Rethinking Variational Inference for Probabilistic Programs with Stochastic Support

Anonymous Author(s)
Affiliation
Address
email

Abstract

We introduce *Support Decomposition Variational Inference* (SDVI), a new variational inference (VI) approach for probabilistic programs with stochastic support. Existing approaches to this problem rely on designing a single global variational guide on a variable-by-variable basis, while maintaining the stochastic control flow of the original program. SDVI instead breaks the program down into sub-programs with static support, before automatically building separate sub-guides for each. This decomposition significantly aids in the construction of suitable variational families, enabling, in turn, substantial improvements in inference performance.

1 Introduction

Probabilistic programming systems (PPSs) enable users to express probabilistic models with computer programs and provide tools for inference. Many PPS, such as Stan [1] or PyMC3 [2], limit the expressiveness of their language to ensure that the programs in their language always correspond to models with static support—i.e. the number of variables and their support do not vary between program executions. In contrast, *universal PPSs* [3–11] can encode programs where the sequence of variables itself—not just the variable values—changes between executions, leading to models with stochastic support. These models have applications in numerous fields, such as natural language processing [12], Bayesian Nonparametrics [13], and statistical phylogenetics [14]. A wide range of simulator-based models similarly require such stochastic control flow [15–17].

The effectiveness of PPSs is heavily reliant on the underlying inference schemes they support. Variational inference (VI) is one of the most popular such schemes, both in PPSs and more generally [18–20]. This popularity is due to its ability to use derivatives to scale to large datasets and high-dimensional models [21–24], often providing much faster and more scalable inferences compared to Monte Carlo approaches [25]. To provide the required derivatives, a number of modern universal PPSs—such as Pyro [5], ProbTorch [26], PyProb [15], Gen [7], and Turing [6]—have introduced automatic differentiation [27] capabilities for programs with stochastic control flow. One of the core aims behind these developments was to support VI schemes in such settings [5].

However, this aim has arguably gone unfulfilled, due to the lack of suitable VI algorithms themselves that can effectively deal with the unique challenges stochastic support poses. In particular, automating the construction of appropriate variational families, typically known as guides in PPSs, has become a serious stumbling block [28]. Indeed, existing systems often give unreliable results when running them on models with stochastic support, as we show in Figure 1.

We argue that a significant factor of this shortfall is that existing approaches all construct the guide on a variable-by-variable basis [29–32]. Namely, they use a single global guide that mirrors the control flow of the input program, then introduce a variational approximation for each unique variable. This is problematic because control flows inherently introduce discontinuities into the program's density

Submitted to 36th Conference on Neural Information Processing Systems (NeurIPS 2022). Do not distribute.

```
def model():
  x = sample("x", Normal(0, 1))
  if x < 0:
    z = sample("z1", Normal(-3, 1))
  else:
    z = sample("z2", Normal(3, 1))
  sample("y", Normal(z, 2), obs=2.0)
```

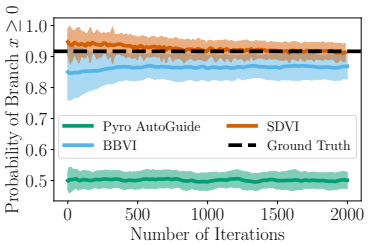


Figure 1: Pyro program with stochastic control flow [Left]. Users can automatically create a guide which mirrors the control flow of the input program using Pyro's built-in `AutoGuide` functionality. However, the stochastic control flow here causes this to provide highly inaccurate posterior estimators, as shown on the right [Green]. This can be partly corrected by overwriting Pyro's default use of reparameterized gradients with a score function estimator [BBVI, Blue]. However, the limitations of the variational family itself still lead to unsatisfactory performance, despite the problem's simplicity. By instead breaking down the guide over paths, SDVI [Red] is able to provide accurate inference. Results computed over 10^2 replications, plotted for each method are mean and standard deviation.

function, such that the conditional distribution of each variable will typically change significantly whenever the program path—that is the sequence of random variables—changes. Thus it is extremely challenging to learn a single approximation for each variable that is appropriate across all paths. Further, as the set of variables that exist can itself be stochastic, it is difficult for such guides to appropriately condition on previously sampled variables. Existing approaches, therefore, typically rely on mean-field assumptions [30], thereby forgoing any conditioning on the program path itself, consequently leading to poor approximations for most problems.

To overcome these difficulties, we propose *Support Decomposition Variational Inference* (SDVI), a new VI approach based around a novel way of constructing the variational guide. Rather than building the guide on a variable-by-variable basis, SDVI instead breaks the problem down over *paths*. Specifically, by utilizing the fact that any program can be reformulated as a mixture of *straight-line programs* (SLPs) [33–35]—each defined by a unique realization of the path—SDVI constructs the guide as a mixture of sub-guides with static support. We show that optimizing the variational objective with this guide structure leads to a natural decomposition of the overall optimization problem into independent sub-problems, each taking the form of a VI with static support. The sub-guides can thus be effectively constructed and trained using more standard VI techniques, before being recombined to form our overall variational approximation. To make SDVI accessible to a wide audience, we have implemented it in Pyro [5]. We evaluate it on a set of example problems with synthetic and real-world data, finding that it provides substantial improvements over existing techniques.

2 Background

2.1 Probabilistic Programs in Universal PPSs

PPSs allow users to express probabilistic models and condition on observed data [32, 36]. A common mechanism to achieve this is to extend standard programming languages with two new primitives: `sample` and `observe`.[1] `sample(id, dist)` draws samples from the distribution object `dist`, where `id` is a unique lexical identifier. `observe(id, data, dist)` enables conditioning on an observed outcome `data`, where `dist` and `id` are as before. For problems admitting a Bayesian formulation, the `sample` and `observe` terms can informally be thought of as prior and likelihood factors respectively.

Universal PPSs utilize higher-order programming languages to allow users to write complex models whose support can vary from one execution to the next, e.g. stochastic branching can mean certain variables only sometimes exist. This can substantially complicate the process of performing inference.

A probabilistic program in a universal PPS defines an unnormalized *density function* $\gamma(x_{1:n_x})$ over the *raw random draws* $x_{1:n_x} \in \mathcal{X}$—defined as the (sequences of) direct outputs of `sample` statements—where $n_x \in \mathbb{N}^+$ is itself potentially random. Though each outcome of $x_{1:n_x}$ uniquely defines a program execution, it is notationally convenient to further associate an *address* a_i to each draw x_i that indicates the position in the program the draw was made. This address can be uniquely defined as the tuple formed by the `id` of the `sample` and the number of times that `sample` has previously been called. For a given execution of the program, the addresses now form an *address path* $A = a_{1:n_x}$.

[1] Pyro instead overloads the `sample` primitive: `sample(id,dist,obs=data)` $\equiv$ `observe(id,data,dist)`.

Each `sample` statement encountered during execution contributes the factor $f_{a_i}(x_i \mid \eta_i)$ to the program density, where a_i is the address of the `sample` statement, f_{a_i} is a parameterized density function, and η_i are its associated parameters. Similarly, each encountered `observe` statement contributes the factor $g_{b_j}(y_j \mid \phi_j)$, with b_j denoting an address, y_j the observed value, g_{b_j} a parameterized density function, and ϕ_j its parameters. Following [37, §4.3.2], we write the program density function as

$$\gamma(x_{1:n_x}) := \prod_{i=1}^{n_x} f_{a_i}(x_i \mid \eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j \mid \phi_j). \tag{1}$$

All of $n_x, n_y, a_{1:n_x}$, $\eta_{1:n_x}$, $y_{1:n_y}$, $b_{1:n_y}$, and $\phi_{1:n_y}$ are potentially random variables. The goal of *inference* is to approximate the conditional distribution of the program, which has normalized density $\pi(x_{1:n}) = \gamma(x_{1:n})/Z$ with marginal likelihood $Z = \int_{\mathcal{X}} \gamma(x_{1:n_x}) dx_{1:n_x}$ and the integral is computed with respect to a reference measure that is implicitly defined by the `sample` statements in the program.

2.2 Variational Inference

Variational Inference (VI) [18, 38] solves the inference problem by transforming it to an optimization problem. Specifically, given an unnormalized joint distribution $\gamma(x)$ and a parameterized distribution $q(x;\phi)$, VI computes the variational parameters ϕ such that $q(x;\phi)$ most closely approximates $\pi(x) = \gamma(x)/Z$. This is most commonly done by maximizing the *Evidence Lower Bound* (ELBO) $\mathcal{L}(\phi) := \mathbb{E}_{q(x;\phi)}\left[\log {}^{\gamma(x)}/_{q(x;\phi)}\right]$ via stochastic gradient ascent using Monte Carlo estimates of $\nabla_\phi \mathcal{L}(\phi)$ [39]. Two popular estimators are the score function estimator [40, 41], and the reparameterized gradient estimator [24, 42, 43]. The latter provides lower variance gradient estimates but requires that the distribution $q(x;\phi)$ can be reparameterized and that $\gamma(x)$ is differentiable everywhere.

3 Difficulties for Variational Inference in Universal PPSs

The starting point for any VI scheme is to construct an appropriate variational family, also known as a guide. In a PPS, we further desire to (at least partially) automate the process of constructing this guide. Existing methods for this all generate the guide on a variable-by-variable basis [29, 30, 32]: they introduce a single variational distribution $q_{a_i}(x;\phi_{a_i})$ for each unique sampling address a_i, then form the guide by replacing all the original random draws, $x_i \sim f_{a_i}(\cdot \mid \eta_i)$, with draws from the corresponding variational distribution instead. This forms a global guide that maintains the stochastic dependency structure of the original program, such that the guide itself has stochastic support.

Our motivating insight is that this high-level approach has some fundamental limitations. Consider the simple example from Fig. 1. Here the variable `x` influences the program's control flow. This causes discontinuities that mean it is difficult to approximate its conditional density with a single variational approximation, especially if that variational approximation is restricted to a simple distribution class. Here the different possible paths are essentially working against each other, as what is helpful for the approximation of `x` on one path, is generally detrimental for the other.

A further complication occurs when the stochastic control flow of a program influences whether a variable exists at all. Here it can become extremely challenging to set up guides which are faithful to the dependency structure of previously sampled variables [28], as the set of variables we condition on is itself stochastic. Because of this, existing approaches typically rely on mean-field assumptions across paths [29–31]. However, this assumption is rarely reasonable given that the path typically strongly influences the distribution of individual variables.

Finally, creating a single unique variational approximation for each address also leads to challenging optimization problems: the same address can be present in multiple program paths, and the number of variables and their dependencies can vary between different paths.

4 Support Decomposition Variational Inference

We now introduce a novel VI approach to overcome the challenges mentioned in Sec. 3. We call our method *Support Decomposition Variational Inference* (SDVI), because the key design decision is the choice (and automatic construction) of a guide that takes the form of a mixture distribution over the set of possible paths a program can take. That is, rather than constructing a single guide with the same stochastic control flow as the original program and a separate variational approximation for

each *unique address*, we instead construct separate sub-guides with deterministic control flows for each *unique path*. These are then combined into an overall guide using the mixture distribution which maximizes the overall ELBO. As we will see, this alternative approach substantially simplifies the process of constructing effective guides and allows the full weight of the well-developed techniques for VI in the static support setting to be brought to bear on problems with stochastic support.

4.1 Decomposing Probabilistic Programs into Straight-Line Programs

As noted by, e.g., [33–35], all probabilistic programs can be reformulated as mixture distributions over *straight-line programs* (SLPs), which are sub-programs without any stochastic control flow. Building on our earlier notation, the constituent SLPs of a program correspond directly to the unique instances of the program path, A. Given the path, the set of variables making up the raw random draws in the program is fixed, along with their form and reference distribution; that is each SLP represents a probabilistic model with fixed variable typing and support.

Following the notation of Zhou et al. [35], we can apply an arbitrary fixed ordering on the set of SLPs in a program, such that we can uniquely define and index them using the set of possible addresses $\{A_k\}_{k=1}^K$, where K is countable (but potentially infinite). Each SLP A_k now corresponds to a particular sub-region, $\mathcal{X}_k$, of the raw random draw sample space, $\mathcal{X}$. These sub-regions are disjoint and their union is the full sample space. Unlike $\mathcal{X}$, each element in any given $\mathcal{X}_k$ has the same length n_k and is measurable with respect to the same reference measure. The unnormalized density for the kth SLP is now given by

$$\gamma_k(x_{1:n_k}) = \mathbb{I}[x_{1:n_k} \in \mathcal{X}_k]\, \gamma(x_{1:n_k}) = \mathbb{I}[x_{1:n_k} \in \mathcal{X}_k] \prod_{i=1}^{n_k} f_{A_k[i]}(x_i \mid \eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j \mid \phi_j), \quad (2)$$

and the unnormalized density function for the original program can be written as a simple sum of the individual SLP densities: $\gamma(x_{1:n_x}) = \sum_{k=1}^K \gamma_k(x_{1:n_x})$. The corresponding normalized conditional density can then be written as mixture distribution over the conditional distributions of the individual SLPs, with mixture weights given by their (normalized) local partition functions:

$$\pi(x) = \sum_{k=1}^K \pi(x \mid k)\, \pi(k) \;\text{ where }\; \pi(x \mid k) = \frac{\gamma_k(x)}{Z_k},\; \pi(k) = \frac{Z_k}{\sum_{\ell=1}^K Z_\ell},\; Z_k = \int_{\mathcal{X}_k} \gamma_k(x) dx. \quad (3)$$

4.2 Decomposing the Variational Family into Straight-Line Programs

The key idea behind SDVI is now to construct the guide using an analogous factorization to the SLP decomposition above. Precisely, we aim to learn variational approximations of the form

$$q(x; \phi, \lambda) = \sum_{k=1}^K q_k(x; \phi_k) q(k; \lambda) \quad (4)$$

where $q(k; \lambda)$ defines a categorical distribution over the indices of the SLPs, with support $k \in \{1, \ldots, K\}$; and $q_k(x; \phi_k)$ is the local guide of the kth SLP, with support $x \in \mathcal{X}_k$. Critically, as each $\mathcal{X}_k$ represents a fixed support, the local variational families q_k can be automatically constructed using standard techniques for static problems, as we discuss in Sec. 4.5.

Writing $\phi = \{\phi_k\}_{k=1}^K$, the KL divergence we wish to minimize for standard VI is now

$$\mathrm{KL}(q(x; \phi, \lambda) \parallel \pi(x)) = \mathbb{E}_{q(x;\phi,\lambda)} \left[\log q(x; \phi, \lambda) - \log \pi(x)\right], \quad (5)$$

which we call the *global KL divergence*. By standard reasoning, minimizing this is equivalent to maximizing the *global ELBO*

$$\mathcal{L}(\phi, \lambda) = \mathbb{E}_{q(x;\phi,\lambda)} \left[\log \gamma(x) - \log q(x; \phi, \lambda)\right] \quad (6)$$

which, as we show in App. A, can be rewritten as

$$\mathcal{L}(\phi, \lambda) = \mathbb{E}_{q(k;\lambda)} \left[\mathcal{L}_k(\phi_k) - \log q(k; \lambda)\right], \;\text{ where }\; \mathcal{L}_k(\phi_k) := \mathbb{E}_{q_k(x;\phi_k)} \left[\log \frac{\gamma_k(x)}{q_k(x; \phi_k)}\right] \quad (7)$$

is the term we refer to as the *local ELBO* for the kth SLP. Notice that each $\mathcal{L}_k(\phi_k)$ depends only on the parameter ϕ_k and the local SLP density γ_k; it is completely independent of $q(k; \lambda)$, the other SLPs, and $\phi_{k'}$ for $k' \neq k$. Thus, it follows from (7) that the inference problem for the whole program can be decomposed into independent 'local' inference problems for the component SLPs, along with establishing the mixture probabilities $q(k; \lambda)$. Furthermore, it turns out that the optimal $q(k; \lambda)$ is simply the softmax of $\mathcal{L}_1, \ldots, \mathcal{L}_K$, as shown by the following result.

Algorithm 1 Support Decomposition Variational Inference

Require: Target program γ, iteration budget T, minimum no. of SH candidates m

Extract SLPs $\{\gamma_k\}_{k=1}^{K}$ from γ and set $\mathcal{C} = \{1, \ldots, K\}$ ▷ Sec 4.3
Formulate guide q_k for each SLP and initialize parameters ϕ_k ▷ Sec 4.5
for $l = 1, \ldots, L = \lceil \log_2(K) - \log_2(m) + 1 \rceil$ **do**
  for $k \in \mathcal{C}$ **do**
    Perform $\lfloor T/L|\mathcal{C}| \rfloor$ optimization iterations of ϕ_k targeting $\mathcal{L}_{\text{surr},k}(\phi_k)$ ▷ Sec 4.5
    Estimate $\mathcal{L}_{\text{surr},k}(\phi_k)$ using Monte Carlo estimate of Eq. (11)
  end for
  Remove $\min(\lfloor |\mathcal{C}|/2 \rfloor, |\mathcal{C}| - m)$ SLPs from $\mathcal{C}$ with the lowest $\mathcal{L}_{\text{surr},k}(\phi_k)$ ▷ Sec. 4.4
end for
Truncate q_k outside of SLP support, $\mathcal{X}_k$, using Eq. (13)
Estimate each $\mathcal{L}_k(\phi_k)$ using Monte Carlo estimate of Eq. (7)
Calculate $q(k; \lambda)$ according to Eq. (8) and return $q(x; \phi, \lambda)$ as per Eq. (4)

Proposition 1. *Let $\mathcal{L}_1, \ldots, \mathcal{L}_K$ be the set of local ELBOs, defined as per (7), where K is countable but potentially not finite. If $0 < \sum_{k=1}^{K} \exp(\mathcal{L}_k) < \infty$, then the optimal corresponding $q(k; \lambda)$ in terms of the global ELBO (6) is given by*

$$q(k; \lambda) = \exp(\mathcal{L}_k) \,/\, \sum_{\ell=1}^{K} \exp(\mathcal{L}_\ell) \tag{8}$$

The proof of this result is given in App. A. Though each of the $\mathcal{L}_k$ terms here is itself intractable, they can be estimated efficiently and accurately by simple Monte Carlo. We can thus straightforwardly construct $q(k; \lambda)$ once we have learned our local variational approximations: noting that these two processes are separable, $q(k; \lambda)$ is not needed until *after* the individual q_k are trained.

4.3 Finding SLPs

We have just shown how we can solve the VI problem of a probabilistic program with stochastic support by reducing it to a set of *independent* and *simpler* VI problems, each concerning an SLP, a program with static support. However, we still need a mechanism to 'discover' the SLPs, i.e. extract the possible address paths from a program.

Here we first note that we only need to consider an SLP if it has a non-zero probability of being identified under forward simulation of the program, while ignoring conditioning statements. This hints at a cheap and simple discovery mechanism whereby we draw samples by forward simulation and take note of the unique paths that have been generated. Although this is a stochastic process that is not guaranteed to find all the SLPs, our experiments show that it can reliably identify all SLPs with *non-negligible posterior mass*. Moreover, we can also use this approach to discover SLPs in an *online manner*, whereby we seek new SLPs as our budget increases and we have scope to deal with them (see App. B for details). If necessary, more sophisticated methods could be used instead, such as those based on MCMC sampling [35] or static analysis of the program code [33, 44, 45].

4.4 Allocating Resources

Using the same amount of computational budget on each SLP is potentially wasteful, particularly if there is a large number of SLPs with insignificant marginal likelihoods. Therefore, we seek a scheme that allocates more computational resources to promising SLPs, making sure to exploit the fact that the different inference problems are trivially parallelizable.

To formalize this resource allocation problem, let T represent some fixed resource budget. Further, let t_k be the amount of this budget we spend on optimizing the kth SLP, such that $\sum_k t_k = T$ at the end of our training. Our ultimate aim is produce the maximum possible final global ELBO, which will be a function of $\phi_1(t_1), \ldots, \phi_K(t_K)$, where $\phi_k(t_k)$ denotes the value of ϕ_k achieved after allocating t_k resources to that SLP. By plugging the optimal mixture distribution $q(k; \lambda)$ from (8) into (6), we see that, after some rearranging, our resource allocation can be formulated as trying to maximize

$$\mathcal{L}(\phi, \lambda^*) = \log \sum_{k=1}^{K} \exp(\mathcal{L}_k(\phi_k(t_k))) \quad \text{s.t.} \sum_{k=1}^{K} t_k = T. \tag{9}$$

In practice, this is not a suitable objective for controlling our resource allocation directly, as it is still itself a random variable given $t_1, \ldots, t_K$, because the optimization procedure is stochastic. Moreover, we cannot consider its expectation, since the distribution of the $\phi_k(t_k)$ is unknown. However, it does provide insight into how we ideally would like to allocate resources: we want to allocate more resources to SLPs whose exponentiated ELBOs are significant. In particular, we can think of the 'reward' for allocating ϵ more resources to SLP k as $\exp(\mathcal{L}_k(\phi_k(t_k + \epsilon))) - \exp(\mathcal{L}_k(\phi_k(t_k)))$.

One could now, in principle, formulate the problem as a sequential decision making problem [46]. However, the diminishing nature of the rewards and the fact that they are highly unlikely to be sub-Gaussian, along with the need to allow choosing multiple arms at once for parallelization, mean that setting up such an approach which is effective in practice is likely to be quite challenging.

Instead, we propose a simple heuristic, based on the *Successive Halving* algorithm (SH) [47] (see App. B for a description), an approach commonly used for resource allocation in hyperparameter optimization (HO) [48, 49]. In standard SH, the final objective is to identify and train the single best candidate, whereas ours is to maximize the *sum* of all the local ELBOs. Despite this difference, the use of SH can still be justified by the fact that the distribution over $\exp(\mathcal{L}_k(\phi_k(\infty)))$ will typically be heavily concentrated to a small number of SLPs, often only a single one. Nonetheless, we make a small adaptation to the approach to stop over-focusing on a single SLP: we stop the halving process when a chosen number, $1 \leqslant m \leqslant K$, of the candidates are left, with $m = 1$ corresponding to standard SH. The minimum proportion of the budget allocated to any given candidate by this scheme is $1/(K\lceil \log_2 K - \log_2 m + 1 \rceil)$, so we can use m as a hyperparameter to control how evenly resources are allocated, with $m = K$ corresponding to uniform allocation. This approach is also helpful for parallelization, as we can set m equal to the number of available cores.

Putting everything together, a summary of our SDVI algorithm is given in Algo. 1. In App. B, we further show how this can be extended to an online variant of the approach, wherein we repeatedly run SH using the objective $\exp(\alpha \mathcal{L}_k(\phi_k(t_k)))/t_k$, where $0 < \alpha \leqslant 1$ is a hyperparameter, with smaller values of α encouraging more exploration.

4.5 Formulating and Training the Local Guides

In Algo. 1 we assume a mechanism to construct the *local guide* $q_k(x; \phi_k)$ for each SLP specified by path A_k. In many situations—notably when the program path is uniquely determined by the sampled values from discrete distributions—it is possible to construct guides q_k that are guaranteed to place support within the sub-region $\mathcal{X}_k$, which, in turn, allows us to use the reparameterized gradient estimator for the gradients of $\mathcal{L}_k(\phi_k)$. Many models encountered in practice, e.g. mixture models [13], have this property. In this case it is possible to eliminate all the variables which influence the control flow by conditioning, effectively setting them to constants; see App. C for further details.

In situations where we cannot easily construct a q_k which places support only within $\mathcal{X}_k$, we need to take care when training our guide. Recall that for path A_k the number of variables n_k and their type sequence is fixed, which allows us to construct an initial guide $\tilde{q}_k$ with correct dimensionality and variable typing. Let the support of this guide be denoted by $\mathcal{X}'_k = \text{supp}(\tilde{q}_k)$. In general, we will have $\mathcal{X}_k \subset \mathcal{X}'_k$, because the control flow in the program imposes additional constraints on each individual variable. Having constructed a guide with $\text{supp}(\tilde{q}_k) = \mathcal{X}'_k$, one might be tempted to optimize $\text{KL}(\tilde{q}_k(x; \phi_k) \parallel \pi(x \mid k))$, but we cannot guarantee the absolute continuity condition (i.e. $\tilde{q}_k(x; \phi_k) = 0$ if $\pi(x \mid k) = 0$), and so, the KL divergence may not be well-defined, giving an ELBO of $-\infty$. To alleviate this issue we create a new surrogate target density defined as

$$\tilde{\gamma}_k(x_{1:n_k}) := \gamma_k(x_{1:n_k}) + c\,\mathbb{I}[x_{1:n_k} \notin \mathcal{X}_k], \tag{10}$$

for a small positive, finite constant c and we optimize the corresponding surrogate ELBO

$$\mathcal{L}_{\text{surr},k}(\phi_k) := \mathbb{E}_{q_k(x;\phi_k)}\left[\log \tilde{\gamma}_k(x) - \log \tilde{q}_k(x; \phi_k)\right]. \tag{11}$$

We need to be careful to choose an appropriate c that is sufficiently small compared to the values of $\gamma_k(x)$ for $x \in \mathcal{X}_k$, which we ensure by setting c adaptively. During the SLP discovery phase (Line 1 in Algo. 1), we keep track of the smallest density value encountered so far, and call that d_{min}. We then set $c = 0.01 d_{min}$ to ensure that the density values for $\tilde{\gamma}_k(x)$ outside of $\mathcal{X}_k$ are significantly below the values of $\tilde{\gamma}_k(x)$ for $x \in \mathcal{X}_k$. Hence, optimizing (11) faithfully optimizes q_k to be a good approximation to $\gamma_k(x)$ while avoiding the issues of infinite ELBO values. While $\tilde{\gamma}_k$ is not a proper unnormalized density (it will in general not integrate to a finite value) this is not an issue in practice due to the mode-seeking behaviour of optimizing the ELBO.

Unfortunately, the bounds on the support of the SLP inevitably create a discontinuity in the objective. Thus, for fully unbiased gradients we need to use the score function estimator or some extension thereof. However, in some cases, the bias of the reparameterization gradient estimator may be sufficiently small to warrant its use. Note that $\mathcal{L}_{\text{surr},k}(\phi_k)$ retains the desirable property of the standard ELBO that, if the observations are conditionally independent given the latent variables, we can get unbiased estimates of the ELBO using minibatches of the full dataset [19, 22, 43].

Further we need to be careful when initializing $\tilde{q}_k$ as we require it to place sufficient probability mass within $\mathcal{X}_k$ to provide a suitable training signal. To ensure this, we initialize q_k by minimizing the *forward* KL divergence between the prior density of the kth SLP and $\tilde{q}_k(x;\phi_k)$

$$\mathrm{KL}(\pi_{prior,k}(x) \parallel \tilde{q}_k(x;\phi_k)) \propto \mathbb{E}_{\pi_{prior}(x)}\left[-\mathbb{I}[x \in \mathcal{X}_k] \log \tilde{q}_k(x;\phi_k)\right] \tag{12}$$

where $\pi_{prior}(x_{1:n_x}) := \prod_{i=1}^{n_x} f_{a_i}(x_i \mid \eta_i)$. This objective can be optimized via stochastic gradient descent (cf. App. C). Note that, for the purpose of initialization, we are targeting the prior, and thus we do not have to resort to expensive schemes to estimate the gradients which are necessary if one aims to minimize the forward KL targeting the posterior [50].

Finally, we can truncate the trained $\tilde{q}_k$ to implicitly define a new distribution that satisfies the hard constraint $\mathrm{supp}(q_k) = \mathcal{X}_k$. This requires us to check whether specific raw random draws $x'_{1:n_k}$ are valid for the path A_k, i.e. whether $\mathbb{I}\left[x'_{1:n_k} \in \mathcal{X}_k\right]$. We can do this by simply executing the program with fixed draws set to $x'_{1:n_k}$ and then noting that the program terminates and follows the address path A_k if, and only if, $x'_{1:n_k} \in \mathcal{X}_k$. Thus to truncate q_k we update it according to

$$q_k(x;\phi_k) = \frac{\tilde{q}_k(x;\phi_k)\mathbb{I}\left[x \in \mathcal{X}_k\right]}{\tilde{Z}_k}, \quad \text{where} \quad \tilde{Z}_k = \int_{\mathcal{X}'_k} \tilde{q}_k(x;\phi_k)\mathbb{I}\left[x \in \mathcal{X}_k\right] dx. \tag{13}$$

Hence, q_k is implicitly defined as the output of a rejection sampler with $\tilde{q}_k$ as a proposal. This, in turn, allows us to evaluate the local ELBO in (7) and thus also the optimal $q(k;\lambda)$.

5 Related Work

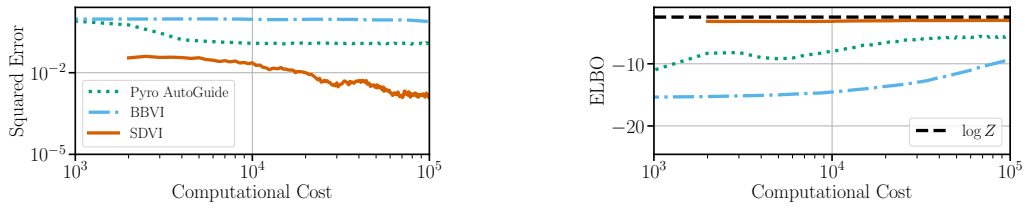
The vast majority of prior work on deriving automated VI algorithms focuses on the setting of static support [20, 23, 51–55]. Of particular note, [28, 56, 57] also consider using variational families that do not match the dependency structure of the original problem, but they still require static support.

The Divide-Conquer-Combine (DCC) algorithm [35] also exploits the breakdown of the program density into individual SLPs. However, Zhou et al. [35] mainly focused on local inference algorithms that are sampling based, especially MCMC. As we showed in Sec. 4 unique challenges and opportunities arise when we consider the breakdown from a variational perspective. One of the most practical differences is that SDVI only requires (exponentiated) ELBOs to be estimated for each SLP, rather than marginal likelihoods. The former can typically be estimated substantially more accurately for a given budget, allowing SDVI to scale better to high dimensional problems (see Sec. 6.2).

Our work is situated in the larger context automated inference for universal PPSs. Other popular approaches include particle-based methods [6, 11, 58–60] and MCMC approaches with automated proposals [1, 61–63]. Some work has also looked to perform *amortized* inference over a range of possible datasets [16, 56, 57], often by training a proposal that is similar to a variational approximation.

6 Experiments

To make SDVI easily accessible to practitioners we have implemented it in Pyro (code is made available in the Supplementary Material). The first baseline we consider, **Pyro AutoGuide**, uses the `AutoNormalMessenger` class to automatically generate a guide, and trains it with Pyro's built-in tools for VI (`http://pyro.ai/examples/svi_part_i.html`). As an additional VI baseline, we also implement a custom guide for each model which uses the variable-by-variable scheme outlined in Sec. 3, in combination with the score function gradient estimator; we refer to this baseline as **BBVI**. For SDVI, we run SH until there are 10 active SLPs left (i.e. $m = 10$ in Algo. 1) and parallelize the computation across 10 cores. We further construct each local guide distribution q_k as a mean-field normal. The specific configurations for each method for each experiment are provided in App. D.



(a) Squared error.

(b) ELBOs.

Figure 2: Results for the model in § 6.1. Computational cost is measured in the number of likelihood evaluations. For each metric we show the mean and standard deviation over 10 runs. a) Squared error between the true SLP weights and the estimated SLP weights. b) Evidence Lower Bounds (ELBOs) for the variational algorithms, dashed line indicates the analytic log marginal likelihood.

6.1 Program with Normal Distributions

We use our first experiment to further clarify the failure modes of existing VI approaches. We consider an extension of the model from Fig. 1 to contain more SLPs. The full model is

$$\begin{aligned} u &\sim \mathcal{N}(0, 5^2), \\ x &\sim \mathcal{N}(z, 1), \\ y &\sim \mathcal{N}(x, 1). \end{aligned} \qquad \text{where} \qquad z = \begin{cases} 0, & \text{if } u \in (-\infty, -4] \\ K, & \text{if } u \in (-5+K, -4+K] \text{ for } K = 1, \dots, 8 \\ 9, & \text{if } u \in (4, \infty) \end{cases} \tag{14}$$

We assume we have observed $y = 2$. The results in Fig. 2 demonstrate that SDVI is able to overcome the limitations of the other variational approaches. BBVI and Pyro AutoGuide both use the same guide in this model; BBVI uses the score function gradient estimator for training, whereas Pyro AutoGuide uses the reparameterized gradient estimator. This difference results in different posterior approximations for the different baselines. The BBVI guide tends to place all its mass on a single SLP and then provides a suitable approximation for only that one SLP, ignoring all the others. This explains the large standard deviations for the ELBO values in Fig. 2b as the ELBOs in different SLPs will converge to drastically different values. For Pyro AutoGuide the biased gradient estimates will train the variational approximation for variable u to be close to the prior $\mathcal{N}(0, 5^2)$. SDVI is able to avoid the shortcomings of the baselines as it provides an overall better posterior approximation leading to larger ELBO values, i.e. lower KL divergences to the true posterior, and a more accurate weighting of the different SLPs (Fig. 2a).

6.2 Infinite Gaussian Mixture Model

Our next model is a Gaussian Mixture Model (GMM) with a Poisson prior on the number of clusters:

$$K \sim \text{Poisson}(9) + 1; \quad u_k \sim \mathcal{N}(\mathbf{0}, 10\,\mathrm{I}) \text{ for } k = 1, \dots, K; \quad y \sim \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mu_k, 0.1\,\mathrm{I}),$$

where I is the $D \times D$ identity matrix and $\mathbf{0}$ is a D dimensional vector of zeros (we set $D = 100$). A similar model was considered in Zhou et al. [35] but with $D = 1$ instead of $D = 100$. We generate a dataset of 1250 observations with $K = 5$. To compare and evaluate the different algorithms, we hold out 250 data points as a test dataset to compute the log posterior predictive density (LPPD).

The Pyro AutoGuide baseline from the previous experiment is not applicable here since it assumes all latent variables are continuous. To provide a further baseline, we have also implemented **DCC** [35] in Pyro with Random-walk lightweight Metropolis-Hastings (RMH) [62] as a local inference algorithm. We chose DCC in particular because it also exploits the same breakdown into SLPs, so comparing against DCC is an opportunity to highlight the benefits of using a VI method.

In this model, the observations are assumed to be conditionally independent given the latent variables, thus enabling SDVI to work on subsets of the whole dataset [19, 22]. Specifically, we run SDVI on a model which samples a random minibatch of size $B = 100$ at each iteration and then scales the likelihood by the factor N/B, where $N = 1000$ is the size of the full dataset; we refer to this setup as Stochastic SDVI (S-SDVI). Furthermore, for this model SDVI is able to directly construct valid local guides q_k (using the mechanism for models branching on discrete variables outlined in Sec. 4.5) and therefore (S-)SDVI can use the reparameterized gradient estimator.

Table 1: Log posterior predictive densitiy (LPPD), ELBO, and maximum a posteriori (MAP) estimate for K for GMM model. Mean and standard deviation for LPPD and ELBO computed over 5 runs.

Method	LPPD ($\uparrow, \times 10^3$)	ELBO ($\uparrow, \times 10^3$)	MAP K
DCC	-9842.90 ± 3904.57	N/A	14, 11, 16, 14, 15
BBVI	-2217.07 ± 146.31	-8770.55 ± 544.95	25, 25, 25, 25, 25
SDVI	$\mathbf{32.84 \pm 0.02}$	$\mathbf{128.76 \pm 0.17}$	5, 5, 6, 6, 5
S-SDVI	$\mathbf{32.80 \pm 0.02}$	$\mathbf{128.63 \pm 0.22}$	5, 5, 6, 5, 6

Table 1 shows that SDVI and S-SDVI significantly outperform the baselines, yielding a several orders of magnitude larger posterior predictive density and providing the only reasonable predictions for the numbers of clusters. We found that for the runs in which (S-)SDVI outputs a MAP estimate of $K = 6$, the local guide q_k corresponding to the SLP with 5 components had failed to correctly identify all the clusters in the data. This leads to lower values of the local ELBO compared to other SLPs resulting in a smaller weight for this SLP in the final posterior. BBVI and DCC struggle with this model due to the high-dimensional parameter space. DCC's local inference algorithm, RMH, only updates one variable at a time which results in slow mixing times for the MCMC chains. For practical reasons, we had to cap the maximum number of clusters in the BBVI guide at 25 (cf. App. D).

6.3 Inferring Gaussian Process Kernels

For our final experiment, we consider the problem of inferring the kernel structure of a Gaussian Process (GP). Following [64, 65], we place a prior over kernel functions using a probabilistic context-free grammar (PCFG) . We consider the squared exponential (SE), rational quadratic (RQ), periodic (PER), and linear (LIN) base kernels, and use the production rules

$$\mathcal{K} \rightarrow \text{SE} \mid \text{RQ} \mid \text{PER} \mid \text{LIN} \mid \mathcal{K} \times \mathcal{K} \mid \mathcal{K} + \mathcal{K}.$$

Sampling from the PCFG is implemented with a recursive probabilistic program that uses samples from a categorical distribution to decide which production rule in the PCFG should be applied. In addition to the kernel structure, we also perform inference over the kernel hyperparameters for each base kernel and the observation noise; we place an inverse-gamma prior on each base kernel hyperparameter and a half-normal prior on the observation noise. We further assume a normal likelihood function and marginalize out the latent GP. Additional model details are in App. D. We apply this model to a dataset of monthly counts of international airline passengers [66], withholding the last 10 % of all observations as a test dataset.

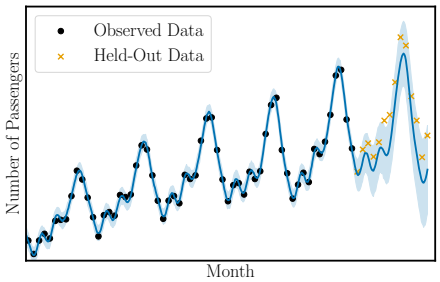


Figure 3: Posterior predictions of the GP for SDVI, shaded regions indicate 2 standard deviations that are computed from 100 posterior samples.

For SDVI we can construct local valid proposals q_k using the mechanism for models with discrete branching outlined in Sec 4.5. Hence, in each SLP the local guide q_k provides a posterior approximation over the kernel hyperparameters and the observation noise; the posterior distribution over kernel structures is implicitly defined through the mixture distribution over program paths. Table 2 shows that SDVI provides higher LPPD values, and is also able to achieve a higher final ELBO value compared to BBVI. Fig. 3 shows the posterior predictions for the SDVI run with the median LPPD score. SDVI is able to provide qualitatively reasonable predictions, as the predictions follow the periodic trend in the observed data.

Table 2: Final log posterior predictive densitiy (LPPD) and ELBO for GP model. Shown are mean and standard deviation computed over 5 runs.

Method	LPPD ($\uparrow$)	ELBO ($\uparrow$)
DCC	-58.92 ± 32.47	N/A
BBVI	-18.82 ± 1.20	-48.48 ± 0.33
SDVI	$\mathbf{2.05 \pm 3.30}$	$\mathbf{34.53 \pm 21.42}$

7 Conclusion

We have presented SDVI and shown that it is able overcome the limitations of existing VI approaches for programs with stochastic support by using a novel guide structure that breaks the program down into SLPs with fixed support, rather than matching the original stochastic control flow. We further found that it was able to outperform the state-of-the-art non-VI baseline DCC in our experiments.

References

[1] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76:1–32, January 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. URL https://doi.org/10.18637/jss.v076.i01.

[2] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, April 2016. ISSN 2376-5992. doi: 10.7717/peerj-cs.55. URL https://peerj.com/articles/cs-55.

[3] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and Implementation of Probabilistic Programming Language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages - IFL 2016*, pages 1–12, Leuven, Belgium, 2016. ACM Press. ISBN 978-1-4503-4767-9. doi: 10.1145/3064899.3064910. URL http://dl.acm.org/citation.cfm?doid=3064899.3064910.

[4] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 220–229, Arlington, Virginia, USA, July 2008. AUAI Press. ISBN 978-0-9749039-4-1.

[5] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019. ISSN 1533-7928. URL http://jmlr.org/papers/v20/18-403.html.

[6] Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: A Language for Flexible Probabilistic Inference. In *International Conference on Artificial Intelligence and Statistics*, pages 1682–1690. PMLR, March 2018. URL http://proceedings.mlr.press/v84/ge18b.html.

[7] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 221–236, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314642. URL https://doi.org/10.1145/3314221.3314642.

[8] Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: A higher-order probabilistic programming platform with programmable inference. *arXiv:1404.0099 [cs, stat]*, March 2014. URL http://arxiv.org/abs/1404.0099.

[9] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic Inference by Program Transformation in Hakaru (System Description). In Oleg Kiselyov and Andy King, editors, *Functional and Logic Programming*, volume 9613, pages 62–79. Springer International Publishing, Cham, 2016. ISBN 978-3-319-29603-6 978-3-319-29604-3. doi: 10.1007/978-3-319-29604-3_5. URL http://link.springer.com/10.1007/978-3-319-29604-3_5.

[10] Noah D Goodman and Andreas Stuhlmüller. WebPPL - probabilistic programming for the web, 2014. URL http://webppl.org/.

[11] Lawrence M. Murray and Thomas B. Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43, 2018. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2018.10.013. URL https://www.sciencedirect.com/science/article/pii/S1367578818301202.

[12] Christopher Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, May 1999. ISBN 978-0-262-13360-9.

[13] Sylvia. Richardson and Peter J. Green. On Bayesian Analysis of Mixtures with an Unknown Number of Components (with discussion). *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 59(4):731–792, 1997. ISSN 1467-9868. doi: 10.1111/1467-9868.00095. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00095.

[14] Fredrik Ronquist, Jan Kudlicka, Viktor Senderov, Johannes Borgström, Nicolas Lartillot, Daniel Lundén, Lawrence Murray, Thomas B. Schön, and David Broman. Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *bioRxiv*, page 2020.06.16.154443, December 2020. doi: 10.1101/2020.06.16.154443. URL https://www.biorxiv.org/content/10.1101/2020.06.16.154443v4.

[15] Atilim Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip Torr, Victor Lee, Kyle Cranmer, Prabhat, and Frank Wood. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, pages 1–24, New York, NY, USA, November 2019. Association for Computing Machinery. ISBN 978-1-4503-6229-0. doi: 10.1145/3295500.3356180. URL https://doi.org/10.1145/3295500.3356180.

[16] Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference Compilation and Universal Probabilistic Programming. In *Artificial Intelligence and Statistics*, pages 1338–1348. PMLR, April 2017. URL http://proceedings.mlr.press/v54/le17a.html.

[17] Bradley Gram-Hansen, Christian Schröder de Witt, Tom Rainforth, Philip H. S. Torr, Yee Whye Teh, and Atılım Güneş Baydin. Hijacking Malaria Simulators with Probabilistic Programming. *arXiv:1905.12432 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1905.12432.

[18] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, July 2017. ISSN 0162-1459. URL https://www.tandfonline.com/doi/full/10.1080/01621459.2017.1285773.

[19] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic Variational Inference in Stan. *arXiv:1506.03431 [stat]*, June 2015. URL http://arxiv.org/abs/1506.03431.

[20] Abhinav Agrawal, Daniel R Sheldon, and Justin Domke. Advances in black-box VI: Normalizing flows, importance weighting, and optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17358–17369. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/c91e3483cf4f90057d02aa492d2b25b1-Paper.pdf.

[21] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

[22] Matthew D Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, 14(5), May 2013.

[23] Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538. PMLR, June 2015. URL https://proceedings.mlr.press/v37/rezende15.html.

[24] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1312.6114.

[25] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC Press, May 2011. ISBN 978-1-4200-7942-5.

[26] N. Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning disentangled representations with semi-supervised deep generative models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5927–5937. Curran Associates, Inc., 2017.

[27] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[28] Stefan Webb, Adam Golinski, Robert Zinkov, N. Siddharth, Tom Rainforth, Yee Whye Teh, and Frank Wood. Faithful Inversion of Generative Models for Effective Amortized Inference. *arXiv:1712.00287 [cs, stat]*, November 2018. URL http://arxiv.org/abs/1712.00287.

[29] David Wingate and Theophane Weber. Automated Variational Inference in Probabilistic Programming. *arXiv:1301.1299 [cs, stat]*, January 2013. URL http://arxiv.org/abs/1301.1299.

[30] Timothy Brooks Paige. *Automatic Inference for Higher-Order Probabilistic Programs*. http://purl.org/dc/dcmitype/Text, University of Oxford, 2016. URL https://ora.ox.ac.uk/objects/uuid:d912c4de-4b08-4729-aa19-766413735e2a.

[31] Jan-Willem Vandemeent, Brooks Paige, David Tolpin, and Frank Wood. Black-Box Policy Search with Probabilistic Programs. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 1195–1204. PMLR, May 2016. URL https://proceedings.mlr.press/v51/vandemeent16.html.

[32] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. *arXiv:1809.10756 [cs, stat]*, September 2018. URL http://arxiv.org/abs/1809.10756.

[33] Arun Chaganty, Aditya Nori, and Sriram Rajamani. Efficiently Sampling Probabilistic Programs via Program Analysis. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 153–160. PMLR, April 2013. URL https://proceedings.mlr.press/v31/chaganty13a.html.

[34] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 447–458, New York, NY, USA, June 2013. Association for Computing Machinery. ISBN 978-1-4503-2014-6. doi: 10.1145/2491956.2462179. URL https://doi.org/10.1145/2491956.2462179.

[35] Yuan Zhou, Hongseok Yang, Yee Whye Teh, and Tom Rainforth. Divide, Conquer, and Combine: A New Inference Strategy for Probabilistic Programs with Stochastic Support. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11534–11545. PMLR, November 2020. URL https://proceedings.mlr.press/v119/zhou20e.html.

[36] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Future of Software Engineering Proceedings*, FOSE 2014, pages 167–181, New York, NY, USA, May 2014. Association for Computing Machinery. ISBN 978-1-4503-2865-4. doi: 10.1145/2593882.2593900. URL https://doi.org/10.1145/2593882.2593900.

[37] Thomas William Gamlen Rainforth. *Automating Inference, Learning, and Design Using Probabilistic Programming*. http://purl.org/dc/dcmitype/Text, University of Oxford, 2017. URL https://ora.ox.ac.uk/objects/uuid:e276f3b4-ff1d-44bf-9d67-013f68ce81f0.

[38] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[39] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo Gradient Estimation in Machine Learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020. ISSN 1533-7928. URL http://jmlr.org/papers/v21/19-346.html.

[40] Jack P. C. Kleijnen and Reuven Y. Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, February 1996. ISSN 0377-2217. doi: 10.1016/0377-2217(95)00107-7. URL https://www.sciencedirect.com/science/article/pii/0377221795001077.

[41] Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Language*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[42] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv:1401.4082 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1401.4082.

[43] Michalis Titsias and Miguel Lázaro-Gredilla. Doubly Stochastic Variational Bayes for non-Conjugate Inference. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1971–1979. PMLR, June 2014. URL https://proceedings.mlr.press/v32/titsias14.html.

[44] Aditya V Nori, Chung-Kil Hur, Sriram K Rajamani, and Selva Samuel. R2: An Efficient MCMC Sampler for Probabilistic Programs. *AAAI Conference on Artificial Intelligence (AAAI)*, page 7, 2015.

[45] Raven Beutner, Luke Ong, and Fabian Zaiser. Guaranteed bounds for posterior inference in universal probabilistic programming. *PLDI 2022: International Conference on Programming Language Design and Implementation*, 2022.

[46] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. doi: 10.1017/9781108571401.

[47] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pages 1238–1246. PMLR, 2013.

[48] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL http://jmlr.org/papers/v18/16-558.html.

[49] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

[50] Christian Naesseth, Fredrik Lindsten, and David Blei. Markovian score climbing: Variational inference with kl(p || q). In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15499–15510. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/b20706935de35bbe643733f856d9e5d6-Paper.pdf.

[51] Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 814–822. PMLR, April 2014. URL https://proceedings.mlr.press/v33/ranganath14.html.

[52] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18(1):430–474, jan 2017. ISSN 1532-4435.

[53] Luca Ambrogioni, Kate Lin, Emily Fertig, Sharad Vikram, Max Hinne, Dave Moore, and Marcel van Gerven. Automatic structured variational inference. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 676–684. PMLR, March 2021. URL https://proceedings.mlr.press/v130/ambrogioni21a.html.

[54] Akash Kumar Dhaka, Alejandro Catalina, Manushi Welandawe, Michael Riis Andersen, Jonathan Huggins, and Aki Vehtari. Challenges and Opportunities in High-dimensional Variational Inference. *arXiv:2103.01085 null*, March 2021. URL http://arxiv.org/abs/2103.01085.

[55] Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization Gradient for Non-differentiable Models. In *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://papers.nips.cc/paper/2018/hash/b096577e264d1ebd6b41041f392eec23-Abstract.html.

[56] Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2016.

[57] Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. *Advances in neural information processing systems*, 26, 2013.

[58] Brooks Paige and Frank Wood. A Compilation Target for Probabilistic Programming Languages. *arXiv:1403.0504 [cs, stat]*, July 2014. URL http://arxiv.org/abs/1403.0504.

[59] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A New Approach to Probabilistic Programming Inference. In *Artificial Intelligence and Statistics*, pages 1024–1032. PMLR, April 2014. URL http://proceedings.mlr.press/v33/wood14.html.

[60] Tom Rainforth, Christian Naesseth, Fredrik Lindsten, Brooks Paige, Jan-Willem Vandemeent, Arnaud Doucet, and Frank Wood. Interacting particle markov chain monte carlo. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2616–2625, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/rainforth16.html.

[61] David Wingate, Andreas Stuhlmueller, and Noah Goodman. Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 770–778. JMLR Workshop and Conference Proceedings, June 2011. URL https://proceedings.mlr.press/v15/wingate11a.html.

[62] Tuan Anh Le. *Inference for Higher Order Probabilistic Programs*. Master's Thesis, University of Oxford, 2016.

[63] Carol Mak, Fabian Zaiser, and Luke Ong. Nonparametric Hamiltonian Monte Carlo. In *Proceedings of the 38th International Conference on Machine Learning*, pages 7336–7347. PMLR, July 2021. URL https://proceedings.mlr.press/v139/mak21a.html.

[64] David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure Discovery in Nonparametric Regression through Compositional Kernel Search. In *International Conference on Machine Learning*, pages 1166–1174. PMLR, May 2013. URL http://proceedings.mlr.press/v28/duvenaud13.html.

[65] David Janz, Brooks Paige, Tom Rainforth, Jan-Willem van de Meent, and Frank Wood. Probabilistic structure discovery in time series data. *arXiv preprint arXiv:1611.06863*, 2016.

[66] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2015. ISBN 9781118674925. URL https://books.google.co.uk/books?id=rNt5CgAAQBAJ.

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

[68] L. Martino, V. Elvira, D. Luengo, and J. Corander. Layered adaptive importance sampling. *Statistics and Computing*, 27(3):599–623, May 2017. ISSN 1573-1375. doi: 10.1007/s11222-016-9642-5. URL https://doi.org/10.1007/s11222-016-9642-5.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Limitations are discussed in relevant sections throughout the paper.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] The paper provides a generic variational inference algorithm for probabilistic models with stochastic support. As our contributions are methodological and our experiments do not contain any personalized data, we believe that this paper does not introduce any fundamentally new risks.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Proposition 1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 6 and Appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code in supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]