




Fast and Correct Gradient-Based Optimisation for Probabilistic Programming via Smoothing

Basim Khajwal¹, C.-H. Luke Ong^{1,2}, and Dominik Wagner¹ 

¹ University of Oxford, Oxford, United Kingdom

² NTU, Singapore, Singapore

Abstract. We study the foundations of variational inference, which frames posterior inference as an optimisation problem, for probabilistic programming. The dominant approach for optimisation in practice is stochastic gradient descent. In particular, a variant using the so-called reparameterisation gradient estimator exhibits fast convergence in a traditional statistics setting. Unfortunately, discontinuities, which are readily expressible in programming languages, can compromise the correctness of this approach. We consider a simple (higher-order, probabilistic) programming language with conditionals, and we endow our language with both a measurable and a *smoothed* (approximate) value semantics. We present type systems which establish technical pre-conditions. Thus we can prove stochastic gradient descent with the reparameterisation gradient estimator to be correct when applied to the smoothed problem. Besides, we can solve the original problem up to any error tolerance by choosing an accuracy coefficient suitably. Empirically we demonstrate that our approach has a similar convergence as a key competitor, but is simpler, faster, and attains orders of magnitude reduction in work-normalised variance.

Keywords: probabilistic programming · variational inference · reparameterisation gradient · value semantics · type systems.

1 Introduction

Probabilistic programming is a programming paradigm which has the vision to make statistical methods, in particular Bayesian inference, accessible to a wide audience. This is achieved by a separation of concerns: the domain experts wishing to gain statistical insights focus on modelling, whilst the inference is performed automatically. (In some recent systems [4,9] users can improve efficiency by writing their own inference code.)

In essence, probabilistic programming languages extend more traditional programming languages with constructs such as **score** or **observe** (as well as **sample**) to define the prior $p(\mathbf{z})$ and likelihood $p(\mathbf{x} \mid \mathbf{z})$. The task of inference is to derive the posterior $p(\mathbf{z} \mid \mathbf{x})$, which is in principle governed by Bayes' law yet usually intractable.

Whilst the paradigm was originally conceived in the context of statistics and Bayesian machine learning, probabilistic programming has in recent years

proven to be a very fruitful subject for the programming language community. Researchers have made significant theoretical contributions such as underpinning languages with rigorous (categorical) semantics [35,34,15,37,12,10] and investigating the correctness of inference algorithms [16,7,22]. The latter were mostly designed in the context of “traditional” statistics and features such as conditionals, which are ubiquitous in programming, pose a major challenge for correctness.

Inference algorithms broadly fall into two categories: Markov chain Monte Carlo (MCMC), which yields a sequence of samples asymptotically approaching the true posterior, and variational inference.

Variational Inference. In the variational inference approach to Bayesian statistics [40,30,5,6], the problem of approximating difficult-to-compute posterior probability distributions is transformed to an optimisation problem. The idea is to approximate the posterior probability $p(\mathbf{z} \mid \mathbf{x})$ using a family of “simpler” densities $q_{\boldsymbol{\theta}}(\mathbf{z})$ over the latent variables \mathbf{z} , parameterised by $\boldsymbol{\theta}$. The optimisation problem is then to find the parameter $\boldsymbol{\theta}^*$ such that $q_{\boldsymbol{\theta}^*}(\mathbf{z})$ is “closest” to the true posterior $p(\mathbf{z} \mid \mathbf{x})$. Since the variational family may not contain the true posterior, $q_{\boldsymbol{\theta}^*}$ is an approximation in general. In practice, variational inference has proven to yield good approximations much faster than MCMC.

Formally, the idea is captured by minimising the *KL-divergence* [30,5] between the variational approximation and the true posterior. This is equivalent to maximising the ELBO function, which only depends on the joint distribution $p(\mathbf{x}, \mathbf{z})$ and *not* the posterior, which we seek to infer after all:

$$\text{ELBO}_{\boldsymbol{\theta}} := \mathbb{E}_{\mathbf{z} \sim q_{\boldsymbol{\theta}}(\mathbf{z})} [\log p(\mathbf{x}, \mathbf{z}) - \log q_{\boldsymbol{\theta}}(\mathbf{z})] \quad (1)$$

Gradient Based Optimisation. In practice, variants of *Stochastic Gradient Descent (SGD)* are frequently employed to solve optimisation problems of the following form: $\text{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{s} \sim q(\mathbf{s})} [f(\boldsymbol{\theta}, \mathbf{s})]$. In its simplest version, SGD follows Monte Carlo estimates of the gradient in each step:

$$\boldsymbol{\theta}_{k+1} := \boldsymbol{\theta}_k - \gamma_k \cdot \underbrace{\frac{1}{N} \sum_{i=1}^N \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_k, \mathbf{s}_k^{(i)})}_{\text{gradient estimator}}$$

where $\mathbf{s}_k^{(i)} \sim q(\mathbf{s}_k^{(i)})$ and γ_k is the *step size*.

For the correctness of SGD it is crucial that the estimation of the gradient is *unbiased*, i.e. correct in expectation:

$$\mathbb{E}_{\mathbf{s}^{(1)}, \dots, \mathbf{s}^{(N)} \sim q} \left[\frac{1}{N} \sum_{i=1}^N \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{s}^{(i)}) \right] = \nabla_{\boldsymbol{\theta}} \mathbb{E}_{\mathbf{s} \sim q(\mathbf{s})} [f(\boldsymbol{\theta}, \mathbf{s})]$$

This property, which is about commuting differentiation and integration, can be established by the dominated convergence theorem [21, Theorem 6.28].

Note that we cannot directly estimate the gradient of the ELBO in Eq. (1) with Monte Carlo because the distribution w.r.t. which the expectation is taken also depends on the parameters. However, the so-called *log-derivative trick* can be used to derive an unbiased estimate, which is known as the *Score* or *REINFORCE* estimator [31,38,27,28].

Reparameterisation Gradient. Whilst the score estimator has the virtue of being very widely applicable, it unfortunately suffers from high variance, which can cause SGD to yield very poor results³.

The *reparameterisation gradient estimator*—the dominant approach in variational inference—reparameterises the latent variable \mathbf{z} in terms of a base random variable \mathbf{s} (viewed as the entropy source) via a diffeomorphic transformation ϕ_{θ} , such as a location-scale transformation or cumulative distribution function. For example, if the distribution of the latent variable z is a Gaussian $\mathcal{N}(z \mid \mu, \sigma^2)$ with parameters $\theta = \{\mu, \sigma\}$ then the location-scale transformation using the standard normal as the base distribution gives rise to the reparameterisation

$$z \sim \mathcal{N}(z \mid \mu, \sigma^2) \iff z = \phi_{\mu, \sigma}(s), \quad s \sim \mathcal{N}(0, 1). \quad (2)$$

where $\phi_{\mu, \sigma}(s) := s \cdot \sigma + \mu$. The key advantage of this setup (often called “reparameterisation trick” [20,36,32]) is that we have removed the dependency on θ from the distribution w.r.t. which the expectation is taken. Therefore, we can now differentiate (by backpropagation) with respect to the parameters θ of the variational distributions using Monte Carlo simulation with draws from the base distribution \mathbf{s} . Thus, succinctly, we have

$$\nabla_{\theta} \mathbb{E}_{\mathbf{z} \sim q(\mathbf{z})}[f(\theta, \mathbf{z})] = \nabla_{\theta} \mathbb{E}_{\mathbf{s} \sim q(\mathbf{s})}[f(\theta, \phi_{\theta}(\mathbf{s}))] = \mathbb{E}_{\mathbf{s} \sim q(\mathbf{s})}[\nabla_{\theta} f(\theta, \phi_{\theta}(\mathbf{s}))]$$

The main benefit of the reparameterisation gradient estimator is that it has a significantly lower variance than the score estimator, resulting in faster convergence.

Bias of the Reparameterisation Gradient. Unfortunately, the reparameterisation gradient estimator is biased for non-differentiable models [23], which are readily expressible in programming languages with conditionals:

Example 1. The counterexample in [23, Proposition 2], where the objective function is the ELBO for a non-differentiable model, can be simplified to

$$f(\theta, s) = -0.5 \cdot \theta^2 + \begin{cases} 0 & \text{if } s + \theta < 0 \\ 1 & \text{otherwise} \end{cases}$$

Observe that (see Fig. 1a):

$$\nabla_{\theta} \mathbb{E}_{s \sim \mathcal{N}(0,1)} [f(\theta, s)] = -\theta + \mathcal{N}(-\theta \mid 0, 1) \neq -\theta = \mathbb{E}_{s \sim \mathcal{N}(0,1)} [\nabla_{\theta} f(\theta, s)]$$

³ see e.g. Fig. 5a or [28]

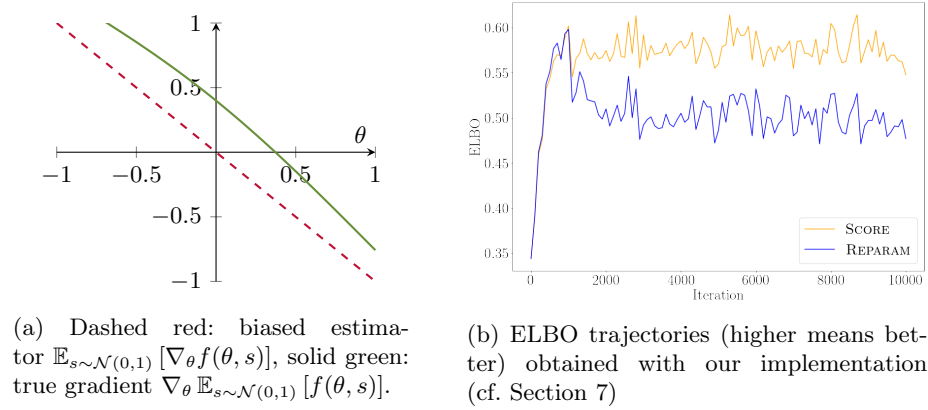


Fig. 1: Bias of the reparameterisation gradient estimator for Example 1.

Crucially *this may compromise convergence to critical points or maximisers*: even if we can find a point where the gradient estimator vanishes, it may not be a critical point (let alone optimum) of the original optimisation problem (cf. Fig. 1b)

Informal Approach

As our starting point we take a variant of the simply typed lambda calculus with reals, conditionals and a sampling construct. We abstract the optimisation of the ELBO to the following generic optimisation problem

$$\operatorname{argmin}_{\theta} \mathbb{E}_{s \sim \mathcal{D}} [\llbracket M \rrbracket(\theta, s)] \quad (3)$$

where $\llbracket M \rrbracket$ is the value function [7,26] of a program M and \mathcal{D} is independent of the parameters θ and it is determined by the distributions from which M samples. Owing to the presence of conditionals, the function $\llbracket M \rrbracket$ may not be continuous, let alone differentiable.

Example 1 can be expressed as

$$(\lambda z. -0.5 \cdot \theta^2 + (\text{if } z < 0 \text{ then } 0 \text{ else } 1)) (\text{sample } \mathcal{N} + \theta)$$

Our approach is based on a denotational semantics $\llbracket (-) \rrbracket_{\eta}$ (for accuracy coefficient $\eta > 0$) of programs in the (new) cartesian closed category \mathbf{VectFr} , which generalises smooth manifolds and extends Frölicher spaces (see e.g. [13,33]) with a vector space structure.

Intuitively, we replace the Heaviside step-function usually arising in the interpretation of conditionals by smooth approximations. In particular, we interpret the conditional of Example 1 as

$$\llbracket \text{if } s + \theta < 0 \text{ then } 0 \text{ else } 1 \rrbracket_{\eta}(\theta, s) := \sigma_{\eta}(s + \theta)$$

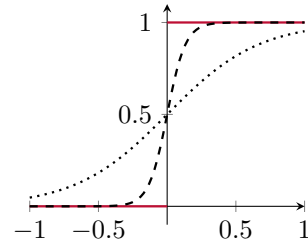


Fig. 2: (Logistic) sigmoid function σ_{η} (dotted: $\eta = \frac{1}{3}$, dashed: $\eta = \frac{1}{15}$) and the Heaviside step function (red, solid).

where σ_η is a smooth function. For instance we can choose $\sigma_\eta(x) := \sigma(\frac{x}{\eta})$ where $\sigma(x) := \frac{1}{1+\exp(-x)}$ is the (logistic) sigmoid function (cf. Fig. 2). Thus, the program M is interpreted by a smooth function $\llbracket M \rrbracket_\eta$, for which the reparameterisation gradient may be estimated unbiasedly. Therefore, we apply stochastic gradient descent on the smoothed program.

Contributions

The high-level contribution of this paper is laying a theoretical foundation for correct yet efficient (variational) inference for probabilistic programming. We employ a smoothed interpretation of programs to obtain unbiased (reparameterisation) gradient estimators and establish technical pre-conditions by type systems. In more detail:

1. We present a simple (higher-order) programming language with conditionals. We employ *trace types* to capture precisely the samples drawn in a fully eager call-by-value evaluation strategy.
2. We endow our language with both a (measurable) denotational value semantics and a smoothed (hence approximate) value semantics. For the latter we furnish a categorical model based on Frölicher spaces.
3. We develop type systems enforcing vital technical pre-conditions: unbiasedness of the reparameterisation gradient estimator and the correctness of stochastic gradient descent, as well as the uniform convergence of the smoothing to the original problem. Thus, our smoothing approach in principle yields correct solutions up to arbitrary error tolerances.
4. We conduct an empirical evaluation demonstrating that our approach exhibits a similar convergence to an unbiased correction of the reparameterised gradient estimator by [23] – our main baseline. However our estimator is simpler and more efficient: it is faster and attains orders of magnitude reduction in work-normalised variance.

Outline. In the next section we introduce a simple higher-order probabilistic programming language, its denotational value semantics and operational semantics; Optimisation Problem 1 is then stated. Section 3 is devoted to a smoothed denotational value semantics, and we state the Smooth Optimisation Problem 2. In Sections 4 and 5 we develop annotation based type systems enforcing the correctness of SGD and the convergence of the smoothing, respectively. Related work is briefly discussed in Section 6 before we present the results of our empirical evaluation in Section 7. We conclude in Section 8 and discuss future directions.

Notation. We use the following conventions: bold font for vectors and lists, ++ for concatenation of lists, $\nabla_{\boldsymbol{\theta}}$ for gradients (w.r.t. $\boldsymbol{\theta}$), $[\phi]$ for the Iverson bracket of a predicate ϕ and calligraphic font for distributions, in particular \mathcal{N} for normal distributions. Besides, we highlight noteworthy items using **red**.

2 A Simple Programming Language

In this section, we introduce our programming language, which is the simply-typed lambda calculus with reals, augmented with conditionals and sampling from continuous distributions.

2.1 Syntax

The *raw terms* of the programming language are defined by the grammar:

$$M ::= x \mid \theta_i \mid \underline{r} \mid \pm \mid : \mid \underline{-} \mid \underline{-}^{-1} \mid \underline{\exp} \mid \underline{\log} \\ \mid \text{if } M < 0 \text{ then } M \text{ else } M \mid \text{sample}_{\mathcal{D}} \mid \lambda x. M \mid M M$$

where x and θ_i respectively range over (denumerable collections of) *variables* and *parameters*, $r \in \mathbb{R}$, and \mathcal{D} is a probability distribution over \mathbb{R} (potentially with a support which is a strict subset of \mathbb{R}). As is customary we use infix, postfix and prefix notation: $M \pm N$ (addition), $M : N$ (multiplication), $M \underline{-}^{-1}$ (inverse), and $\underline{-}M$ (numeric negation). We frequently omit the underline to reduce clutter.

Example 2 (Encoding the ELBO for Variational Inference). We consider the example used by [23] in their Prop. 2 to prove the biasedness of the reparameterisation gradient. (In Example 1 we discussed a simplified version thereof.) The density is

$$p(z) := \mathcal{N}(z \mid 0, 1) \cdot \begin{cases} \mathcal{N}(0 \mid -2, 1) & \text{if } z < 0 \\ \mathcal{N}(0 \mid 5, 1) & \text{otherwise} \end{cases}$$

and they use a variational family with density $q_{\theta}(z) := \mathcal{N}(z \mid \theta, 1)$, which is reparameterised using a standard normal noise distribution and transformation $s \mapsto s + \theta$.

First, we define an auxiliary term for the pdf of normals with mean m and standard derivation s :

$$N \equiv \lambda x, m, s. (\sqrt{2\pi} \cdot s)^{-1} \cdot \underline{\exp} \left(\underline{-0.5} \cdot ((x + (-m)) \cdot s^{-1})^2 \right)$$

Then, we can define

$$M \equiv \underbrace{\left(\lambda z. \underline{\log} (N z \underline{0} \underline{1}) + \left(\text{if } z < 0 \text{ then } \underline{\log} (N \underline{0} \underline{-2} \underline{1}) \text{ else } \underline{\log} (N \underline{0} \underline{5} \underline{1}) \right) \right)}_{\log p} - \underbrace{\left(\underline{\log} (N z \theta \underline{1}) \right)}_{\log q_{\theta}} \left(\text{sample}_{\mathcal{N}} + \theta \right)$$

2.2 A Basic Trace-Based Type System

Types are generated from *base types* (R and $R_{>0}$, the reals and positive reals) and *trace types* (typically Σ , which is a finite list of probability distributions)

as well as by a *trace-based function space* constructor of the form $\tau \bullet \Sigma \rightarrow \tau'$. Formally types are defined by the following grammar:

trace types	$\Sigma ::= [\mathcal{D}_1, \dots, \mathcal{D}_n] \quad n \geq 0$
base types	$\iota ::= R \mid R_{>0}$
safe types	$\sigma ::= \iota \mid \sigma \bullet [] \rightarrow \sigma$
types	$\tau ::= \iota \mid \tau \bullet \Sigma \rightarrow \tau$

where \mathcal{D}_i are probability distributions. Intuitively a trace type is a description of the space of execution traces of a probabilistic program. Using trace types, a distinctive feature of our type system is that a program's type precisely characterises the space of its possible execution traces [24]. We use list concatenation notation $++$ for trace types, and the shorthand $\tau_1 \rightarrow \tau_2$ for function types of the form $\tau_1 \bullet [] \rightarrow \tau_2$. Intuitively, a term has type $\tau \bullet \Sigma \rightarrow \tau'$ if, when given a value of type τ , it reduces to a value of type τ' using all the samples in Σ .

Dual context *typing judgements* of the form, $\Gamma \mid \Sigma \vdash M : \tau$, are defined in Fig. 3b, where $\Gamma = x_1 : \tau_1, \dots, x_n : \tau_n, \theta_1 : \tau'_1, \dots, \theta_m : \tau'_m$ is a finite map describing a set of variable-type and parameter-type bindings; and the trace type Σ precisely captures the distributions from which samples are drawn in a (fully eager) call-by-value evaluation of the term M .

The subtyping of types, as defined in Fig. 3a, is essentially standard; for contexts, we define $\Gamma \sqsubseteq \Gamma'$ if for every $x : \tau$ in Γ there exists $x : \tau'$ in Γ' such that $\tau' \sqsubseteq \tau$.

Trace types are unique [18]:

Lemma 1. *If $\Gamma \mid \Sigma \vdash M : \tau$ and $\Gamma \mid \Sigma' \vdash M : \tau'$ then $\Sigma = \Sigma'$.*

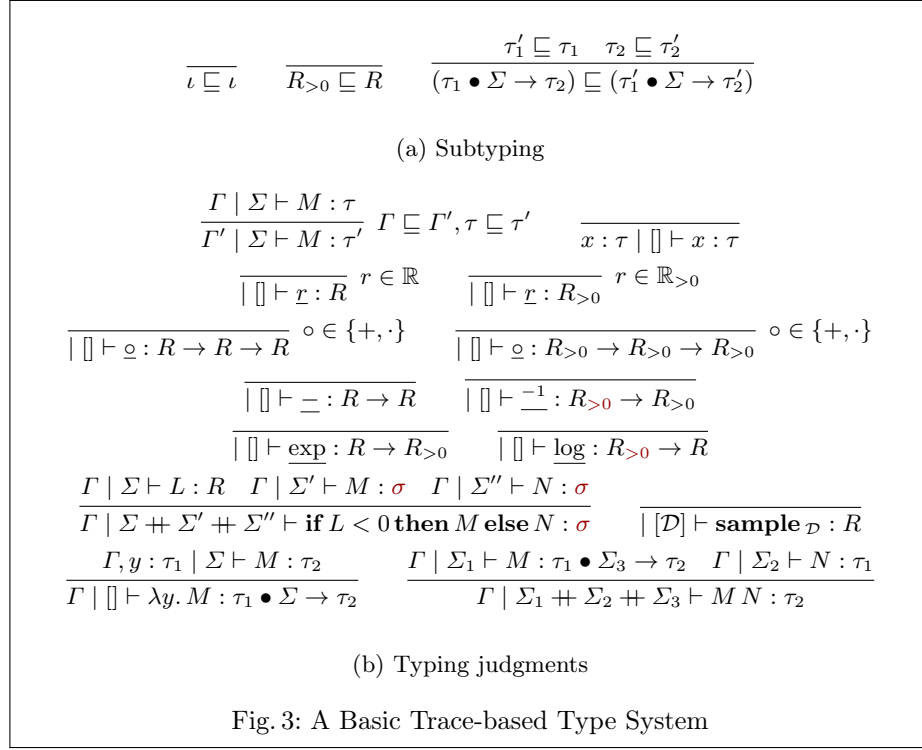
A term has *safe type* σ if it does not contain **sample** _{\mathcal{D}} or σ is a base type. Thus, perhaps slightly confusingly, we have $[\mathcal{D}] \vdash \mathbf{sample}_{\mathcal{D}} : R$, and R is considered a safe type. Note that we use the metavariable σ to denote safe types.

Conditionals. The branches of conditionals must have a safe type. Otherwise it would not be clear how to type terms such as

$$\begin{aligned} M &\equiv \mathbf{if} \ x < 0 \ \mathbf{then} \ (\lambda x. \mathbf{sample}_{\mathcal{N}}) \ \mathbf{else} \ (\lambda x. \mathbf{sample}_{\mathcal{E}} + \mathbf{sample}_{\mathcal{E}}) \\ N &\equiv (\lambda f. f (f \ \mathbf{sample}_{\mathcal{N}})) \ M \end{aligned}$$

because the branches draw a different number of samples from different distributions, and have types $R \bullet [\mathcal{N}] \rightarrow R$ and $R \bullet [\mathcal{E}, \mathcal{E}] \rightarrow R$, respectively. However, for $M' \equiv \mathbf{if} \ x < 0 \ \mathbf{then} \ \mathbf{sample}_{\mathcal{N}} \ \mathbf{else} \ \mathbf{sample}_{\mathcal{E}} + \mathbf{sample}_{\mathcal{E}}$ we can (safely) type

$$\begin{aligned} x : R \mid [\mathcal{N}, \mathcal{E}, \mathcal{E}] \vdash M' : R \\ \mid [] \vdash \lambda x. M' : R \bullet [\mathcal{N}, \mathcal{E}, \mathcal{E}] \rightarrow R \\ \mid [\mathcal{N}, \mathcal{N}, \mathcal{E}, \mathcal{E}, \mathcal{N}, \mathcal{E}, \mathcal{E}] \vdash (\lambda f. f (f \ \mathbf{sample}_{\mathcal{N}})) (\lambda x. M') : R \end{aligned}$$



Example 3. Consider the following terms:

$$L \equiv \lambda x. \mathbf{sample}_{\mathcal{N}} + \mathbf{sample}_{\mathcal{N}}$$

$$M \equiv \mathbf{if} x < 0 \mathbf{then} (\lambda y. y + y) \mathbf{sample}_{\mathcal{N}} \mathbf{else} (\mathbf{sample}_{\mathcal{N}} + \mathbf{sample}_{\mathcal{N}})$$

We can derive the following typing judgements:

$$\boxed{\vdash} L : R_{>0} \bullet [\mathcal{N}, \mathcal{N}] \rightarrow R$$

$$x : R_{>0} \mid [\mathcal{N}, \mathcal{N}, \mathcal{N}] \vdash M : R$$

$$\boxed{\vdash} \lambda x. M : R_{>0} \bullet [\mathcal{N}, \mathcal{N}, \mathcal{N}] \rightarrow R$$

$$\mid [\mathcal{N}, \mathcal{N}, \mathcal{N}] \vdash (\lambda x. M) \mathbf{sample}_{\mathcal{N}} : R$$

$$\mid [\mathcal{N}, \mathcal{N}] \vdash (\lambda f. f(f 0)) (\lambda x. \mathbf{sample}_{\mathcal{N}}) : R$$

Note that $\mathbf{if} x < 0 \mathbf{then} (\lambda x. \mathbf{sample}_{\mathcal{N}}) \mathbf{else} (\lambda x. x)$ is *not* typable.

2.3 Denotational Value Semantics

Next, we endow our language with a (measurable) value semantics. It is well-known that the category of measurable spaces and measurable functions is not cartesian-closed [1], which means that there is no interpretation of the lambda

calculus as measurable functions. These difficulties led [14] to develop the category **QBS** of *quasi-Borel spaces*. Notably, morphisms can be combined piecewisely, which we need for conditionals.

We interpret our programming language in the category **QBS** of quasi-Borel spaces. Types are interpreted as follows:

$$\begin{aligned} \llbracket R \rrbracket &:= (\mathbb{R}, M_{\mathbb{R}}) & \llbracket R_{>0} \rrbracket &:= (\mathbb{R}_{>0}, M_{\mathbb{R}_{>0}}) & \llbracket [\mathcal{D}_1, \dots, \mathcal{D}_n] \rrbracket &:= (\mathbb{R}, M_{\mathbb{R}})^n \\ \llbracket \tau_1 \bullet \Sigma \rightarrow \tau_2 \rrbracket &:= \llbracket \tau_1 \rrbracket \times \llbracket \Sigma \rrbracket \Rightarrow \llbracket \tau_2 \rrbracket \end{aligned}$$

where $M_{\mathbb{R}}$ is the set of measurable functions $\mathbb{R} \rightarrow \mathbb{R}$; similarly for $M_{\mathbb{R}_{>0}}$. (As for trace types, we use list notation (and list concatenation) for traces.)

We first define a handy helper function for interpreting application. For $f : \llbracket \Gamma \rrbracket \times \mathbb{R}^{n_1} \Rightarrow \llbracket \tau_1 \bullet \Sigma_3 \rightarrow \tau_2 \rrbracket$ and $g : \llbracket \Gamma \rrbracket \times \mathbb{R}^{n_2} \Rightarrow \llbracket \tau_1 \rrbracket$ define

$$\begin{aligned} f @ g &: \llbracket \Gamma \rrbracket \times \mathbb{R}^{n_1+n_2+|\Sigma_3|} \Rightarrow \llbracket \tau_2 \rrbracket \\ (\gamma, \mathbf{s}_1 \uplus \mathbf{s}_2 \uplus \mathbf{s}_3) &\mapsto f(\gamma, \mathbf{s}_1)(g(\gamma, \mathbf{s}_2), \mathbf{s}_3) \quad \mathbf{s}_1 \in \mathbb{R}^{n_1}, \mathbf{s}_2 \in \mathbb{R}^{n_2}, \mathbf{s}_3 \in \mathbb{R}^{|\Sigma_3|} \end{aligned}$$

We interpret terms-in-context, $\llbracket \Gamma \mid \Sigma \vdash M : \tau \rrbracket : \llbracket \Gamma \rrbracket \times \llbracket \Sigma \rrbracket \rightarrow \llbracket \tau \rrbracket$, as follows:

$$\begin{aligned} \llbracket \Gamma \mid [\mathcal{D}] \vdash \mathbf{sample}_{\mathcal{D}} : R \rrbracket(\gamma, [s]) &:= s \\ \llbracket \Gamma \mid [] \vdash \lambda y. M : \tau_1 \bullet \Sigma \rightarrow \tau_2 \rrbracket(\gamma, []) &:= \\ & (v, \mathbf{s}) \in \llbracket \tau_1 \rrbracket \times \llbracket \Sigma \rrbracket \mapsto \llbracket \Gamma, x : \tau_1 \mid \Sigma \vdash M : \tau_2 \rrbracket((\gamma, v), \mathbf{s}) \\ \llbracket \Gamma \mid \Sigma_1 \uplus \Sigma_2 \uplus \Sigma_3 \vdash M N : \tau \rrbracket &:= \\ & \llbracket \Gamma \mid \Sigma_1 \vdash M : \tau_1 \bullet \Sigma_3 \rightarrow \tau_2 \rrbracket @ \llbracket \Gamma \mid \Sigma_2 \vdash N : \tau_1 \rrbracket \\ \llbracket \Gamma \mid \Sigma_1 \uplus \Sigma_2 \uplus \Sigma_3 \vdash \mathbf{if} L < 0 \mathbf{then} M \mathbf{else} N : \tau \rrbracket(\gamma, \mathbf{s}_1 \uplus \mathbf{s}_2 \uplus \mathbf{s}_3) &:= \\ & \begin{cases} \llbracket \Gamma \mid \Sigma_2 \vdash M : \tau \rrbracket(\gamma, \mathbf{s}_2) & \text{if } \llbracket \Gamma \mid \Sigma_1 \vdash L : R \rrbracket(\gamma, \mathbf{s}_1) < 0 \\ \llbracket \Gamma \mid \Sigma_3 \vdash N : \tau \rrbracket(\gamma, \mathbf{s}_3) & \text{otherwise} \end{cases} \end{aligned}$$

It is not difficult to see that this interpretation of terms-in-context is well-defined and total. For the conditional clause, we may assume that the trace type and the trace are presented as partitions $\Sigma_1 \uplus \Sigma_2 \uplus \Sigma_3$ and $\mathbf{s}_1 \uplus \mathbf{s}_2 \uplus \mathbf{s}_3$ respectively. This is justified because it follows from the judgement $\Gamma \mid \Sigma_1 \uplus \Sigma_2 \uplus \Sigma_3 \vdash \mathbf{if} L < 0 \mathbf{then} M \mathbf{else} N : \tau$ that $\Gamma \mid \Sigma_1 \vdash L : R$, $\Gamma \mid \Sigma_2 \vdash M : \sigma$ and $\Gamma \mid \Sigma_3 \vdash N : \sigma$ are provable; and we know that each of Σ_1, Σ_2 and Σ_3 is unique, thanks to Lemma 1; their respective lengths then determine the partition $\mathbf{s}_1 \uplus \mathbf{s}_2 \uplus \mathbf{s}_3$. Similarly for the application clause, the components Σ_1 and Σ_2 are determined by Lemma 1, and Σ_3 by the type of M .

2.4 Relation to Operational Semantics

We can also endow our language with a big-step CBV sampling-based semantics similar to [7,26], as defined in [18, Fig. 6]. We write $M \Downarrow_w^{\mathbf{s}} V$ to mean that M reduces to value V , which is a real constant or an abstraction, using the execution trace \mathbf{s} and accumulating weight w .

Based on this, we can define the *value*- and *weight*-functions:

$$\text{value}_M(\mathbf{s}) := \begin{cases} V & \text{if } M \Downarrow_w^{\mathbf{s}} V \\ \text{undef} & \text{otherwise} \end{cases} \quad \text{weight}_M(\mathbf{s}) := \begin{cases} w & \text{if } M \Downarrow_w^{\mathbf{s}} V \\ 0 & \text{otherwise} \end{cases}$$

Our semantics is a bit non-standard in that for conditionals we evaluate both branches eagerly. The technical advantage is that for every (closed) term-in-context, $\mid [\mathcal{D}_1, \dots, \mathcal{D}_n] \vdash M : \iota$, M reduces to a (unique) value using exactly the traces of the length encoded in the typing, i.e., n .

So in this sense, the operational semantics is “total”: there is no divergence. Notice that there is no partiality caused by partial primitives such as $1/x$, thanks to the typing.

Moreover there is a simple connection to our denotational value semantics:

Proposition 1. *Let $\mid [\mathcal{D}_1, \dots, \mathcal{D}_n] \vdash M : \iota$. Then*

1. $\text{dom}(\text{value}_M) = \mathbb{R}^n$
2. $\llbracket M \rrbracket = \text{value}_M$
3. $\text{weight}_M(\mathbf{s}) = \prod_{j=1}^n \text{pdf}_{\mathcal{D}_j}(s_j)$

2.5 Problem Statement

We are finally ready to formally state our optimisation problem:

Problem 1. Optimisation

Given: term-in-context, $\theta_1 : \iota_1, \dots, \theta_m : \iota_m \mid [\mathcal{D}_1, \dots, \mathcal{D}_n] \vdash M : R$

Find: $\text{argmin}_{\theta} \mathbb{E}_{s_1 \sim \mathcal{D}_1, \dots, s_n \sim \mathcal{D}_n} [\llbracket M \rrbracket(\theta, \mathbf{s})]$

3 Smoothed Denotational Value Semantics

Now we turn to our smoothed denotational value semantics, which we use to avoid the bias in the reparameterisation gradient estimator. It is parameterised by a family of smooth functions $\sigma_\eta : \mathbb{R} \rightarrow [0, 1]$. Intuitively, we replace the Heaviside step-function arising in the interpretation of conditionals by smooth approximations (cf. Fig. 2). In particular, conditionals **if** $z < 0$ **then** $\underline{0}$ **else** $\underline{1}$ are interpreted as $z \mapsto \sigma_\eta(z)$ rather than $[z \geq 0]$ (using Iverson brackets).

Our primary example is $\sigma_\eta(x) := \sigma(\frac{x}{\eta})$, where σ is the (logistic) sigmoid $\sigma(x) := \frac{1}{1 + \exp(-x)}$, see Fig. 2. Whilst at this stage no further properties other than smoothness are required, we will later need to restrict σ_η to have good properties, in particular to convergence to the Heaviside step function.

As a categorical model we propose *vector Frölicher spaces* **VectFr**, which (to our knowledge) is a new construction, affording a simple and direct interpretation of the smoothed conditionals.

3.1 Frölicher Spaces

We recall the definition of Frölicher spaces, which generalise smooth spaces⁴: A *Frölicher space* is a triple $(X, \mathcal{C}_X, \mathcal{F}_X)$ where X is a set, $\mathcal{C}_X \subseteq \mathbf{Set}(\mathbb{R}, X)$ is a set of *curves* and $\mathcal{F}_X \subseteq \mathbf{Set}(X, \mathbb{R})$ is a set of *functionals*, satisfying

1. if $c \in \mathcal{C}_X$ and $f \in \mathcal{F}_X$ then $f \circ c \in C^\infty(\mathbb{R}, \mathbb{R})$
2. if $c : \mathbb{R} \rightarrow X$ such that for all $f \in \mathcal{F}_X$, $f \circ c \in C^\infty(\mathbb{R}, \mathbb{R})$ then $c \in \mathcal{C}_X$
3. if $f : X \rightarrow \mathbb{R}$ such that for all $c \in \mathcal{C}_X$, $f \circ c \in C^\infty(\mathbb{R}, \mathbb{R})$ then $f \in \mathcal{F}_X$.

A *morphism* between Frölicher spaces $(X, \mathcal{C}_X, \mathcal{F}_X)$ and $(Y, \mathcal{C}_Y, \mathcal{F}_Y)$ is a map $\phi : X \rightarrow Y$ satisfying $f \circ \phi \circ c \in C^\infty(\mathbb{R}, \mathbb{R})$ for all $f \in \mathcal{F}_Y$ and $c \in \mathcal{C}_X$.

Frölicher spaces and their morphisms constitute a category \mathbf{Fr} , which is well-known to be cartesian closed [13,33].

3.2 Vector Frölicher Spaces

To interpret our programming language smoothly we would like to interpret conditionals as σ_η -weighted convex combinations of its branches:

$$\begin{aligned} \llbracket \text{if } L < 0 \text{ then } M \text{ else } N \rrbracket_\eta(\gamma, \mathbf{s}_1 \uparrow \mathbf{s}_2 \uparrow \mathbf{s}_3) := \\ \sigma_\eta(-\llbracket L \rrbracket_\eta(\gamma, \mathbf{s}_1)) \cdot \llbracket M \rrbracket_\eta(\gamma, \mathbf{s}_2) + \sigma_\eta(\llbracket L \rrbracket_\eta(\gamma, \mathbf{s}_1)) \cdot \llbracket N \rrbracket_\eta(\gamma, \mathbf{s}_3) \end{aligned} \quad (4)$$

By what we have discussed so far, this only makes sense if the branches have ground type because Frölicher spaces are not equipped with a vector space structure but we take weighted combinations of morphisms. In particular if $\phi_1, \phi_2 : X \rightarrow Y$ and $\alpha : X \rightarrow \mathbb{R}$ are morphisms then $\alpha \phi_1 + \phi_2$ ought to be a morphism too. Therefore, we enrich Frölicher spaces with an additional vector space structure:

Definition 1. An \mathbb{R} -vector Frölicher space is a Frölicher space $(X, \mathcal{C}_X, \mathcal{F}_X)$ such that X is an \mathbb{R} -vector space and whenever $c, c' \in \mathcal{C}_X$ and $\alpha \in C^\infty(\mathbb{R}, \mathbb{R})$ then $\alpha c + c' \in \mathcal{C}_X$ (defined pointwise).

A morphism between \mathbb{R} -vector Frölicher spaces is a morphism between Frölicher spaces, i.e. $\phi : (X, \mathcal{C}_X, \mathcal{F}_X) \rightarrow (Y, \mathcal{C}_Y, \mathcal{F}_Y)$ is a morphism if for all $c \in \mathcal{C}_X$ and $f \in \mathcal{F}_Y$, $f \circ \phi \circ c \in C^\infty(\mathbb{R}, \mathbb{R})$.

\mathbb{R} -vector Frölicher space and their morphisms constitute a category \mathbf{VectFr} . There is an evident forgetful functor fully faithfully embedding \mathbf{VectFr} in \mathbf{Fr} . Note that the above restriction is a bit stronger than requiring that \mathcal{C}_X is also a vector space. (α is not necessarily a constant.) The main benefit is the following, which is crucial for the interpretation of conditionals as in Eq. (4):

Lemma 2. If $\phi_1, \phi_2 \in \mathbf{VectFr}(X, Y)$ and $\alpha \in \mathbf{VectFr}(X, \mathbb{R})$ then $\alpha \phi_1 + \phi_2 \in \mathbf{VectFr}(X, Y)$ (defined pointwisely).

Proof. Suppose $c \in \mathcal{C}_X$ and $f \in \mathcal{F}_Y$. Then $(\alpha_1 \phi_1 + \phi_2) \circ c = (\alpha \circ c) \cdot (\phi_1 \circ c) + (\phi_2 \circ c) \in \mathcal{C}_Y$ (defined pointwisely) and the claim follows.

⁴ $C^\infty(\mathbb{R}, \mathbb{R})$ is the set of smooth functions $\mathbb{R} \rightarrow \mathbb{R}$

Similarly as for Frölicher spaces, if X is an \mathbb{R} -vector space then any $\mathcal{C} \subseteq \mathbf{Set}(X, \mathbb{R})$ generates a \mathbb{R} -vector Frölicher space $(X, \mathcal{C}_X, \mathcal{F}_X)$, where

$$\begin{aligned}\mathcal{F}_X &:= \{f : X \rightarrow \mathbb{R} \mid \forall c \in \mathcal{C}. f \circ c \in C^\infty(\mathbb{R}, \mathbb{R})\} \\ \tilde{\mathcal{C}}_X &:= \{c : \mathbb{R} \rightarrow X \mid \forall f \in \mathcal{F}_X. f \circ c \in C^\infty(\mathbb{R}, \mathbb{R})\} \\ \mathcal{C}_X &:= \left\{ \sum_{i=1}^n \alpha_i c_i \mid n \in \mathbb{N}, \forall i \leq n. \alpha_i \in C^\infty(\mathbb{R}, \mathbb{R}), c_i \in \tilde{\mathcal{C}}_X \right\}\end{aligned}$$

Having modified the notion of Frölicher spaces generated by a set of curves, the proof for cartesian closure carries over [18] and we conclude:

Proposition 2. *VectFr is cartesian closed.*

3.3 Smoothed Interpretation

We have now discussed all ingredients to interpret our language (smoothly) in the cartesian closed category **VectFr**. We call $\llbracket M \rrbracket_\eta$ the η -smoothing of $\llbracket M \rrbracket$ (or of M , by abuse of language). The interpretation is mostly standard and follows Section 2.3, except for the case for conditionals. The latter is given by Eq. (4), for which the additional vector space structure is required.

Finally, we can phrase a smoothed version of our Optimisation Problem 1:

Problem 2. η -Smoothed Optimisation

Given: term-in-context, $\theta_1 : \iota_1, \dots, \theta_m : \iota_m \mid [\mathcal{D}_1, \dots, \mathcal{D}_n] \vdash M : R$, and accuracy coefficient $\eta > 0$

Find: $\operatorname{argmin}_{\theta} \mathbb{E}_{s_1 \sim \mathcal{D}_1, \dots, s_n \sim \mathcal{D}_n} [\llbracket M \rrbracket_\eta(\theta, \mathbf{s})]$

4 Correctness of SGD for Smoothed Problem and Unbiasedness of the Reparameterisation Gradient

Next, we apply stochastic gradient descent (SGD) with the reparameterisation gradient estimator to the smoothed problem (for the batch size $N = 1$):

$$\boldsymbol{\theta}_{k+1} := \boldsymbol{\theta}_k - \gamma_k \cdot \nabla_{\theta} \llbracket M \rrbracket_\eta(\boldsymbol{\theta}_k, \mathbf{s}_k) \quad \mathbf{s}_k \sim \mathcal{D} \quad (5)$$

where $\boldsymbol{\theta} \mid [\mathbf{s} \sim \mathcal{D}] \vdash M : R$ (slightly abusing notation in the trace type).

A classical choice for the step-size sequence is $\gamma_k \in \Theta(1/k)$, which satisfies the so-called *Robbins-Monro* criterion:

$$\sum_{k \in \mathbb{N}} \gamma_k = \infty \quad \sum_{k \in \mathbb{N}} \gamma_k^2 < \infty \quad (6)$$

In this section we wish to establish the correctness of the SGD procedure applied to the smoothing Eq. (5).

4.1 Desiderata

First, we ought to take a step back and observe that the optimisation problems we are trying to solve can be ill-defined due to a failure of integrability: take $M \equiv (\lambda x. \overline{\exp}(x _ x)) \mathbf{sample}_{\mathcal{N}}$: we have $\mathbb{E}_{z \sim \mathcal{N}}[\llbracket M \rrbracket(z)] = \infty$, independently of parameters. Therefore, we aim to guarantee:

(SGD0) The optimisation problems (both smoothed and unsmoothed) are well-defined.

Since $\mathbb{E}[\llbracket M \rrbracket_{\eta}(\boldsymbol{\theta}, \mathbf{s})]$ (and $\mathbb{E}[\llbracket M \rrbracket(\boldsymbol{\theta}, \mathbf{s})]$) may not be a convex function in the parameters $\boldsymbol{\theta}$, we cannot hope to always find global optima. We seek instead *stationary points*, where the gradient w.r.t. the parameters $\boldsymbol{\theta}$ vanishes. The following results (whose proof is standard) provide sufficient conditions for the convergence of SGD to stationary points (see e.g. [3] or [2, Chapter 2]):

Proposition 3 (Convergence). *Suppose $(\gamma_k)_{k \in \mathbb{N}}$ satisfies the Robbins-Monro criterion Eq. (6) and $g(\boldsymbol{\theta}) := \mathbb{E}_{\mathbf{s}}[f(\boldsymbol{\theta}, \mathbf{s})]$ is well-defined. If $\Theta \subseteq \mathbb{R}^m$ satisfies*

(SGD1) *Unbiasedness: $\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{s}}[\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \mathbf{s})]$ for all $\boldsymbol{\theta} \in \Theta$*

(SGD2) *g is L -Lipschitz smooth on Θ for some $L > 0$:*

$$\|\nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}) - \nabla_{\boldsymbol{\theta}} g(\boldsymbol{\theta}')\| \leq L \cdot \|\boldsymbol{\theta} - \boldsymbol{\theta}'\| \quad \text{for all } \boldsymbol{\theta}, \boldsymbol{\theta}' \in \Theta$$

(SGD3) *Bounded Variance: $\sup_{\boldsymbol{\theta} \in \Theta} \mathbb{E}_{\mathbf{s}}[\|\nabla_{\boldsymbol{\theta}} f_k(\boldsymbol{\theta}, \mathbf{s})\|^2] < \infty$*

then $\inf_{i \in \mathbb{N}} \mathbb{E}[\|\nabla g(\boldsymbol{\theta}_i)\|^2] = 0$ or $\boldsymbol{\theta}_i \notin \Theta$ for some $i \in \mathbb{N}$.

Unbiasedness (SGD1) requires commuting differentiation and integration. The validity of this operation can be established by the dominated convergence theorem [21, Theorem 6.28], see [18]. To be applicable the partial derivatives of f w.r.t. the parameters need to be dominated uniformly by an integrable function. Formally:

Definition 2. *Let $f : \Theta \times \mathbb{R}^n \rightarrow \mathbb{R}$ and $g : \mathbb{R}^n \rightarrow \mathbb{R}$. We say that g uniformly dominates f if for all $(\boldsymbol{\theta}, \mathbf{s}) \in \Theta \times \mathbb{R}^n$, $|f(\boldsymbol{\theta}, \mathbf{s})| \leq g(\mathbf{s})$.*

Also note that for Lipschitz smoothness (SGD2) it suffices to uniformly bound the second-order partial derivatives.

In the remainder of this section we present two type systems which restrict the language to guarantee properties (SGD0) to (SGD3).

4.2 Piecewise Polynomials and Distributions with Finite Moments

As a first illustrative step we consider a type system \vdash_{poly} , which restricts terms to (piecewise) polynomials, and distributions with finite moments. Recall that a distribution \mathcal{D} has (all) *finite moments* if for all $p \in \mathbb{N}$, $\mathbb{E}_{s \sim \mathcal{D}}[|s|^p] < \infty$. Distributions with finite moments include the following commonly used distributions: normal, exponential, logistic and gamma distributions. A non-example is the Cauchy distribution, which famously does not even have an expectation.

Definition 3. For a distribution \mathcal{D} with finite moments, $f : \mathbb{R}^n \rightarrow \mathbb{R}$ has (all) finite moments if for all $p \in \mathbb{N}$, $\mathbb{E}_{\mathbf{s} \sim \mathcal{D}}[|f(\mathbf{s})|^p] < \infty$.

Functions with finite moments have good closure properties:

Lemma 3. If $f, g : \mathbb{R}^n \rightarrow \mathbb{R}$ have (all) finite moments so do $-f, f + g, f \cdot g$.

In particular, if a distribution has finite moments then polynomials do, too. Consequently, intuitively, it is sufficient to simply (the details are explicitly spelled out in [18]):

1. require that the distributions \mathcal{D} in the sample rule have finite moments:

$$\overline{|\mathcal{D}| \vdash_{\text{poly}} \text{sample}_{\mathcal{D}} : R} \quad \mathcal{D} \text{ has finite moments}$$

2. remove the rules for $\underline{-1}$, $\underline{\text{exp}}$ and $\underline{\text{log}}$ from the type system \vdash_{poly} .

Type Soundness I: Well-Definedness. Henceforth, we fix parameters $\theta_1 : \iota_1, \dots, \theta_m : \iota_m$. Intuitively, it is pretty obvious that $\llbracket M \rrbracket$ is a piecewise polynomial whenever $\theta \mid \Sigma \vdash_{\text{poly}} M : \iota$. Nonetheless, we prove the property formally to illustrate our proof technique, a variant of logical relations, employed throughout the rest of the paper.

We define a slightly stronger logical predicate $\mathcal{P}_\tau^{(n)}$ on $\Theta \times \mathbb{R}^n \rightarrow \llbracket \tau \rrbracket$, which allows us to obtain a uniform upper bound:

1. $f \in \mathcal{P}_\iota^{(n)}$ if f is uniformly dominated by a function with finite moments
2. $f \in \mathcal{P}_{\tau_1 \bullet \Sigma_3 \rightarrow \tau_2}^{(n)}$ if for all $n_2 \in \mathbb{N}$ and $g \in \mathcal{P}_{\tau_1}^{(n+n_2)}$, $f \odot g \in \mathcal{P}_{\tau_2}^{(n+n_2+|\Sigma_3|)}$

where for $f : \Theta \times \mathbb{R}^{n_1} \rightarrow \llbracket \tau_1 \bullet \Sigma_3 \rightarrow \tau_2 \rrbracket$ and $g : \Theta \times \mathbb{R}^{n_1+n_2} \rightarrow \llbracket \tau_1 \rrbracket$ we define

$$\begin{aligned} f \odot g : \Theta \times \mathbb{R}^{n_1+n_2+|\Sigma_3|} &\rightarrow \tau_2 \\ (\theta, \mathbf{s}_1 \uplus \mathbf{s}_2 \uplus \mathbf{s}_3) &\mapsto f(\theta, \mathbf{s}_1)(g(\theta, \mathbf{s}_1 \uplus \mathbf{s}_2), \mathbf{s}_3) \end{aligned}$$

Intuitively, g may depend on the samples in \mathbf{s}_2 (in addition to \mathbf{s}_1) and the function application may consume further samples \mathbf{s}_3 (as determined by the trace type Σ_3). By induction on safe types we prove the following result, which is important for conditionals:

Lemma 4. If $f \in \mathcal{P}_\iota^{(n)}$ and $g, h \in \mathcal{P}_\sigma^{(n)}$ then $[f(-) < 0] \cdot g + [f(-) \geq 0] \cdot h \in \mathcal{P}_\sigma^{(n)}$.

Proof. For base types it follows from Lemma 3. Hence, suppose σ has the form $\sigma_1 \bullet \llbracket \cdot \rrbracket \rightarrow \sigma_2$. Let $n_2 \in \mathbb{N}$ and $x \in \mathcal{P}_{\sigma_1}^{(n+n_2)}$. By definition, $(g \odot x), (h \odot x) \in \mathcal{P}_{\sigma_2}^{(n+n_2)}$. Let \widehat{f} be the extension (ignoring the additional samples) of f to $\Theta \times \mathbb{R}^{n+n_2} \rightarrow \mathbb{R}$. It is easy to see that also $\widehat{f} \in \mathcal{P}_\iota^{(n+n_2)}$. By the inductive hypothesis,

$$[\widehat{f}(-) < 0] \cdot (g \odot x) + [\widehat{f}(-) \geq 0] \cdot (h \odot x) \in \mathcal{P}_{\sigma_2}^{(n+n_2)}$$

Finally, by definition,

$$([f(-) < 0] \cdot g + [f(-) \geq 0] \cdot h) \odot x = [\widehat{f}(-) < 0] \cdot (g \odot x) + [\widehat{f}(-) \geq 0] \cdot (h \odot x)$$

Assumption 1 We assume that $\Theta \subseteq \llbracket \iota_1 \rrbracket \times \cdots \times \llbracket \iota_m \rrbracket$ is compact.

Lemma 5 (Fundamental). If $\theta, x_1 : \tau_1, \dots, x_\ell : \tau_\ell \mid \Sigma \vdash_{\text{poly}} M : \tau, n \in \mathbb{N}, \xi_1 \in \mathcal{P}_{\tau_1}^{(n)}, \dots, \xi_\ell \in \mathcal{P}_{\tau_\ell}^{(n)}$ then $\llbracket M \rrbracket * \langle \xi_1, \dots, \xi_\ell \rangle \in \mathcal{P}_\tau^{(n+|\Sigma|)}$, where

$$\begin{aligned} \llbracket M \rrbracket * \langle \xi_1, \dots, \xi_\ell \rangle : \Theta \times \mathbb{R}^{n+|\Sigma|} &\rightarrow \llbracket \tau \rrbracket \\ (\theta, \mathbf{s} \# \mathbf{s}') &\mapsto \llbracket M \rrbracket((\theta, \xi_1(\theta, \mathbf{s}), \dots, \xi_\ell(\theta, \mathbf{s})), \mathbf{s}') \end{aligned}$$

It is worth noting that, in contrast to more standard fundamental lemmas, here we need to capture the dependency of the free variables on some number n of further samples. E.g. in the context of $(\lambda x. x) \mathbf{sample}_{\mathcal{N}}$ the subterm x depends on a sample although this is not apparent if we consider x in isolation.

Lemma 5 is proven by structural induction [18]. The most interesting cases include: parameters, primitive operations and conditionals. In the case for parameters we exploit the compactness of Θ (Assumption 1). For primitive operations we note that as a consequence of Lemma 3 each $\mathcal{P}_\iota^{(n)}$ is closed under negation⁵, addition and multiplication. Finally, for conditionals we exploit Lemma 3.

Type Soundness II: Correctness of SGD. Next, we address the integrability for the *smoothed* problem as well as (SGD1) to (SGD3). We establish that not only $\llbracket M \rrbracket_\eta$ but also its partial derivatives up to order 2 are uniformly dominated by functions with finite moments. For this to possibly hold we require:

Assumption 2 For every $\eta > 0$,

$$\sup_{x \in \mathbb{R}} |\sigma_\eta(x)| < \infty \quad \sup_{x \in \mathbb{R}} |\sigma'_\eta(x)| < \infty \quad \sup_{x \in \mathbb{R}} |\sigma''_\eta(x)| < \infty$$

Note that, for example, the logistic sigmoid satisfies Assumption 2.

We can then prove a fundamental lemma similar to Lemma 5, *mutatis mutandis*, using a logical predicate in **VectFr**. We stipulate $f \in \mathcal{Q}_\iota^{(n)}$ if its partial derivatives up to order 2 are uniformly dominated by a function with finite moments. In addition to Lemma 3 we exploit standard rules for differentiation (such as the sum, product and chain rule) as well as Assumption 2. We conclude:

Proposition 4. If $\theta \mid \Sigma \vdash_{\text{poly}} M : R$ then the partial derivatives up to order 2 of $\llbracket M \rrbracket_\eta$ are uniformly dominated by a function with all finite moments.

Consequently, the Smoothed Optimisation Problem 2 is not only well-defined but, by the dominated convergence theorem [21, Theorem 6.28], the reparameterisation gradient estimator is unbiased. Furthermore, (SGD1) to (SGD3) are satisfied and SGD is correct.

Discussion. The type system \vdash_{poly} is simple yet guarantees correctness of SGD. However, it is somewhat restrictive; in particular, it does not allow the expression of many ELBOs arising in variational inference directly as they often have the form of logarithms of exponential terms (cf. Example 2).

⁵ for $\iota = R$

4.3 A Generic Type System with Annotations

Next, we present a generic type system with annotations. In Section 4.4 we give an instantiation to make \vdash_{poly} more permissible and in Section 5 we turn towards a different property: the uniform convergence of the smoothings.

Typing judgements have the form $\Gamma \mid \Sigma \vdash_{?} M : \tau$, where “?” indicates the property we aim to establish, and we annotate base types. Thus, types are generated from

trace types	$\Sigma ::= [s_1 \sim \mathcal{D}_1, \dots, s_n \sim \mathcal{D}_n]$
base types	$\iota ::= R \mid R_{>0}$
safe types	$\sigma ::= \iota^{\beta} \mid \sigma \bullet [] \rightarrow \sigma$
types	$\tau ::= \iota^{\alpha} \mid \tau \bullet \Sigma \rightarrow \tau$

Annotations are drawn from a set and may possibly be restricted for safe types. Secondly, the trace types are now annotated with variables, typically $\Sigma = [s_1 \sim \mathcal{D}_1, \dots, s_n \sim \mathcal{D}_n]$ where the variables s_j are pairwise distinct.

For the subtyping relation we can constrain the annotations at the base type level [18]; the extension to higher types is accomplished as before.

The typing rules have the same form but they are extended with the annotations on base types and side conditions possibly constraining them. For example, the rules for addition, exponentiation and sampling are modified as follows:

$$\frac{}{[] \vdash_{?} \pm : \iota^{\alpha_1} \rightarrow \iota^{\alpha_2} \rightarrow \iota^{\alpha}} \text{ (cond. Add)} \quad \frac{}{[] \vdash_{?} \underline{\text{exp}} : R^{\alpha} \rightarrow R_{>0}^{\alpha'}} \text{ (cond. Exp)}$$

$$\frac{}{[s_j \sim \mathcal{D}] \vdash_{?} \text{sample}_{\mathcal{D}} : R^{\alpha}} \text{ (cond. Sample)}$$

The rules for subtyping, variables, abstractions and applications do not need to be changed at all but they use annotated types instead of the types of Section 2.2.

$$\frac{\frac{\Gamma \mid \Sigma \vdash_{?} M : \tau}{\Gamma' \mid \Sigma \vdash_{?} M : \tau'} \quad \Gamma \sqsubseteq_{?} \Gamma', \tau \sqsubseteq_{?} \tau' \quad \frac{}{x : \tau \mid [] \vdash_{?} x : \tau}}{\Gamma, y : \tau_1 \mid \Sigma \vdash_{?} M : \tau_2} \quad \frac{\Gamma \mid \Sigma_2 \vdash_{?} M : \tau_1 \bullet \Sigma_3 \rightarrow \tau_2 \quad \Gamma \mid \Sigma_1 \vdash_{?} N : \tau_1}{\Gamma \mid \Sigma_1 \vdash_{?} \lambda y. M : \tau_1 \bullet \Sigma \rightarrow \tau_2} \quad \frac{}{\Gamma \mid \Sigma_1 \vdash_{?} MN : \tau_2}$$

The full type system is presented in [18].

\vdash_{poly} can be considered a special case of $\vdash_{?}$ whereby we use the singleton $*$ as annotations, a contradictory side condition (such as false) for the undesired primitives $\underline{\quad}^{-1}$, $\underline{\text{exp}}$ and $\underline{\text{log}}$, and use the side condition “ \mathcal{D} has finite moments” for sample as above.

Table 1 provides an overview of the type systems of this paper and their purpose. $\vdash_{?}$ and its instantiations refine the basic type system of Section 2.2 in the sense that if a term-in-context is provable in the annotated type system, then its erasure (i.e. erasure of the annotations of base types and distributions) is provable in the basic type system. This is straightforward to check.

Table 1: Overview of type systems in this paper.

property	Section	judgement	annotation
totality	Section 2.2	\vdash	–
correctness SGD	Section 4.2	\vdash_{poly}	none/*
	Section 4.4	\vdash_{SGD}	0/1
uniform convergence	Section 5.1	\vdash_{unif}	$(\mathbf{f}, \Delta)/(\mathbf{t}, \Delta)$

$$\begin{array}{c}
\frac{}{| \square \vdash_{\text{SGD}} \underline{\text{exp}} : R^{(0)} \rightarrow R_{>0}^{(1)} |} \quad \frac{}{| \square \vdash_{\text{SGD}} \underline{\text{log}} : R_{>0}^{(e)} \rightarrow R^{(0)} |} \\
\frac{}{| \square \vdash_{\text{SGD}} \pm : \iota^{(0)} \rightarrow \iota^{(0)} \rightarrow \iota^{(0)} |} \quad \frac{}{| \square \vdash_{\text{SGD}} \pm : \iota^{(e)} \rightarrow \iota^{(e)} \rightarrow \iota^{(e)} |} \\
\frac{}{| \square \vdash_{\text{SGD}} \underline{=} : R^{(0)} \rightarrow R^{(0)} |} \quad \frac{}{| \square \vdash_{\text{SGD}} \underline{-1} : R_{>0}^{(e)} \rightarrow R_{>0}^{(e)} |} \\
\frac{\Gamma \mid \Sigma \vdash_{\text{SGD}} L : \iota^{(0)} \quad \Gamma \mid \Sigma' \vdash_{\text{SGD}} M : \sigma \quad \Gamma \mid \Sigma'' \vdash_{\text{SGD}} N : \sigma}{\Gamma \mid \Sigma \# \Sigma' \# \Sigma'' \vdash_{\text{SGD}} \mathbf{if} L < 0 \mathbf{then} M \mathbf{else} N : \sigma} \\
\frac{}{| [s_j \sim \mathcal{D}] \vdash_{\text{SGD}} \mathbf{sample}_{\mathcal{D}} : R^{(0)} |} \quad \mathcal{D} \text{ has finite moments}
\end{array}$$

Fig. 4: Excerpt of the typing rules (cf. [18]) for the correctness of SGD.

4.4 A More Permissible Type System

In this section we discuss another instantiation, \vdash_{SGD} , of the generic type system system to guarantee (SGD0) to (SGD3), which is more permissible than \vdash_{poly} . In particular, we would like to support Example 2, which uses logarithms and densities involving exponentials. Intuitively, we need to ensure that subterms involving $\underline{\text{exp}}$ are “neutralised” by a corresponding $\underline{\text{log}}$. To achieve this we annotate base types with 0 or 1, ordered discretely. 0 is the only annotation for safe base types and can be thought of as “integrable”; 1 denotes “needs to be passed through log”. More precisely, we constrain the typing rules such that if $\theta \mid \Sigma \vdash_{\text{SGD}} M : \iota^{(e)}$ then⁶ $\log^e \circ \llbracket M \rrbracket$ and the partial derivatives of $\log^e \circ \llbracket M \rrbracket$ up to order 2 are uniformly dominated by a function with finite moments.

We subtype base types as follows: $\iota_1^{(e_1)} \sqsubseteq_{\text{SGD}} \iota_2^{(e_2)}$ if $\iota_1 \sqsubseteq \iota_2$ (as defined in Fig. 3a) and $e_1 = e_2$, or $\iota_1 = R_{>0} = \iota_2$ and $e_1 \leq e_2$. The second disjunct may come as a surprise but we ensure that terms of type $R_{>0}^{(0)}$ cannot depend on samples at all.

In Fig. 4 we list the most important rules; we relegate the full type system to [18]. $\underline{\text{exp}}$ and $\underline{\text{log}}$ increase and decrease the annotation respectively. The rules for the primitive operations and conditionals are motivated by the closure properties

⁶ using the convention \log^0 is the identity

of Lemma 3 and the elementary fact that $\log \circ (f \cdot g) = (\log \circ f) + (\log \circ g)$ and $\log \circ (f^{-1}) = -\log \circ f$ for $f, g : \Theta \times \mathbb{R}^n \rightarrow \mathbb{R}$.

Example 4. $\theta : R_{>0}^{(0)} \mid [\mathcal{N}, \mathcal{N}] \vdash_{\text{SGD}} \underline{\log}(\theta^{-1} \cdot \underline{\exp}(\mathbf{sample}_{\mathcal{N}})) + \mathbf{sample}_{\mathcal{N}} : R^{(0)}$

Note that the branches of conditionals need to have safe type, which rules out branches with type $R^{(1)}$. This is because logarithms do not behave nicely when composed with addition as used in the smoothed interpretation of conditionals.

Besides, observe that in the rules for logarithm and inverses $e = 0$ is allowed, which may come as a surprise⁷. This is e.g. necessary for the typability of the variational inference Example 2:

Example 5 (Typing for Variational Inference). It holds $\mid \mid \vdash N : R^{(0)} \rightarrow R^{(0)} \rightarrow R_{>0}^{(0)} \rightarrow R_{>0}^{(1)}$ and $\theta : R^{(0)} \mid [s_1 \sim \mathcal{N}] \vdash M : R^{(0)}$.

Type Soundness. To formally establish type soundness, we can use a logical predicate, which is very similar to the one in Section 4.2 (N.B. the additional Item 2): in particular $f \in \mathcal{Q}_{\iota^{(e)}}^{(n)}$ if

1. partial derivatives of $\log^e \circ f$ up to order 2 are uniformly dominated by a function with finite moments
2. if $\iota^{(e)}$ is $R_{>0}^{(0)}$ then f is dominated by a positive constant function

Using this and a similar logical predicate for $\llbracket (-) \rrbracket$ we can show:

Proposition 5. *If $\theta_1 : \iota^{(0)}, \dots, \theta_m : \iota_m^{(0)} \mid \Sigma \vdash_{\text{SGD}} M : \iota^{(0)}$ then*

1. *all distributions in Σ have finite moments*
2. *$\llbracket M \rrbracket$ and for each $\eta > 0$ the partial derivatives up to order 2 of $\llbracket M \rrbracket_{\eta}$ are uniformly dominated by a function with finite moments.*

Consequently, again the Smoothed Optimisation Problem 2 is not only well-defined but by the dominated convergence theorem, the reparameterisation gradient estimator is unbiased. Furthermore, (SGD1) to (SGD3) are satisfied and SGD is correct.

5 Uniform Convergence

In the preceding section we have shown that SGD with the reparameterisation gradient can be employed to correctly (in the sense of Proposition 3) solve the Smoothed Optimisation Problem 2 for any fixed accuracy coefficient. However, *a priori*, it is not clear how a solution of the Smoothed Problem 2 can help to solve the original Problem 1.

The following illustrates the potential for significant discrepancies:

⁷ Recall that terms of type $R_{>0}^{(0)}$ cannot depend on samples.

Example 6. Consider $M \equiv \mathbf{if} \ 0 < 0 \ \mathbf{then} \ \theta \cdot \theta + \underline{1} \ \mathbf{else} \ (\theta - \underline{1}) \cdot (\theta - \underline{1})$. Notice that the global minimum and the only stationary point of $\llbracket M \rrbracket_\eta$ is at $\theta = \frac{1}{2}$ regardless of $\eta > 0$, where $\llbracket M \rrbracket_\eta(\frac{1}{2}) = \frac{3}{4}$. On the other hand $\llbracket M \rrbracket(\frac{1}{2}) = \frac{1}{4}$ and the global minimum of $\llbracket M \rrbracket$ is at $\theta = 1$.

In this section we investigate under which conditions the smoothed objective function converges to the original objective function *uniformly* in $\theta \in \Theta$:

$$(\text{Unif}) \ \mathbb{E}_{\mathbf{s} \sim \mathcal{D}} [\llbracket M \rrbracket_\eta(\theta, \mathbf{s})] \xrightarrow{\text{unif.}} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}} [\llbracket M \rrbracket(\theta, \mathbf{s})] \text{ as } \eta \searrow 0 \text{ for } \theta \in \Theta$$

We design a type system guaranteeing this.

The practical significance of uniform convergence is that *before* running SGD, for every error tolerance $\epsilon > 0$ we can find an accuracy coefficient $\eta > 0$ such that the difference between the smoothed and original objective function does not exceed ϵ , in particular for θ^* delivered by the SGD run for the η -smoothed problem.

Discussion of Restrictions. To rule out the pathology of Example 6 we require that guards are non-0 almost everywhere.

Furthermore, as a consequence of the uniform limit theorem [29], (Unif) can only possibly hold if the *expectation* $\mathbb{E}_{\mathbf{s} \sim \mathcal{D}} [\llbracket M \rrbracket(\theta, \mathbf{s})]$ is continuous (as a function of the parameters θ). For a straightforward counterexample take $M \equiv \mathbf{if} \ \theta < 0 \ \mathbf{then} \ \underline{0} \ \mathbf{else} \ \underline{1}$, we have $\mathbb{E}_{\mathbf{s}}[\llbracket M \rrbracket(\theta)] = [\theta \geq 0]$ which is discontinuous, let alone differentiable, at $\theta = 0$. Our approach is to require that guards do not depend directly on parameters but they may do so, indirectly, via a diffeomorphic⁸ reparameterisation transform; see Example 8. We call such guards *safe*.

In summary, our aim, intuitively, is to ensure that guards are the composition of a diffeomorphic transformation of the random samples (potentially depending on parameters) and a function which does not vanish almost everywhere.

5.1 Type System for Guard Safety

In order to enforce this requirement and to make the transformation more explicit, we introduce syntactic sugar, **transform sample** \mathcal{D} **by** T , for applications of the form $T \ \mathbf{sample} \ \mathcal{D}$.

Example 7. As expressed in Eq. (2), we can obtain samples from $\mathcal{N}(\mu, \sigma^2)$ via **transform sample** \mathcal{N} **by** $(\lambda s. s \cdot \sigma + \mu)$, which is syntactic sugar for the term $(\lambda s. s \cdot \sigma + \mu) \ \mathbf{sample} \ \mathcal{N}$.

We propose another instance of the generic type system of Section 4.3, \vdash_{unif} , where we annotate base types by $\alpha = (g, \Delta)$, where $g \in \{\mathbf{f}, \mathbf{t}\}$ denotes whether we seek to establish guard safety and Δ is a finite set of s_j capturing possible dependencies on samples. We subtype base types as follows: $\iota_1^{(g_1, \Delta_1)} \sqsubseteq_{\text{unif}} \iota_2^{(g_2, \Delta_2)}$

⁸ [18, Example 12] illustrates why it is *not* sufficient to restrict the reparameterisation transform to *bijections* (rather, we require it to be a diffeomorphism).

if $\iota_1 \sqsubseteq \iota_2$ (as defined in Fig. 3a), $\Delta_1 \subseteq \Delta_2$ and $g_1 \preceq g_2$, where $\mathbf{t} \preceq \mathbf{f}$. This is motivated by the intuition that we can always drop⁹ guard safety and add more dependencies.

The rule for conditionals ensures that only safe guards are used. The unary operations preserve variable dependencies and guard safety. Parameters and constants are not guard safe and depend on no samples (see [18] for the full type system):

$$\frac{\frac{\frac{\Gamma \mid \Sigma \vdash_{\text{unif}} L : \iota(\mathbf{t}, \Delta) \quad \Gamma \mid \Sigma' \vdash_{\text{unif}} M : \sigma \quad \Gamma \mid \Sigma'' \vdash_{\text{unif}} N : \sigma}{\Gamma \mid \Sigma ++ \Sigma' ++ \Sigma'' \vdash_{\text{unif}} \mathbf{if} L < 0 \mathbf{then} M \mathbf{else} N : \sigma}}{\mid \square \vdash_{\text{unif}} _ : R^{(g, \Delta)} \rightarrow R^{(g, \Delta)}}}{\frac{\frac{\theta_i : \iota(\mathbf{f}, \emptyset) \mid \square \vdash_{\text{unif}} \theta_i : \iota(\mathbf{f}, \emptyset) \quad \mid \square \vdash_{\text{unif}} _ : \iota(\mathbf{f}, \emptyset)}{\theta \mid \square \vdash_{\text{unif}} T : R^\alpha \rightarrow R^\alpha} \quad r \in \llbracket \iota \rrbracket}{\theta \mid [s_j \sim \mathcal{D}] \vdash_{\text{unif}} \mathbf{transform\ sample} \mathcal{D} \mathbf{by} T : R^{(\mathbf{t}, \{s_j\})}} T \text{ diffeomorphic}}$$

A term $\theta \mid \square \vdash_{\text{unif}} T : R^\alpha \rightarrow R^\alpha$ is diffeomorphic if $\llbracket T \rrbracket(\theta, \square) = \llbracket T \rrbracket_\eta(\theta, \square) : \mathbb{R} \rightarrow \mathbb{R}$ is a diffeomorphism for each $\theta \in \Theta$, i.e. differentiable and bijective with differentiable inverse.

First, we can express affine transformations, in particular, the location-scale transformations as in Example 7:

Example 8 (Location-Scale Transformation). The term-in-context

$$\sigma : R_{>0}^{(\mathbf{f}, \emptyset)}, \mu : R^{(\mathbf{f}, \emptyset)} \mid \square \vdash \lambda s. \sigma \cdot s + \mu : R^{(\mathbf{f}, \{s_1\})} \rightarrow R^{(\mathbf{f}, \{s_1\})}$$

is diffeomorphic. (However for $\sigma : R^{(\mathbf{f}, \emptyset)}$ it is *not* because it admits $\sigma = 0$.) Hence, the reparameterisation transform

$$G \equiv \sigma : R_{>0}^{(\mathbf{f}, \emptyset)}, \mu : R^{(\mathbf{f}, \emptyset)} \mid [s_1 : \mathcal{D}] \vdash \mathbf{transform\ sample} \mathcal{D} \mathbf{by} (\lambda s. s \cdot \sigma + \mu) : R^{(\mathbf{t}, \{s_1\})}$$

which has g -flag \mathbf{t} , is admissible as a guard term. Notice that G depends on the parameters, σ and μ , *indirectly* through a diffeomorphism, which is permitted by the type system.

If guard safety is sought to be established for the binary operations, we require that operands do not share dependencies on samples:

$$\frac{\frac{\mid \square \vdash_{\text{unif}} \ominus : \iota(\mathbf{f}, \Delta) \rightarrow \iota(\mathbf{f}, \Delta) \rightarrow \iota(\mathbf{f}, \Delta)}{\mid \square \vdash_{\text{unif}} \ominus : \iota(\mathbf{f}, \Delta) \rightarrow \iota(\mathbf{f}, \Delta) \rightarrow \iota(\mathbf{f}, \Delta)} \circ \in \{+, \cdot\}}{\mid \square \vdash_{\text{unif}} \ominus : \iota(\mathbf{t}, \Delta_1) \rightarrow \iota(\mathbf{t}, \Delta_2) \rightarrow \iota(\mathbf{t}, \Delta_1 \cup \Delta_2)} \circ \in \{+, \cdot\}, \Delta_1 \cap \Delta_2 = \emptyset}$$

This is designed to address:

⁹ as long as it is not used in guards

Example 9 (Non-Constant Guards). We have $|\Box \vdash (\lambda x.x + (-x)) : R^{\mathbf{f},\{s_1\}} \rightarrow R^{\mathbf{f},\{s_1\}}$, noting that we must use $g = \mathbf{f}$ for the $\underline{\pm}$ rule; and because $R^{\mathbf{t},\{s_j\}} \sqsubseteq_{\text{unif}} R^{\mathbf{f},\{s_j\}}$, we have

$$|\Box \vdash (\lambda x.x + (\underline{-}x)) : R^{\mathbf{t},\{s_1\}} \rightarrow R^{\mathbf{f},\{s_1\}}.$$

Now **transform sample** \mathcal{D} **by** $(\lambda y.y)$ has type $R^{\mathbf{t},\{s_1\}}$ with the g -flag necessarily set to \mathbf{t} ; and so the term

$$M \equiv (\lambda x.x + (-x)) \text{ **transform sample } \mathcal{D} \text{ by } (\lambda y.y)**$$

which denotes 0, has type $R^{\mathbf{f},\{s_1\}}$, but *not* $R^{\mathbf{t},\{s_1\}}$. It follows that M cannot be used in guards (notice the side condition of the rule for conditional), which is as desired: recall Example 6. Similarly consider the term

$$N \equiv (\lambda x.(\lambda y.z.\text{if } y + (-z) < 0 \text{ then } M_1 \text{ else } M_2) x x) \\ (\text{transform sample } \mathcal{D} \text{ by } (\lambda y.y)) \quad (7)$$

When evaluated, the term $y + (-z)$ in the guard has denotation 0. For the same reason as above, the term N is not refinement typable.

The type system is however incomplete, in the sense that there are terms-in-context that satisfy the property (Unif) but which are not typable.

Example 10 (Incompleteness). The following term-in-context denotes the ‘‘identity’’:

$$|\Box \vdash (\lambda x.(\underline{2} \cdot x) + (-x)) : R^{\mathbf{t},\{s_1\}} \rightarrow R^{\mathbf{f},\{s_1\}}$$

but it does *not* have type $R^{\mathbf{t},\{s_1\}} \rightarrow R^{\mathbf{t},\{s_1\}}$. Then, using the same reasoning as Example 9, the term

$$G \equiv (\lambda x.(\underline{2} \cdot x) + (-x)) (\text{transform sample } \mathcal{D} \text{ by } (\lambda y.y))$$

has type $R^{\mathbf{f},\{s_1\}}$, but *not* $R^{\mathbf{t},\{s_1\}}$, and so **if** $G < 0$ **then** $\underline{0}$ **else** $\underline{1}$ is not typable, even though G can safely be used in guards.

5.2 Type Soundness

Henceforth, we fix parameters $\theta_1 : \iota_1^{(\mathbf{f},\emptyset)}, \dots, \theta_m : \iota_m^{(\mathbf{f},\emptyset)}$.

Now, we address how to show property (Unif), i.e. that for $\theta \mid \Sigma \vdash_{\text{unif}} M : \iota^{(g,\Delta)}$, the η -smoothed $\mathbb{E}[\llbracket M \rrbracket_\eta](\theta, \mathbf{s})$ converges uniformly for $\theta \in \Theta$ as $\eta \searrow 0$. For this to hold we clearly need to require that σ_η has good (uniform) convergence properties (as far as the unavoidable discontinuity at 0 allows for):

Assumption 3 For every $\delta > 0$, $\sigma_\eta \xrightarrow{\text{unif.}} [(-) > 0]$ on $(-\infty, -\delta) \cup (\delta, \infty)$.

Observe that in general even if M is typable $\llbracket M \rrbracket_\eta$ does *not* converge uniformly in both θ and \mathbf{s} because $\llbracket M \rrbracket$ may still be discontinuous in \mathbf{s} :

Example 11. For $M \equiv \mathbf{if}(\mathbf{transform\ sample}_{\mathcal{N}} \mathbf{by}(\lambda s. s + \theta)) < 0 \mathbf{then} \mathbf{0} \mathbf{else} \mathbf{1}$, $\llbracket M \rrbracket(\theta, s) = [s + \theta \geq 0]$, which is discontinuous, and $\llbracket M \rrbracket_{\eta}(\theta, s) = \sigma_{\eta}(s + \theta)$.

However, if $\theta \mid \Sigma \vdash M : \iota^{(g, \Delta)}$ then $\llbracket M \rrbracket_{\eta}$ does converge to $\llbracket M \rrbracket$ uniformly almost uniformly, i.e., uniformly in $\theta \in \Theta$ and almost uniformly in $\mathbf{s} \in \mathbb{R}^n$. Formally, we define:

Definition 4. Let $f, f_{\eta} : \Theta \times \mathbb{R}^n \rightarrow \mathbb{R}$, μ be a measure on \mathbb{R}^n . We say that f_{η} converges uniformly almost uniformly to f (notation: $f_{\eta} \xrightarrow{u.a.u.} f$) if there exist sequences $(\delta_k)_{k \in \mathbb{N}}$, $(\epsilon_k)_{k \in \mathbb{N}}$ and $(\eta_k)_{k \in \mathbb{N}}$ such that $\lim_{k \rightarrow \infty} \delta_k = 0 = \lim_{k \rightarrow \infty} \epsilon_k$; and for every $k \in \mathbb{N}$ and $\theta \in \Theta$ there exists $U \subseteq \mathbb{R}^n$ such that

1. $\mu(U) < \delta_k$ and
2. for every $0 < \eta < \eta_k$ and $\mathbf{s} \in \mathbb{R}^n \setminus U$, $|f_{\eta}(\theta, \mathbf{s}) - f(\theta, \mathbf{s})| < \epsilon_k$.

If f, f_{η} are independent of θ this notion coincides with standard almost uniform convergence. For M from Example 11 $\llbracket M \rrbracket_{\eta} \xrightarrow{u.a.u.} \llbracket M \rrbracket$ holds although uniform convergence fails.

However, uniform almost uniform convergence entails uniform convergence of expectations:

Lemma 6. Let $f, f_{\eta} : \Theta \times \mathbb{R}^n \rightarrow \mathbb{R}$ have finite moments.

If $f_{\eta} \xrightarrow{u.a.u.} f$ then $\mathbb{E}_{\mathbf{s} \sim \mathcal{D}}[f_{\eta}(\theta, \mathbf{s})] \xrightarrow{unif.} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}}[f(\theta, \mathbf{s})]$.

As a consequence, it suffices to establish $\llbracket M \rrbracket_{\eta} \xrightarrow{u.a.u.} \llbracket M \rrbracket$. We achieve this by positing an infinitary logical relation between sequences of morphisms in **VectFr** (corresponding to the smoothings) and morphisms in **QBS** (corresponding to the measurable standard semantics). We then prove a fundamental lemma (details are in [18]). Not surprisingly the case for conditionals is most interesting. This makes use of Assumption 3 and exploits that guards, for which the typing rules assert the guard safety flag to be \mathbf{t} , can only be 0 at sets of measure 0. We conclude:

Theorem 1. If $\theta_1 : \iota_1^{(\mathbf{f}, \emptyset)}, \dots, \theta_m : \iota_m^{(\mathbf{f}, \emptyset)} \mid \Sigma \vdash_{\text{unif}} M : R^{(g, \Delta)}$ then $\llbracket M \rrbracket_{\eta} \xrightarrow{u.a.u.} \llbracket M \rrbracket$. In particular, if $\llbracket M \rrbracket_{\eta}$ and $\llbracket M \rrbracket$ also have finite moments then

$$\mathbb{E}_{\mathbf{s} \sim \mathcal{D}}[\llbracket M \rrbracket_{\eta}(\theta, \mathbf{s})] \xrightarrow{unif.} \mathbb{E}_{\mathbf{s} \sim \mathcal{D}}[\llbracket M \rrbracket(\theta, \mathbf{s})] \quad \text{as } \eta \searrow 0 \text{ for } \theta \in \Theta$$

We finally note that \vdash_{unif} can be made more permissible by adding syntactic sugar for a -fold (for $a \in \mathbb{N}_{>0}$) addition $\underline{a} \cdot M \equiv M \underline{+} \dots \underline{+} M$ and multiplication $M \underline{^a} \equiv M \underline{\cdot} \dots \underline{\cdot} M$. This admits more terms as guards, but safely [18].

6 Related Work

[23] is both the starting point for our work and the most natural source for comparison. They correct the (biased) reparameterisation gradient estimator for non-differentiable models by additional non-trivial *boundary* terms. They present

an efficient method for *affine* guards only. Besides, they are not concerned with the *convergence* of gradient-based optimisation procedures; nor do they discuss how assumptions they make may be manifested in a programming language.

In the context of the reparameterisation gradient, [25] and [17] relax discrete random variables in a continuous way, effectively dealing with a specific class of discontinuous models. [39] use a similar smoothing for discontinuous optimisation but they do not consider a full programming language.

Motivated by guaranteeing absolute continuity (which is a necessary but not sufficient criterion for the correctness of e.g. variational inference), [24] use an approach similar to our trace types to track the samples which are drawn. They do not support standard conditionals but their “work-around” is also eager in the sense of combining the traces of both branches. Besides, they do not support a full higher-order language, in which higher-order terms can draw samples. Thus, they do not need to consider *function* types tracking the samples drawn during evaluation.

7 Empirical Evaluation

We evaluate our smoothed gradient estimator (SMOOTH) against the biased reparameterisation estimator (REPARAM), the unbiased correction of it (LYY18) due to [23], and the unbiased (SCORE) estimator [31,38,27]. The experimental setup is based on that of [23]. The implementation is written in Python, using automatic differentiation (provided by the `jax` library) to implement each of the above estimators for an arbitrary probabilistic program. For each estimator and model, we used the Adam [19] optimiser for 10,000 iterations using a learning rate of 0.001, with the exception of `xornet` for which we used 0.01. The initial model parameters θ_0 were fixed for each model across all runs. In each iteration, we used $N = 16$ Monte Carlo samples from the gradient estimator. For the LYY18 estimator, a single subsample for the boundary term was used in each estimate. For our smoothed estimator we use accuracy coefficients $\eta \in \{0.1, 0.15, 0.2\}$. Further details are discussed in [18, Appendix E.1].

Compilation for First-Order Programs. All our benchmarks are first-order. We compile a potentially discontinuous program to a smooth program (parameterised by σ_η) using the compatible closure of

$$\mathbf{if } L < 0 \mathbf{ then } M \mathbf{ else } N \rightsquigarrow (\lambda w. \sigma_\eta(-w) \cdot M + \sigma_\eta(w) \cdot N) L$$

Note that the size only increases linearly and that we avoid of an exponential blow-up by using abstractions rather than duplicating the guard L .

Models. We include the models from [23], an example from differential privacy [11] and a neural network for which our main competitor, the estimator of [23], is *not* applicable (see [18, Appendix E.2] for more details).

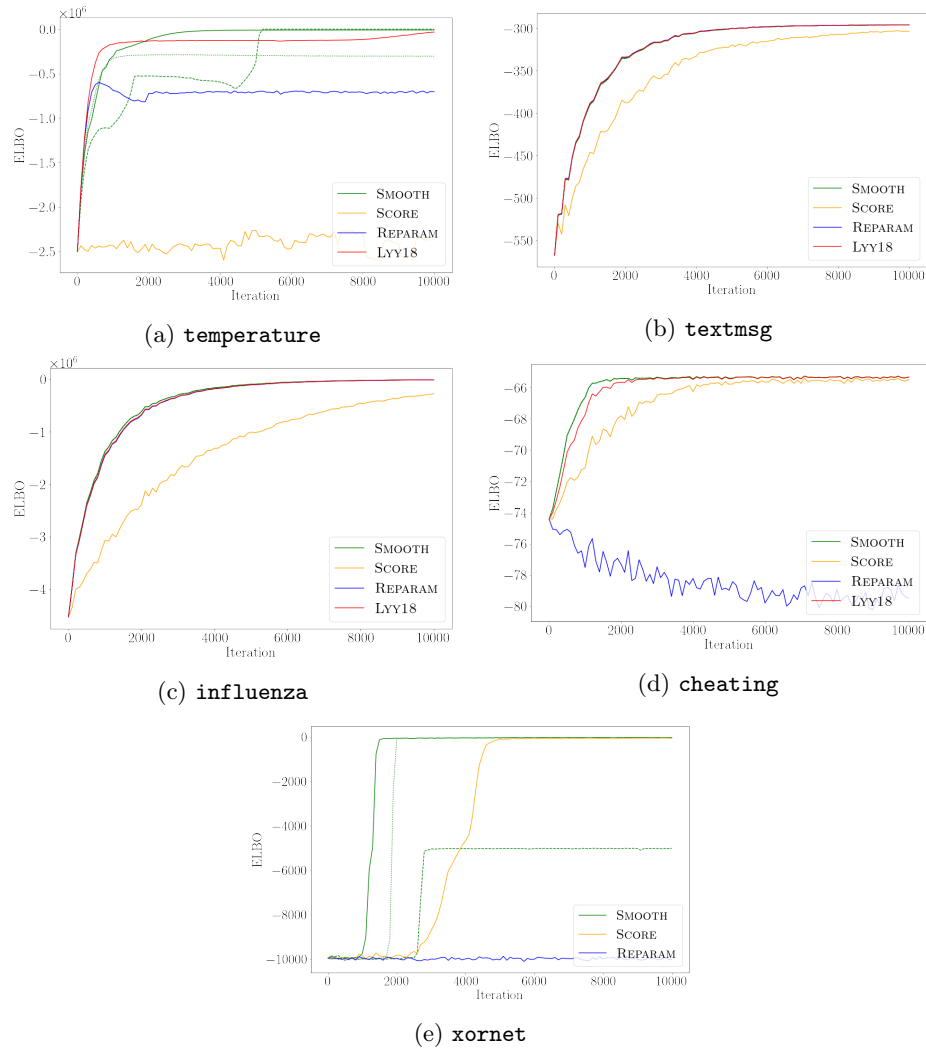


Fig. 5: ELBO trajectories for each model. A single colour is used for each estimator and the accuracy coefficient $\eta = 0.1, 0.15, 0.2$ for SMOOTH is represented by dashed, solid and dotted lines respectively.

Analysis of Results

We plot the ELBO trajectories in Fig. 5 and include data on the computational cost and *work-normalised* variance [8] in [18, Table 2]. (Variances can be improved in a routine fashion by e.g. taking more samples.)

The ELBO graph for the **temperature** model in Fig. 5a and the **cheating** model in Fig. 5d shows that the REPARAM estimator is biased, converging to suboptimal values when compared to the SMOOTH and LYY18 estimators. For

`temperature` we can also see from the graph and the data in [18, Table 2a] that the SCORE estimator exhibits extremely high variance, and does not converge.

Finally, the `xornet` model shows the difficulty of training step-function based neural nets. The LYY18 estimator is not applicable here since there are non-affine conditionals. In Fig. 5e, the REPARAM estimator makes no progress while other estimators manage to converge to close to 0 ELBO, showing that they learn a network that correctly classifies all points. In particular, the SMOOTH estimator converges the quickest.

Summa summarum, the results reveal where the REPARAM estimator is biased and that the SMOOTH estimator does not have the same limitation. Where the LYY18 estimator is defined, they converge to roughly the same objective value. Our smoothing approach is generalisable to more complex models such as neural networks with non-linear boundaries, as well as simpler and cheaper (there is no need to compute a correction term). Besides, our estimator has consistently significantly lower work-normalised variance, up to 3 orders of magnitude.

8 Conclusion and Future Directions

We have discussed a simple probabilistic programming language to formalise an optimisation problem arising e.g. in variational inference for probabilistic programming. We have endowed our language with a denotational (measurable) value semantics and a smoothed approximation of potentially discontinuous programs, which is parameterised by an accuracy coefficient. We have proposed type systems to guarantee pleasing properties in the context of the optimisation problem: For a fixed accuracy coefficient, stochastic gradient descent converges to stationary points even with the reparameterisation gradient (which is *unbiased*). Besides, the smoothed objective function converges uniformly to the true objective as the accuracy is improved.

Our type systems can be used to *independently* check these two properties to obtain partial theoretical guarantees even if one of the systems suffers from incompleteness. We also stress that SGD and the smoothed unbiased gradient estimator can even be applied to programs which are *not* typable.

Experiments with our prototype implementation confirm the benefits of reduced variance and unbiasedness. Compared to the unbiased correction of the reparameterised gradient estimator due to [23], our estimator has a similar convergence, but is simpler, faster, and attains orders of magnitude (2 to 3,000 x) reduction in work-normalised variance.

Future Directions. A natural avenue for future research is to make the language and type systems more complete, i.e. to support more well-behaved programs, in particular programs involving recursion.

Furthermore, the choice of accuracy coefficients leaves room for further investigations. We anticipate it could be fruitful not to fix an accuracy coefficient upfront but to gradually enhance it *during* the optimisation either via a pre-determined schedule (dependent on structural properties of the program), or adaptively.

References

1. Aumann, R.J.: Borel structures for function spaces. *Illinois Journal of Mathematics* **5** (1961)
2. Bertsekas, D.: *Convex optimization algorithms*. Athena Scientific (2015)
3. Bertsekas, D.P., Tsitsiklis, J.N.: Gradient convergence in gradient methods with errors. *SIAM J. Optim.* **10**(3), 627–642 (2000)
4. Bingham, E., Chen, J.P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P.A., Horsfall, P., Goodman, N.D.: Pyro: Deep universal probabilistic programming. *J. Mach. Learn. Res.* **20**, 28:1–28:6 (2019)
5. Bishop, C.M.: *Pattern recognition and machine learning*, 5th Edition. Information science and statistics, Springer (2007)
6. Blei, D.M., Kucukelbir, A., McAuliffe, J.D.: Variational inference: A review for statisticians. *Journal of the American Statistical Association* **112**(518), 859–877 (2017)
7. Borgström, J., Lago, U.D., Gordon, A.D., Szymczak, M.: A lambda-calculus foundation for universal probabilistic programming. In: *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming, ICFP 2016, Nara, Japan, September 18–22, 2016*. pp. 33–46 (2016)
8. Botev, Z., Ridder, A.: Variance Reduction. In: *Wiley StatsRef: Statistics Reference Online*, pp. 1–6 (2017)
9. Cusumano-Towner, M.F., Saad, F.A., Lew, A.K., Mansinghka, V.K.: Gen: a general-purpose probabilistic programming system with programmable inference. In: *McKinley, K.S., Fisher, K. (eds.) Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22–26, 2019*. pp. 221–236. ACM (2019)
10. Dahlqvist, F., Kozen, D.: Semantics of higher-order probabilistic programs with conditioning. *Proc. ACM Program. Lang.* **4**(POPL), 57:1–57:29 (2020)
11. Davidson-Pilon, C.: *Bayesian Methods for Hackers: Probabilistic Programming and Bayesian Inference*. Addison-Wesley Professional (2015)
12. Ehrhard, T., Tasson, C., Pagani, M.: Probabilistic coherence spaces are fully abstract for probabilistic PCF. In: *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL ’14, San Diego, CA, USA, January 20–21, 2014*. pp. 309–320 (2014)
13. Frölicher, A., Kriegl, A.: *Linear Spaces and Differentiation Theory*. Interscience, J. Wiley and Son, New York (1988)
14. Heunen, C., Kammar, O., Staton, S., Yang, H.: A convenient category for higher-order probability theory. *Proc. Symposium Logic in Computer Science* (2017)
15. Heunen, C., Kammar, O., Staton, S., Yang, H.: A convenient category for higher-order probability theory. In: *32nd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2017, Reykjavik, Iceland, June 20–23, 2017*. pp. 1–12 (2017)
16. Hur, C., Nori, A.V., Rajamani, S.K., Samuel, S.: A provably correct sampler for probabilistic programs. In: *35th IARCS Annual Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2015, December 16–18, 2015, Bangalore, India*. pp. 475–488 (2015)
17. Jang, E., Gu, S., Poole, B.: Categorical reparameterization with gumbel-softmax. In: *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24–26, 2017, Conference Track Proceedings* (2017)

18. Khajwal, B., Ong, C.L., Wagner, D.: Fast and correct gradient-based optimisation for probabilistic programming via smoothing (2023), <https://arxiv.org/abs/2301.03415>
19. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. In: Bengio, Y., LeCun, Y. (eds.) 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings (2015)
20. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. In: Bengio, Y., LeCun, Y. (eds.) 2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings (2014)
21. Klenke, A.: Probability Theory: A Comprehensive Course. Universitext, Springer London (2014)
22. Lee, W., Yu, H., Rival, X., Yang, H.: Towards verified stochastic variational inference for probabilistic programs. PACMPL **4**(POPL) (2020)
23. Lee, W., Yu, H., Yang, H.: Reparameterization gradient for non-differentiable models. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada. pp. 5558–5568 (2018)
24. Lew, A.K., Cusumano-Towner, M.F., Sherman, B., Carbin, M., Mansinghka, V.K.: Trace types and denotational semantics for sound programmable inference in probabilistic languages. Proc. ACM Program. Lang. **4**(POPL), 19:1–19:32 (2020)
25. Maddison, C.J., Mnih, A., Teh, Y.W.: The concrete distribution: A continuous relaxation of discrete random variables. In: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings (2017)
26. Mak, C., Ong, C.L., Paquet, H., Wagner, D.: Densities of almost surely terminating probabilistic programs are differentiable almost everywhere. In: Yoshida, N. (ed.) Programming Languages and Systems - 30th European Symposium on Programming, ESOP 2021, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2021, Luxembourg City, Luxembourg, March 27 - April 1, 2021, Proceedings. Lecture Notes in Computer Science, vol. 12648, pp. 432–461. Springer (2021)
27. Minh, A., Gregor, K.: Neural variational inference and learning in belief networks. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. JMLR Workshop and Conference Proceedings, vol. 32, pp. 1791–1799. JMLR.org (2014)
28. Mohamed, S., Rosca, M., Figurnov, M., Mnih, A.: Monte carlo gradient estimation in machine learning. J. Mach. Learn. Res. **21**, 132:1–132:62 (2020)
29. Munkres, J.R.: Topology. Prentice Hall, New Delhi, 2nd. edn. (1999)
30. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press (2012)
31. Ranganath, R., Gerrish, S., Blei, D.M.: Black box variational inference. In: Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics, AISTATS 2014, Reykjavik, Iceland, April 22-25, 2014. pp. 814–822 (2014)
32. Rezende, D.J., Mohamed, S., Wierstra, D.: Stochastic backpropagation and approximate inference in deep generative models. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. JMLR Workshop and Conference Proceedings, vol. 32, pp. 1278–1286. JMLR.org (2014)
33. Stacey, A.: Comparative smootheology. Theory and Applications of Categories **25**(4), 64–117 (2011)

34. Staton, S.: Commutative semantics for probabilistic programming. In: Programming Languages and Systems - 26th European Symposium on Programming, ESOP 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings. pp. 855–879 (2017)
35. Staton, S., Yang, H., Wood, F.D., Heunen, C., Kammar, O.: Semantics for probabilistic programming: higher-order functions, continuous distributions, and soft constraints. In: Proceedings of the 31st Annual ACM/IEEE Symposium on Logic in Computer Science, LICS '16, New York, NY, USA, July 5-8, 2016. pp. 525–534 (2016)
36. Titsias, M.K., Lázaro-Gredilla, M.: Doubly stochastic variational bayes for non-conjugate inference. In: Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014. pp. 1971–1979 (2014)
37. Vákár, M., Kammar, O., Staton, S.: A domain theory for statistical probabilistic programming. PACMPL **3**(POPL), 36:1–36:29 (2019)
38. Wingate, D., Weber, T.: Automated variational inference in probabilistic programming. CoRR **abs/1301.1299** (2013)
39. Zang, I.: Discontinuous optimization by smoothing. Mathematics of Operations Research **6**(1), 140–152 (1981)
40. Zhang, C., Butepage, J., Kjellstrom, H., Mandt, S.: Advances in Variational Inference. IEEE Trans. Pattern Anal. Mach. Intell. **41**(8), 2008–2026 (2019)