0,1)$. If $\alpha = \alpha_1/\alpha_2$ we have ultimately,

$$\mathbb{P}(D_1 = 1) = \mathbb{P}(G_1 + \log \alpha_1 > G_2 + \log \alpha_2) = \mathbb{P}(L + \log \alpha > 0)$$

Thus, $D_1 \overset{d}{=} H(\log \alpha + \log U - \log(1-U))$, where $H$ is the unit step function.

To generalize this consider an arbitrary random variable $X$ with infinite support on $\mathbb{R}$. If $\Phi : \mathbb{R} \to [0,1]$ is the CDF of $X$, then

$$\mathbb{P}(H(X) = 1) = 1 - \Phi(0)$$

If we want this to have a Bernoulli distribution with probability $\alpha/(1+\alpha)$, then we should solve the equation

$$1 - \Phi(0) = \frac{\alpha}{1+\alpha}.$$

This gives $\Phi(0) = 1/(1+\alpha)$, which can be accomplished by relocating the random variable $Y$ with CDF $\Phi$ to be $X = Y - \Phi^{-1}(1/(1+\alpha))$.

**Implementation Details for Concrete Distributions**

In this section we include some tips for implementing and using the concrete distribution. We use the following notation

$$\sigma(x) = \frac{1}{1+\exp(-x)} \qquad \underset{k=1}{\overset{n}{\mathrm{L\Sigma E}}}\{x_k\} = \log\left(\sum_{k=1}^{n}\exp(x_k)\right)$$

Both sigmoid and log-sum-exp are common operations in libraries like TensorFlow or theano.

*Which random variable to treat as the stochastic node.* There can be numerical issues when working with concrete random variables, if they are implemented naively. When implementing a SCG like a variational autoencoder the question arises as to which part of the graph to treat as the stochastic node on which log-likelihood terms are computed. It's tempting in the case of concrete random variables to treat the Gumbels as the stochastic node on which the log-likelihood terms are evaluated and the softmax as downstream computation. This will be a looser bound in the context of variational inference than the corresponding bound when treating the concrete relaxed states as the node. Treating the concrete random variables as the stochastic nodes can lead to underflow then log of 0 issues. Outside of generic underflow/overflow issues when computing with softmaxes, is important to consider these issues when implementing a generic drop-in distribution class.

The solution we found to work well was to consider the following vector in $\mathbb{R}^n$ for location parameters $\alpha \in (0,\infty)^n$ and $\lambda \in (0,\infty)$ and $G_k \sim \mathrm{Gumbel}$,

$$Y_k = \frac{\log\alpha_k + G_k}{\lambda} - \underset{i=1}{\overset{n}{\mathrm{L\Sigma E}}}\left\{\frac{\log\alpha_i + G_i}{\lambda}\right\}$$

$Y \in \mathbb{R}^n$ has the property that $\exp(Y) \sim \mathrm{Concrete}(\alpha,\lambda)$, therefore we call $Y$ an $\mathrm{ExpConcrete}(\alpha,\lambda)$. The advantage of this reparameterization is that the KL terms of a variational loss are invariant under *invertible* transformation. $\exp$ is invertible, so the KL between two expconcrete random variables is the same as the KL between two concrete random variables. The log-density $\log p_{\alpha,\lambda}(y)$ of an $\mathrm{ExpConcrete}(\alpha,\lambda)$ is also simple to compute:

$$\log p_{\alpha,\lambda}(y) = \log((n-1)!) + (n-1)\log\lambda + \left(\sum_{k=1}^{n}\log\alpha_k - \lambda y_k\right) - n\,\underset{k=1}{\overset{n}{\mathrm{L\Sigma E}}}\left\{\log\alpha_k - \lambda y_k\right\}$$

for $y \in \mathbb{R}^n$ such that $\mathrm{L\Sigma E}_{k=1}^{n}\{y_k\} = 0$. Therefore, we used $\mathrm{ExpConcrete}$ random variables as the stochastic nodes and treated $\exp$ as a downstream computation. This has the additional advantage that it can also represent the discrete limit. In the limit of $\lambda \to 0$ $\mathrm{ExpConcrete}$ random variables become discrete random variables over the one-hot vectors of $d \in \{-\infty, 0\}^n$ where $\mathrm{L\Sigma E}_{k=1}^{n}\{d_k\} = 0$, and following this with $\exp$ results in the traditional one-hot vectors in $\{0,1\}^n$.

In the binary case, the logistic function is invertible, so it makes most sense to treat the logit plus noise as the stochastic node. In particular, the binary random node was sample from:

$$X = \frac{\log\alpha + \log U - \log(1-U)}{\lambda}$$

where $U \sim \mathrm{Uniform}(0,1)$ and always followed by $\sigma$ as downstream computation. $\log U - \log(1-U)$ is a logistic random variable, details in the cheat sheet, and so the log-density $\log p_{\alpha,\lambda}(x)$ of this node (before applying $\sigma$) is

$$\log p_{\alpha,\lambda}(x) = \log\lambda - \lambda x + \mu - 2\log(1+\exp(-\lambda x + \mu))$$

*Choosing the temperature.* The success of concrete relaxations will depend heavily on the choice of temperature during training. It is important that the relaxed nodes are not able to represent a precise real valued mode in the interior of the simplex as in Figure 2d. For example, choosing additive Gaussian noise $\epsilon \sim \mathrm{Normal}(0,1)$ with the logistic function $\sigma(x)$ to get relaxed Bernoullis of the form $\sigma(\epsilon + \mu)$ will result in a large mode in the centre of the interval. This is because the tails of the Gaussian distribution drop off much faster than the rate at which $\sigma$ squashes. Even including a temperature parameter does not completely solve this problem; the density of $\sigma((\epsilon+\mu)/\lambda)$ at *any* temperature still goes to 0 as its approaches the boundaries 0 and 1 of the unit interval. Therefore 4 of Proposition 1 is a conservative guideline for generic $n$-ary concrete relaxations; at temperatures lower than $(n-1)^{-1}$ we are guaranteed not to have any modes in the interior for any $\alpha \in (0, \infty)^n$. In the case of the binary concrete distribution, the tails of the logistic additive noise are balanced with the logistic squashing function and for temperatures $\lambda \leq 1$ the density of the binary concrete distribution is log-convex for all parameters $\alpha$, see Figure 3b. Still, practice will often disagree with theory here. The peakiness of the concrete distribution increases with $n$, so much higher temperatures are tolerated (usually necessary).

For $n = 1$ temperatures $\lambda \leq (n-1)^{-1}$ is a good guideline. For $n > 1$ taking $\lambda \leq (n-1)^{-1}$ is not necessarily a good guideline, although it will depend on $n$ and the specific application. As $n \to \infty$ the concrete distribution becomes peakier, because the random normalizing constant $\sum_{k=1}^{n} \exp((\log \alpha_k + G_k)/\lambda)$ grows. This means that practically speaking the optimization can tolerate much higher temperatures than $(n-1)^{-1}$. We found in the cases $n = 4$ that $\lambda = 1$ was the best temperature and in $n = 8$, $\lambda = 2/3$ was the best. Yet $\lambda = 2/3$ was the best single performing temperature across the $n \in \{2, 4, 8\}$ cases that we considered. We recommend starting in that ball-park and exploring for any specific application.

## Experimental Details

*— vs ∼.* The conditioning functions we used were either linear or non-linear. Non-linear consisted of two $\tanh$ layers of the same size as the preceding stochastic layer in the computation graph.

*$n$-ary layers.* All our models are neural networks with layers of $n$-ary discrete stochastic nodes with $\log_2(n)$-dimensional states on the corners of the hypercube $\{-1, 1\}^{\log_2(n)}$. For a generic $n$-ary node sampling proceeds as follows. Sample a $n$-ary discrete random variable $D \sim \mathrm{Discrete}(\alpha)$ for $\alpha \in (0, \infty)^n$. If $C$ is the $\log_2(n) \times n$ matrix, which lists the corners of the hypercube $\{-1, 1\}^{\log_2(n)}$ as columns, then we took $Y = CD$ as downstream computation on $D$. The corresponding concrete relaxation is to take $X \sim \mathrm{Concrete}(\alpha, \lambda)$ for some fixed temperature $\lambda \in (0, \infty)$ and set $\tilde{Y} = CX$. For the binary case, this amounts to simply sampling $U \sim \mathrm{Uniform}(0, 1)$ and taking $Y = 2H(\log U - \log(1-U) + \log \alpha) - 1$. The corresponding binary concrete relaxation is $\tilde{Y} = 2\sigma((\log U - \log(1-U) + \log \alpha)/\lambda) - 1$.

*Bias initialization.* All biases were initialized to 0 with the exception of the biases in the prior decoder distribution over the 784 or 392 observed units. These were initialized to the logit of the base rate averaged over the respective dataset (MNIST or Omniglot).

*Centering.* We also found it beneficial to center the layers of the inference network during training. The activity in $(-1, 1)^d$ of each stochastic layer was centered during training by maintaining a exponentially decaying average with rate 0.9 over minibatches. This running average was subtracted from the activity of the layer *before* it was updated. Gradients did not flow throw this computation, so it simply amounted to a dynamic offset. The averages were *not* updated during the evaluation.

*Hyperparameter selection.* All models were initialized with the heuristic of [11] and optimized using Adam [22] with parameters $\beta_1 = 0.9, \beta_2 = 0.999$ for $10^7$ steps on minibatches of size 64. Hyperparameters were selected on the MNIST dataset by grid search taking the values that performed best on the validation set. Learning rates were chosen from $\{10^{-4}, 3 \cdot 10^{-4}, 10^{-3}\}$ and weight decay from $\{0, 10^{-2}, 10^{-1}, 1\}$. Two sets of hyperparameters were selected, one for linear models and one for non-linear models. The linear models' hyperparameters were selected with the 200H–200H–784V density model on the $\mathcal{L}_5(\theta, \phi)$ objective. The non-linear models' hyperparameters were selected with the 200H∼200H∼784V density model on the $\mathcal{L}_5(\theta, \phi)$ objective. For density estimation, the concrete relaxation hyperparameters were (weight decay = 0, learning rate = $3 \cdot 10^{-4}$) for linear and (weight decay = 0, learning rate = $10^{-4}$) for non-linear. For structured prediction concrete relaxations used (weight decay = $10^{-3}$, learning rate = $3 \cdot 10^{-4}$).

In addition to tuning learning rate and weight decay, we tuned temperatures for the concrete relaxations on the density estimation task. We found it valuable to have different values for the prior and posterior distributions. In particular, for binary we found that (prior $\lambda = 1/2$, posterior $\lambda = 2/3$) was best, for 4-ary we found (prior $\lambda = 2/3$, posterior $\lambda = 1$) was best, and (prior $\lambda = 2/5$, posterior $\lambda = 2/3$) for 8-ary. No temperature annealing was used. For structured prediction we used just the corresponding posterior $\lambda$.

We performed early stopping when training with the score function estimators (VIMCO/NVIL) as they were much more prone to overfitting.

## 6.1 Extra Results

| | | MNIST NLL | | Omniglot NLL | |
|---|---|---|---|---|---|
| | $m$ | Test | Train | Test | Train |
| binary | 1 | 91.9 | 90.7 | 108.0 | 102.2 |
| (240H | 5 | 89.0 | 87.1 | 107.7 | 100.0 |
| $\sim$784V) | 50 | 88.4 | 85.7 | 109.0 | 99.1 |
| 4-ary | 1 | 91.4 | 89.7 | 110.7 | 1002.7 |
| (240H | 5 | 89.4 | 87.0 | 110.5 | 100.2 |
| $\sim$784V) | 50 | 89.7 | 86.5 | 113.0 | 100.0 |
| 8-ary | 1 | 92.5 | 89.9 | 119.61 | 105.3 |
| (240H | 5 | 90.5 | 87.0 | 120.7 | 102.7 |
| $\sim$784V) | 50 | 90.5 | 86.7 | 121.7 | 101.0 |
| binary | 1 | 87.9 | 86.0 | 106.6 | 99.0 |
| (240H$\sim$240H | 5 | 86.6 | 83.7 | 106.9 | 97.1 |
| $\sim$784V) | 50 | 86.0 | 82.7 | 108.7 | 95.9 |
| 4-ary | 1 | 87.4 | 85.0 | 106.6 | 97.8 |
| (240H$\sim$240H | 5 | 86.7 | 83.3 | 108.3 | 97.3 |
| $\sim$784V) | 50 | 86.7 | 83.0 | 109.4 | 96.8 |
| 8-ary | 1 | 88.2 | 85.9 | 111.3 | 102.5 |
| (240H$\sim$240H | 5 | 87.4 | 84.6 | 110.5 | 100.5 |
| $\sim$784V) | 50 | 87.2 | 84.0 | 111.1 | 99.5 |

Table 2: Density estimation using concrete relaxations with distinct arity of layers.

**Cheat Sheet**

$$\sigma(x) = \frac{1}{1+\exp(-x)} \qquad \underset{k=1}{\overset{n}{\mathrm{L\Sigma E}}}\{x_k\} = \log\left(\sum_{k=1}^{n} \exp(x_k)\right)$$

$$\log \Delta^{n-1} = \left\{ x \in \mathbb{R}^n \mid x_k \in (-\infty, 0),\ \underset{k=1}{\overset{n}{\mathrm{L\Sigma E}}}\{x_k\} = 0 \right\}$$

| Distribution and Domains | Reparameterization/How To Sample | Mass/Density |
|---|---|---|
| $G \sim \mathrm{Gumbel}$<br>$G \in \mathbb{R}$ | $G \overset{d}{=} -\log(-\log(U))$ | $\exp(-g - \exp(-g))$ |
| $L \sim \mathrm{Logistic}$<br>$L \in \mathbb{R}$ | $L \overset{d}{=} \log(U) - \log(1-U)$ | $\dfrac{\exp(-l)}{(1+\exp(-l))^2}$ |
| $X \sim \mathrm{Logistic}(\mu, \lambda)$<br>$\mu \in \mathbb{R}$<br>$\lambda \in (0, \infty)$ | $X \overset{d}{=} \dfrac{L+\mu}{\lambda}$ | $\dfrac{\lambda \exp(-\lambda x + \mu)}{(1+\exp(-\lambda x + \mu))^2}$ |
| $X \sim \mathrm{Bernoulli}(\alpha)$<br>$X \in \{0,1\}$<br>$\alpha \in (0, \infty)$ | $X \overset{d}{=} \begin{cases} 1 & \text{if } L + \log\alpha \geq 0 \\ 0 & \text{otherwise} \end{cases}$ | $\dfrac{\alpha}{1+\alpha}$ if $x = 1$ |
| $X \sim \mathrm{BinConcrete}(\alpha, \lambda)$<br>$X \in (0,1)$<br>$\alpha \in (0, \infty)$<br>$\lambda \in (0, \infty)$ | $X \overset{d}{=} \sigma((L + \log\alpha)/\lambda)$ | $\dfrac{\lambda\alpha x^{-\lambda-1}(1-x)^{-\lambda-1}}{(\alpha x^{-\lambda} + (1-x)^{-\lambda})^2}$ |
| $X \sim \mathrm{Discrete}(\alpha)$<br>$X \in \{0,1\}^n$<br>$\sum_{k=1}^{n} X_k = 1$<br>$\alpha \in (0, \infty)^n$ | $X_k \overset{d}{=} \begin{cases} 1 & \text{if } \log\alpha_k + G_k > \log\alpha_i + G_i \text{ for } i \neq k \\ 0 & \text{otherwise} \end{cases}$ | $\dfrac{\alpha_k}{\sum_{i=1}^{n} \alpha_i}$ if $x_k = 1$ |
| $X \sim \mathrm{Concrete}(\alpha, \lambda)$<br>$X \in \Delta^{n-1}$<br>$\alpha \in (0, \infty)^n$<br>$\lambda \in (0, \infty)$ | $X_k \overset{d}{=} \dfrac{\exp((\log\alpha_k + G_k)/\lambda)}{\sum_{i=1}^{n} \exp((\log\alpha_k + G_i)/\lambda)}$ | $\dfrac{(n-1)!}{\lambda^{-(n-1)}} \displaystyle\prod_{k=1}^{n} \dfrac{\alpha_k x_k^{-\lambda-1}}{\sum_{i=1}^{n} \alpha_i x_i^{-\lambda}}$ |
| $X \sim \mathrm{ExpConcrete}(\alpha, \lambda)$<br>$X \in \log\Delta^{n-1}$<br>$\alpha \in (0, \infty)^n$<br>$\lambda \in (0, \infty)$ | $X_k \overset{d}{=} \dfrac{\log\alpha_k + G_k}{\lambda} - \underset{i=1}{\overset{n}{\mathrm{L\Sigma E}}}\left\{\dfrac{\log\alpha_i + G_i}{\lambda}\right\}$ | $\dfrac{(n-1)!}{\lambda^{-(n-1)}} \displaystyle\prod_{k=1}^{n} \dfrac{\alpha_k \exp(-\lambda x_k)}{\sum_{i=1}^{n} \alpha_i \exp(-\lambda x_i)}$ |

Table 3: Cheat sheet for the random variables we use in this work. Note that some of these are atypical parameterizations, particularly the Bernoulli and logistic random variables. The table only assumes that you can sample uniform random numbers $U \sim \mathrm{Uniform}(0,1)$. From there on it may define random variables and reuse them later on. For example, $L \sim \mathrm{Logistic}$ is defined in the second row, and after that point $L$ represents a Logistic random variable that can be replaced by $\log U - \log(1-U)$. Whenever random variables are indexed, e.g. $G_k$, they represent separate independent calls to a random number generator.