Semantics for Automatic Differentiation Towards Probabilistic Programming

Literature Review and Dissertation Proposal

Mathieu Huot, St Cross College
DPhil Candidate in Computer Science
Supervisor: Sam Staton

Tuesday 10th March, 2020

Contents

1 Introduction and Motivation

This work contains a literature review and a thesis proposal. It is structured as follows. First, a general introduction to Automatic Differentiation (AD) and some motivations are given. Next, I briefly present previous work I completed on this topic. Then, I present a short literature review on differentiable programming, which roughly consists in importing AD to a functional setting.

This is followed by a short literature review on some aspects of probabilistic programming. These sections are closer than it might look at first glance. Indeed, I see probabilistic programming for AI as the straightforward extension of efficient gradient-based approaches which AD permits, both in terms of generality and applicability, usually to the detriment of performance. Therefore, a complex trade-off is often required between expressivity and performance, and much of the state-of-the-art work lies at the intersection between the two fields.

I then move on to the thesis proposal where I explain what contribution I am willing to make on these topics. This document ends with a summary and risk assessment.

1.1 Introduction

Automatic differentiation is the process of taking a program describing a function, and build another program describing the derivative of that function by applying the chain rule across the program code. As gradients play a central role in many aspects of machine learning, so do automatic differentiation systems such as TensorFlow [Abadi et al., 2016], PyTorch [Paszke et al., 2017], or Stan [Carpenter et al., 2015].

Under the slogan of differentiable programming [Olah, 2015, LeCun, 2018], there is an increasing demand for efficient automatic gradient computation for emerging network architectures that incorporate dynamic control flow, especially in natural language processing (NLP). Differentiable programming refers to a programming model where neural networks are truly functional blocks with data-dependent branches and recursion, while at the same time being trainable with back-propagation and gradient descent. A programming model of such generality requires both expressivity and efficiency from the back-propagation framework. However, the current generation of tools such as TensorFlow and PyTorch trade off one for the other.

There is already an extensive amount of recent work on differentiable programming [Wang et al., 2018, Manzyuk et al., 2012, Baydin and Pearlmutter, 2014, Pearlmutter and Siskind, 2008], as well as

more theoretical work [Cruttwell et al., 2019, Manzyuk, 2012, Cruttwell, 2017, Cockett and Lemay, 2019, Kelly et al., 2016].

As said above, differentiable programming primarily emerged in the context of machine learning (ML) and more specifically in deep learning (DL), so this context is to be kept in mind. Still, it seems important to abstract away from some specific AI architecture and differentiable programming seems of interest in itself as a new programming paradigm. This is reminiscent of probabilistic programming and one might want to explore further the links between the two.

More specifically, a lot of state-of-the-art gradient descent (GD) algorithms as used in AI include different features coming from probabilities and statistics. Indeed, vanilla gradient descent often faces several difficulties [Bengio et al., 1994]. Examples of generalizations include Stochastic Gradient Descent (SGD) [Bottou, 2010, Bottou, 2012, Mandt et al., 2017], adding a momentum term in GD [Swanston et al., 1994, Abdul Hamid et al., 2011, Shao and Zheng, 2009, Rehman and Nawi, 2011] or even in SGD [Welling and Teh, 2011, Allen-Zhu, 2017, Kingma and Ba, 2014], improving the learning rate with the Hessian or approximate Hessian [Martens, 2010, Becker et al., 1988], or doing back-propagation with noise [Rezende et al., 2014].

Another series of examples comes from (Bayesian) inference-based algorithms. Here, the idea is to use the specific shape of the problem considered to turn it into an optimization-based one. Some version of GD is then used to solve the optimization problem and find a good approximate of a true posterior. Examples include variational inference (VI) [Paisley et al., 2012, Hoffman et al., 2013, Saul and Jordan, 1996, Kingma and Welling, 2013], non-parametric generic modelling [Liutkus et al., 2018], or Hamiltonian Monte Carlo (HMC) algorithms [Girolami and Calderhead, 2011, Hoffman and Gelman, 2014, Chen et al., 2014].

These examples show how ubiquitous gradient-based algorithms are, but this also shows that there is an impressive variety of problems whose solution is developped in a very specific sub-field of research, usually with some convenient ad-hoc assumptions on the structure of the problem or the solution. A unifying framework for these various algorithms and their correctness, their correct implementation, as well as the correctness of the various optimizations used, notably variance reduction techniques [Kingma and Welling, 2014, Johnson and Zhang, 2013, Shang et al., 2018], is therefore desirable.

My proposed research involves studying denotational semantics for higher-order automatic differentiation, and its interaction with various effects as required by modern needs in AI. Effects are impure operations, the most important ones in this case being probabilities and memory management, though others like non-determinism or continuations certainly also play a role, e.g. as back-propagation can be expressed functionally using continuations [Wang et al., 2018, Pearlmutter and Siskind, 2008]. This is to be put in perspective with recent developments in denotational semantics for higher-order probabilistic programming [Heunen et al., 2017, Ścibior et al., 2017], and more precise links between the two should be explored.

The motivation for my work partly comes from the recent interests in differentiable programming and a will to mimic recent developments in probabilistic programming. One hope is to abstract AD enough to be able to see a lot of variations of the GD algorithms as instances of the same more general procedure. This seems plausible for all the common variants and optimizations of the usual GD algorithm, but this is bigger a challenge when it comes to interpreting problems turned into optimization procedures coming from variational inference and non-parametric settings. Another hope is to develop the denotational theory of AD and make links between it and the denotational theory of probabilistic programming. This would shed light on the subtle interaction between differentiation and randomness as is required for modern and future research in AI, e.g. better understand generative adversarial networks (GAN) [Goodfellow et al., 2014, Radford et al., 2015] and variational auto-encoders (VAE) [Kingma and Welling, 2013, Rezende et al., 2014].

1.2 Work Completed

In [Huot et al., 2020], we used the category of diffeological spaces [Souriau, 1980] to give a denotational semantics to a higher-order functional language with smooth primitives. Correctness for higher-order is known to be subtle [Manzyuk et al., 2012, Siskind and Pearlmutter, 2005]. We used diffeological spaces and categorical gluing to show correctness of a syntactic differentiation operator in the sense that a a first-order program P which denotes a smooth function f is sent by this operator to a program P' denoting a smooth function g which computes the derivatives of f. That roughly means that $P(i)$ is denoted by the i-th partial derivative of f. A denotational semantics is then useful to talk about differentiation in a clear setting and for higher-order programs. The non-trivial part lies in the fact that P, P' might have higher-order sub-parts for which the usual notion of differentiation does not exist canonically. The use of diffeological spaces is also very well suited for interpreting a rich functional language. We use it for instance to interpret product types, variant types and inductive types, which at least require some good notions from differential geometry to be given a denotational semantics as smoothness is the primary concern here.

Before that, I worked on quantum theory for some time and published a paper at LICS [Huot and Staton, 2019]. This work is not related to my work proposal so it is only briefly mentioned here. The idea was to characterize functions for quantum information theory with measurement (Quantum channels) as coming from a universal (in the sense of category theory) completion of the functions for pure quantum theory (which are reversible).

2 Literature review: Automatic Differentiation in Functional Programming

2.1 Motivation

There are several reasons why one might want to import Automatic Differentiation (AD) into a functional framework. This might at first be counterintuitive as AD is meant to be a simple yet highly efficient way of computing gradients, and hence a low level implementation where local optimizations and a good memory management seems like the good choice.

However, such low level representations (like computational graphs [Griewank et al., 1989]) lack information on the global structure and hence lack more general optimizations coming from the algebra of the problem. Consider the following toy example:

$$\sum_{1\leq i\leq n} y \times f(i) \longrightarrow y \times \Big(\sum_{1\leq i\leq n} \times f(i) \Big)$$

This obvious optimization avoids $n-1$ multiplications which can be significant if n is big. Of course to see it you need to know that such a sum is performed and intermediate level languages help us spot that.

Functional programming is closer to pseudo-code and mathematics and gives us a good language to reason about these algebraic properties that lead to optimizations. It provides a clearer code with higher reusability. This might explain why Pytorch puts significant effort into getting more and more functional. This code efficiency goes beyond optimizations as it also allows to avoid some code duplication and design better implementations of algorithms. Indeed, there is usually a huge difference between pseudo-code and an efficiently implemented algorithm, and again functional programming is a convenient tool which can serve as an intermediate step.

Differentiable programming tends to go beyond pure AD by including effects such as non-determinism, concurrency, probabilities which are necessary for modern algorithms which need some sort of

gradient-descent (GD) and hence indirectly AD, and functional programmign is very good at that. For instance, monads are a useful tool for capturing specific (unpure) effects and somehow isolate them from the pure computation.

Furthermore, FP can be used as a pedagogical tool. An FP implementation can often be closer to the mathematical pseudo-code and hence this leads to simplification of algorithms, e.g. by splitting the implementation and optimizations parts more easily. A further goal would be to provide a general framework for all the variants of GD and other gradient-heavy algorithms. The panorama is currently a zoo and a more high-level perspective on optimization via gradients might be desirable.

2.2 Simple first-order language example

We now give an example of a very simple idealised first-order functional language to illustrate how AD works in a functional framework. It is a slightly simplified first-order version of the language used in [Huot et al., 2020].

Types:

$$\tau := \mathbb{R} \mid \tau \times \tau$$

Terms:

$$e := c \mid x \mid e + e \mid e * e \mid cos(e) \mid sin(e) \mid exp(e) \mid \langle e, e \rangle \mid \pi_i(e) \mid let\ x = e\ in\ e$$

The typing rules are:

$$\frac{}{\Gamma \vdash c : \mathbb{R}}(c \in \mathbb{R}) \quad \frac{\Gamma \vdash t : \mathbb{R} \qquad \Gamma \vdash s : \mathbb{R}}{\Gamma \vdash t + s : \mathbb{R}} \quad \frac{\Gamma \vdash t : \mathbb{R} \qquad \Gamma \vdash s : \mathbb{R}}{\Gamma \vdash t * s : \mathbb{R}} \quad \frac{\Gamma \vdash t : \mathbb{R}}{\Gamma \vdash f(t) : \mathbb{R}}(f \in \{cos, sin, exp\})$$

$$\frac{\Gamma \vdash t : \tau \qquad \Gamma \vdash s : \sigma}{\Gamma \vdash \langle t, s \rangle : \tau \times \sigma} \quad \frac{\Gamma \vdash t : \tau_1 \times \tau_2}{\Gamma \vdash \pi_i t : \tau_i} \quad \frac{}{\Gamma \vdash x : \tau}(x \in \Gamma) \quad \frac{\Gamma \vdash t : \tau \qquad \Gamma, x : \tau \vdash s : \sigma}{\Gamma \vdash let\ x = t\ in\ s : \sigma}$$

We can now define AD as a meta-operator $\overrightarrow{\mathcal{D}}$ acting on the language defined inductively as follows, on types:

$$\begin{aligned} \overrightarrow{\mathcal{D}}(\mathbb{R}) &= \mathbb{R} \times \mathbb{R} \\ \overrightarrow{\mathcal{D}}(\tau \times \sigma) &= \overrightarrow{\mathcal{D}}(\tau) \times \overrightarrow{\mathcal{D}}(\sigma) \end{aligned}$$

and on terms:

$$\begin{aligned} \overrightarrow{\mathcal{D}}(c) &= (c, 0) \\ \overrightarrow{\mathcal{D}}(y : \tau) &= y : \overrightarrow{\mathcal{D}}(\tau) \\ \overrightarrow{\mathcal{D}}(e_1 + e_2) &= \begin{cases} let\ (a, a') = \overrightarrow{\mathcal{D}}(e_1)\ in \\ let\ (b, b') = \overrightarrow{\mathcal{D}}(e_2)\ in \\ (a + b, a' + b') \end{cases} \\ \overrightarrow{\mathcal{D}}(e_1 * e_2) &= \begin{cases} let\ (a, a') = \overrightarrow{\mathcal{D}}(e_1)\ in \\ let\ (b, b') = \overrightarrow{\mathcal{D}}(e_2)\ in \\ (a * b, a * b' + a' * b) \end{cases} \\ \overrightarrow{\mathcal{D}}(let\ y = e_1\ in\ e_2) &= let\ \overrightarrow{\mathcal{D}}(y) = \overrightarrow{\mathcal{D}}(e_1)\ in\ \overrightarrow{\mathcal{D}}(e_2) \\ \overrightarrow{\mathcal{D}}(sin(e)) &= \begin{cases} let\ (a, a') = \overrightarrow{\mathcal{D}}(e)\ in \\ (sin(a), a' * cos(a)) \end{cases} \end{aligned}$$

Note that

$$\begin{cases} let\ (a, a') = \overrightarrow{\mathcal{D}}(e_1)\ in \\ let\ (b, b') = \overrightarrow{\mathcal{D}}(e_2)\ in \\ (a + b, a' + b') \end{cases}$$

is considered syntactic sugar for

$$\begin{cases} let\ a = \overrightarrow{\mathcal{D}}(e_1)\ in \\ let\ b = \overrightarrow{\mathcal{D}}(e_2)\ in \\ (\pi_1(a) + \pi_1(b), \pi_2(a) + \pi_2(b)) \end{cases}$$

The rules for cos, exp are similar to the one for sin.

We extend $\overrightarrow{\mathcal{D}}$ on contexts in the obvious way by

$$\overrightarrow{\mathcal{D}}(\{x_1 : \tau_1, \ldots, x_n : \tau_n\}) := \{\overrightarrow{\mathcal{D}}(x_1 : \tau_1), \ldots, \overrightarrow{\mathcal{D}}(x_n : \tau_n)\}$$

Example 1. Consider a simple term $x, y : \mathbb{R} \vdash sin(x * y) : \mathbb{R}$ which we call f. For the sake of readability, we write $\overrightarrow{\mathcal{D}}(x) = (x, x')$ and $\overrightarrow{\mathcal{D}}(y) = (y, y')$. Then by applying the macro $\overrightarrow{\mathcal{D}}$ to the term f we get $x, x' : \mathbb{R}, y, y' : \mathbb{R} \vdash \big(sin(x * y), (x * y' + x' * y) * cos(x * y)\big)$. This new term contains all the information of the partial derivatives of f. We obtain $\frac{\partial f}{\partial x}$ by replacing (x', y') by $(1, 0)$ and $\frac{\partial f}{\partial y}$ by replacing (x', y') by $(0, 1)$. This is the dual number representation for f. □

From there, it is easy to prove:

Lemma 1 (Well-typedness $\overrightarrow{\mathcal{D}}$)**.** *If* $\Gamma \vdash e : \tau$ *then* $\overrightarrow{\mathcal{D}}(\Gamma) \vdash \overrightarrow{\mathcal{D}}(e) : \overrightarrow{\mathcal{D}}(\tau)$.

Lemma 2 (Functoriality $\overrightarrow{\mathcal{D}}$)**.** *If* $\Gamma, y : \tau \vdash e_1 : \sigma$ *and* $\Gamma \vdash e_2 : \tau$ *then* $\overrightarrow{\mathcal{D}}(\Gamma) \vdash \overrightarrow{\mathcal{D}}(e_1[e_2/y]) = \overrightarrow{\mathcal{D}}(e_1)[\overrightarrow{\mathcal{D}}(e_2)/y] : \overrightarrow{\mathcal{D}}(\sigma)$.

Finally it's also not hard to prove correctness of $\overrightarrow{\mathcal{D}}$:

Theorem 3 (Correctness of $\overrightarrow{\mathcal{D}}$)**.** *For all term* $x_1, \ldots, x_n : \mathbb{R} \vdash e : \mathbb{R}$, $\overrightarrow{\mathcal{D}}(e)$ *is the dual number representation of* e.

As seen in the example above, this means that for all j, the substitution $\overrightarrow{\mathcal{D}}(e)[0/x'_{i\neq j}, 1/x'_j]$ computes $\frac{\partial |e|}{\partial x_j}$ where $|e| : \mathbb{R}^n \to \mathbb{R}$ is the differentiable function denoting e.

2.3 Existing work

We now give some detail about some of the recent work in differentiable programming.

In [Pearlmutter and Siskind, 2008] the authors define a non-local transformation J on all lambda-terms for doing reverse-mode AD in an efficient way. They are possibly the first to introduce an efficient AD as a source-code transformation on a lambda-calculus in their paper.

The papers [Shaikhha et al., 2018, Wang et al., 2019, Huot et al., 2020, Brunel et al., 2019] use a language that is similar to the one of the previous section and AD is also seen as a macro (source-code transformation). A bit more precisely, [Shaikhha et al., 2018] uses a higher-order domain-specific language with some array constructs and aggressive optimization rules for forward-mode AD. [Wang et al., 2019] focuses on reverse-mode AD in an untyped higher-order language with recursion and some unpure features. In particualr, they use delimited continuations for dealing with the reverse pass of reverse-mode in an elegant way, and they have an efficient implementation in Scala. [Huot et al., 2020] uses a higher-order functional language with variant types and inductive types but no recursion. Finally, [Brunel et al., 2019] uses a variant of lambda-calculus, called the linear substitution calculus, which is a typed lambda-calculus augmented with a negation which

accounts for returning type, which they use for the reverse pass of reverse-mode AD, which is also seen as a macro.

Finally, in [Abadi and Plotkin, 2019] the authors see differentiation as a typing rule with specific reduction rules to actually compute the derivatives on a lambda-term. Roughly, if we denote $D(e)$ the result of this typing rule on e, the goal of these reduction rules is to eliminate D by propagating it to sub-terms of e until it reaches some primitive functions like *cos*. For instance $D(if\ e_1\ then\ e_2\ else\ e_2) \to if\ e_1\ then\ D(e_2)\ else\ D(e_2)$ and $D(sin) \to cos$. This is related to the eager-mode implementation of TensorFlow [Agrawal et al., 2019].

3 Literature review: Beyond Automatic Differentiation

3.1 Idea

Automatic differentiation aims at computing gradients efficiently. A first observation was that in the case we are interested in computing the full gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$, forward-mode AD needs n-passes while reverse-mode only needs one. Such functions appear a lot as loss functions for neural networks (NN) and more generally in ML. This explains the popularity of the back-propagation algorithm as learning is often done by doing gradient-descent which computes a lot of gradients of such functions.

Back-propagation came as a sort of refinement of forward-mode coming the realisation that forward-mode was not efficient to compute the full gradient of a function $f : \mathbb{R}^n \to \mathbb{R}$ for a large n, and ML uses a lot of them, e.g. as loss functions. In the same vein, ML has needs for fancier notions of gradients such as computing gradients on noisy data, doing stochastic optimization, finding approximate gradients. There are also some algorithms using the specific structure of the problem considered which look very much like gradient-descent algorithms, e.g. Gradient boosting [Friedman, 2002] and Xgboost [Chen et al., 2015]. The design of AD for ML purposes has also been the topic of some recent papers, e.g. [van Merrienboer et al., 2018a, van Merrienboer et al., 2018b, Baydin et al., 2017].

The need for highly performant gradient computations is thus not enough. Usually, as gradients are used in the gradient-descent algorithm, the latter has received a lot of attention and several now well-known ideas emerged as to how to improve its convergence rate. In a sense the gradient gives good information as it is relatively easy to compute and gives a direction for improvement. But it is also often too naive as well hence the modification of the gradient descent algorithm to make use of gradients in a less naive way. We detail some of these ideas in the next section. Then, in the third section we briefly talk about variational inference which comes from Bayesian inference but uses optimization via gradient descent. We conclude this whole section with a small section on probabilistic programming.

3.2 Some common changes to the Gradient Descent algorithm

Gradient descent (GD) is an algorithm to find the minumum of a function: given differentiable a function $f : A \to \mathbb{R}$, it aims at finding $x \in argmin\, f$. One way to reach a local minimum of f from a point y is to follow to always locally move to the lowest point on the curve of f. Under some good conditions a global minimum will be reached but these conditions are often too strict in practice. The gradient of f at y gives the direction of the steepest descent at y. Hence gradient descent is an iterative algorithm which computes gradients at every iteration. It is used a lot in supervised learning for loss functions. The loss measures the difference between the prediction of a neural network (NN) and the expected outcome, and GD is used to change the parameters of the NN to improve prediction.

However, computing a whole gradient at each iteration of the GD algorithm may be costly, and the number of steps might to find a local minimum might be too large, so several improvements have been considered to improve the algorithm.

3.2.1 Adding a momentum term

See for instance [Qian, 1999]. Given a neural network with weights w and error function $E(w)$, the idea is to replace the usual steepest descent at time t

$$\Delta w_t = -\varepsilon \nabla_w E(w_t)$$

where $\varepsilon > 0$ is a hyper-parameter, the learning rate or step-size.

The new update of the weights at time step t becomes:

$$\Delta w_t = -\varepsilon \nabla_w E(w_t) + p\Delta w_{t-1}$$

where p is another hyper-parameter. The intuition is that the velocity at t represented by Δw_t depends on the hills and valleys on which we move (represented by $\nabla_w E(w_t)$) but also by the previous velocity, which hence acts as momentum.

3.2.2 Varying the learning rate

A common improvement to the convergence speed of gradient descent is performed by having a changing learning rate. A common way (called Newton or quasi-Newton methods) is to include the inverse Hessian or an approximation of it. That is, we obtain

$$\Delta w_t = -(\nabla_w^2 E(w_t))^{-1} \nabla_w E(w_t)$$

Given that computing and inverting the Hessian matrix is computationnaly expensive, an approximate Hessian is often used (see e.g. [Becker et al., 1988]), for instance a diagonal approximation, in which case we get

$$\Delta w_{ij} = -\frac{\frac{\partial E}{\partial w_{ij}}}{\left| \frac{\partial^2 E}{\partial w_{ij}^2} \right|}$$

The idea is that the curvature of the curve of the function we're trying to find the minimal of may greatly influence the speed of convergence of GD, see e.g. [Martens, 2010]. For instance, if the space is locally almost flat in a big area, then the gradient will be small all along this almost-flat part of the curve. However, in this case the Hessian is almost zero as well, so multiplying by the inverse Hessian allows for big leaps in this situation, which is the desired improvement.

Another kind of improvements comes from simulated annealing as in [Szu and Hartley, 1987, Tsallis and Stariolo, 1996]. If the problem is convex (single minimum), any gradient descent method could solves the problem. But if the energy is nonconvex (multiple extrema) the solution requires more sophisticated methods, since a gradient descent procedure could easily trap the system in a local minimum (instead of one of the global minima we are looking for). In this technique, one or more artificial temperatures are introduced and gradually cooled, in complete analogy with the well known annealing technique frequently used in Metallurgy for making a molten metal to reach its crystalline state (global minimum of the thermodynamical en- ergy). This artificial temperature (or set of temperatures) acts as a source of stochasticity, extremely convenient for eventually detrapping from local minima.

3.2.3 Stochastic Gradient Descent

Again in the context of neural networks, the loss function E is often of the form $\sum_{1 \leq i \leq n} E_i$ and computing the gradient of E amounts to computing the n gradients of the E_i, which can be very expensive. The idea of stochastic gradient descent (SGD) is to only compute at each time t a small subset of these gradients, chosen uniformly at random. Then, under the right conditions, it can be shown to converge efficiently to a local minimum. However, by its more complicated nature than normal GD, SGD is harder to optimize. For instance, the addition of a momentum term to SGD is rather recent [Allen-Zhu, 2017, Kingma and Ba, 2014] as doing it in a naive way might increase variance or even cause SGD to diverge.

To sum up, a more general framework for talking about gradient descent algorithms and their optimizations and convergence proofs might be desirable, given the vast amount of variants of the algorithm currently existing. Because the main computation in a GD algorithm is still to compute gradients, a general framework for talking about gradients in a functional language is might be the right thing to look for. As these gradients might be higher-order of higher-order functions, iterated, including probabilities and approximations, that framework would be beyond what is currently called AD. A somewhat similar idea has for instance been pursued in [Kucukelbir et al., 2017, Ranganath et al., 2014].

3.3 Variational inference

We give a last example of where a GD algorithm is used: in the case of Bayesian inference. More specifically, variational inference consists in approximating the true posterior density $p(z \mid x)$ with a density $q_\theta(z)$ from which we can easily sample. We assume that $q_\theta(z)$ belongs to a family $\mathcal{D}$ which can sufficiently well approximate $p(z \mid x)$, and that is parametrized by θ. The goal is then to optimize on θ to minimize a certain distance between $p(z \mid x)$ and $q_\theta(z)$.

This optimization is done by gradient descent as well, but the subtlety is that in this case the gradient gets to be expressed as an integral, so Monte Carlo integration is performed to get an unbiased estimate of the gradient. The algorithm for GD in this case looks different still from all the previously mentioned ones. Because of the nature of the problem considered, the gradient has a specific shape that is not computable by usual AD.

Several papers try to minimize the variance of the estimate of the gradient, e.g. [Kucukelbir et al., 2017, Ranganath et al., 2014, Ruiz et al., 2016]. It would be nice still for instance to add an integral operator as a primitive in a language for differentiable programming and some approximation rules for computing it with generalized AD. This could be useful for including optimizations automatically performed by the AD algorithm which would not only be variance reduction techniques.

3.4 Probabilistic Programming

Another approach to learning is Bayesian inference (see e.g. [Box and Tiao, 2011]). While NN are conceptually easy to optimize by back-propagation, their expressivity is somewhat constrained in many ways. Indeed, the function space which can be reached by a NN is a parametric family of the parameters of the network, and hence can be very constrained and has no a priori guarantee to be anywhere close to the function its trying to learn. One way to make this family of functions richer is to add more parameters, hence the use of huge NN. Another important point is to have non-linear functions, and the non-linearity is introduced in NN by the activation functions (e.g. ReLu or sigmoid). One could say that on the opposite side of expressivity is Bayesian inference, which is however in general much harder to use in practice as computation time might be prohibitive. Thus, it is all about a trade-off between expressivity and practical computation time. For instance,

we could say that VI is somewhere in the middle.

In probabilistic programming [Bingham et al., 2019, Tran et al., 2016, Wood et al., 2014, Goodman et al., 2012, Carpenter et al., 2017, Gordon et al., 2014, Gaunt et al., 2016], programs become a way to specify probabilistic models for observed data, on top of which one can later do inference. This has been a source of inspiration for AI researchers, and has recently been gathering interest in the programming langage community [Borgström et al., 2016]. Probabilistic programming might thus somehow be seen as a potential generalization of differentiable programming in terms of expressivity for designing learning algorithms. Still, probabilistic programming lacks native differentiation and non trivial interactions between differentiable and probabilistic programming might be worth exploring. See [Lee et al., 2018, Lee et al., 2019] for recent papers related to this question.

4 Proposed work

In the following three subsections, I propose three possible directions for my work, ordered from the most well-defined questions and problems to the more speculative ones.

4.1 Proposition 1: Semantics for rich AD languages

Towards verification for rich lamguages for differentiable programming, I would like to explore more denotational semantics of such languages via category theory.

4.1.1 Denotational semantics a for first-order language

Consider the language from 2.2. It can be easily given a denotational semantics in the category of Cartesian spaces $\mathbb{R}^n$ for all n and smooth functions between them. The type $\mathbb{R}$ is interpreted as the object $\mathbb{R}$, cos, sin, exp via their respective smooth functions $\mathbb{R} \to \mathbb{R}$, etc.

We could extend the language with sum types $\tau + \tau$ and inductive types like Lists ($\mu\alpha.1 + \tau \times \alpha$) and give a denotational semantics in the category **Man** of smooth manifolds and smooth functions between them. Smooth manifolds are topological spaces locally homeomorphic to a Cartesian space, e.g. a sphere is a 2 dimentional manifold, locally homeomorphic to $\mathbb{R}^2$. These local homeomorphisms have to satisfy some extra compatibility conditions.

It is well known that neither **CartSp** nor **Man** admit function spaces, that is they are not Cartesian closed. See for instance in the appendix of [Huot et al., 2020] for a short proof. Therefore, if we are to give a denotational semantics to a higher-order language with smooth primitives, we need a Cartesian closed category that contains **CartSp** or **Man**.

4.1.2 Background: Denotational semantics for a higher-order language

There are several candidates for this. We chose in our recent paper [Huot et al., 2020] to base our work on *diffeological spaces* [Iglesias-Zemmour, 2013].

The key idea will be that a higher order function is called smooth if it sends smooth functions to smooth functions, meaning that we can never use it to build first order functions that are not smooth.

Definition 4. *A* diffeological space $(X, \mathcal{P}_X)$ *consists of a set* X *together with, for each* n *and each open subset* U *of* $\mathbb{R}^n$*, a set* $\mathcal{P}_X^U \subseteq [U \to X]$ *of functions, called* plots*, such that*

- *all constant functions are plots;*
- *if $f : V \to U$ is a smooth function and $p \in \mathcal{P}_X^U$, then $f; p \in \mathcal{P}_X^V$;*
- *if $\left(p_i \in \mathcal{P}_X^{U_i}\right)_{i \in I}$ is a compatible family of plots ($x \in U_i \cap U_j \Rightarrow p_i(x) = p_j(x)$) and $(U_i)_{i \in I}$ covers U, then the gluing $p : U \to X : x \in U_i \mapsto p_i(x)$ is a plot.*

We call a function $f : X \to Y$ between diffeological spaces *smooth* if, for all plots $p \in \mathcal{P}_X^U$, we have that $p; f \in \mathcal{P}_Y^U$. We write $\mathbf{Diff}(X, Y)$ for the set of smooth maps from X to Y. Smooth functions compose, and so we have a category **Diff** of diffeological spaces and smooth functions.

We can interpret a subset $Y \subseteq X$ of a diffeological space X as a *subspace* by taking the plots of X which factor over Y as its plots.

A diffeological space is thus a set equipped with structure. Many constructions of sets carry over straightforwardly to diffeological spaces. Hence, in particular, it is well-pointed, meaning that for $f, g : X \to Y$, we have that $f = g$ iff $f(x) = g(x)$ for all $x : 1 \to X$ In particular, morphisms are monos if and only if they are injective.

Example 2 (Cartesian diffeologies)**.** Each open subset U of $\mathbb{R}^n$ can be given the structure of a diffeological space by taking all the smooth functions $V \to U$ as $\mathcal{P}_U^V$. It is easily seen that smooth functions from $V \to U$ in the traditional sense coincide with smooth functions in the sense of diffeological spaces. Thus diffeological spaces have a profound relationship with ordinary calculus. □

In categorical terms, this gives a full embedding of **CartSp** in **Diff**.

Example 3 (Product diffeologies)**.** Given a family $(X_i)_{i \in I}$ of diffeological spaces, we can equip the product $\prod_{i \in I} X_i$ of sets with the *product diffeology* in which U-plots are precisely the functions of the form $(p_i)_{i \in I}$ for $p_i \in \mathcal{P}_{X_i}^U$. □

This gives us the categorical product in **Diff**.

Example 4 (Functional diffeology)**.** We can equip the set $\mathbf{Diff}(X, Y)$ of smooth functions between diffeological spaces with the *functional diffeology* in which U-plots consist of functions $f : U \to \mathbf{Diff}(X, Y)$ such that $(u, x) \mapsto f(u)(x)$ is an element of $\mathbf{Diff}(U \times X, Y)$. □

This specifies the categorical function object in **Diff**.

Diff is cocomplete and can thus interpret inductive types as well.

4.1.3 Proposed work: toward some rich features

The category **Diff** of diffeological spaces has quite a rich structure. It is a quasi-topos [Baez and Hoffnung, 2011] so in particular it is complete and cocopmplete. I would like to use this rich structure to study and give a denotational semantics to a rich higher-order language for differentiable programming with rich extra features.

First, as said above, it is a cocomplete category so admits inductive types. It can be shown to admit a certain partiality monad. This monad can be used to give a semantics to a language of partial functions. More importantly, it is a first step towards a semantics for a language including term recursion. This could be useful for proving properties of recursive NN [Socher et al., 2011].

A second monad of interest would be a probability monad on **Diff**. This might help us prove properties of recent algorithms which need a rich structure including both differentiation and distributions, e.g. variational auto-encoders [Kingma and Welling, 2013, Rezende et al., 2014].

Further proofs of interest would be justifying optimizations used for AD and specific algorithms tailored for computing higher-order derivatives. One example of the latter would be to prove correct the algorithms coming from the jet-based framework [Betancourt, 2018, Bettencourt et al., 2019] in a higher-order language. An example of the former would be to prove correct the efficient reverse-mode algorithm from [Pearlmutter and Siskind, 2008].

Ideally, this would lead to the design of new algorithms that could outperform state-of-the-art implementations computing higher-order derivatives. This could also happen for first-order gradients in contexts with rich algebraic redundant structure like NN, using the optimizations coming from the PL community and the fact that NN building blocks and a lof of operations on them are naturally higher-order.

4.2 Proposition 2: Languages for differentiable probabilistic higher-order languages

4.2.1 Motivation

In the quest of the design of better languages for statistics, category theory migth provide a very useful tool. First, it has already been used to give semantics to functionnal languages and more generally helped in their designs. Functionnal programming separates the pure part which computes a mathematical function, and the effects, e.g. memory management, non-determinism, etc. Arguably, statistics in programming can also be seen as pure mathematical functions with the unpure feature of randomness, which is a bit similar to weighted non-determinism, at least in the case of discrete probabilities. Category theory is also a powerful tool for dealing with sophisticated mathematical abstractions. The use of category theory is thus a well-suited tool for the design, analysis and correctness of probabilistic programming.

Similarly, category theory has shown to be a good tool for emcompassing generalised notions of derivatives, e.g. the category of diffeological spaces can serve as a denotational semantics for a higher-order language with smooth primitives and differentiation as a source-code transformation.

What's more, differentiation and probabilities tend to be used more and more together to reach a tradeoff, e.g. between the expressiveness of Bayesian inference and GD for NN. This can be seen in VI as we discussed before, but also in some of the many variants of Hamiltonian Monte Carlo. Therefore, I propose to study and design idealized languages for statistics with differentiation using category theory.

4.2.2 Background: Quasi-borel spaces

An interesting line of research I would like to explore further comes from the similarity between **Diff** and the category **Qbs** of quasi-Borel spaces (QBS) [Heunen et al., 2017], which we review briefly.

Proposition 5. *For a measurable space (X, Σ_X) the following are equivalent:*

- *(X, Σ_X) is a retract of $(\mathbb{R}, \Sigma_{\mathbb{R}})$, that is, there exist measurable $X \xrightarrow{f} \mathbb{R} \xrightarrow{g} X$ such that $g \circ f =$ id$X$;*
- *(X, Σ_X) is either measurably isomorphic to $(\mathbb{R}, \Sigma_{\mathbb{R}})$ or countable and discrete;*
- *X has a complete metric with a countable dense subset and Σ_X is the least σ-algebra containing all open sets.*

When (X, Σ_X) satisfies any of the above conditions, we call it *standard Borel space*. These spaces play an important role in probability theory because they enjoy properties that do not hold for general measurable spaces, such as the existence of conditional probability kernels and de Finetti's theorem for exchangeable random processes.

Besides $\mathbb{R}$, another popular uncountable standard Borel space is $(0, 1)$ with the σ-algebra $\{U \cap (0, 1) \mid U \in \Sigma_{\mathbb{R}}\}$. As the above proposition indicates, these spaces are isomorphic by, for instance, $\lambda r. \frac{1}{(1+e^{-r})} : \mathbb{R} \to (0, 1)$.

Definition 6. *A* quasi-Borel space *is a set X together with a set $M_X \subseteq [\mathbb{R} \to X]$ satisfying:*

- $\alpha \circ f \in M_X$ *if* $\alpha \in M_X$ *and* $f \colon \mathbb{R} \to \mathbb{R}$ *is measurable;*
- $\alpha \in M_X$ *if* $\alpha \colon \mathbb{R} \to X$ *is constant;*
- *if* $\mathbb{R} = \biguplus_{i \in \mathbb{N}} S_i$*, with each set* S_i *Borel, and* $\alpha_1, \alpha_2, \ldots \in M_X$*, then* β *is in* M_X*, where* $\beta(r) = \alpha_i(r)$ *for* $r \in S_i$*.*

Example 5. For every measurable space (X, Σ_X), let M_{Σ_X} be the set of measurable functions $\mathbb{R} \to X$. Thus M_{Σ_X} is the set of X-valued random variables. In particular: $\mathbb{R}$ itself can be considered as a quasi-Borel space, with $M_{\mathbb{R}}$ the set of measurable functions $\mathbb{R} \to \mathbb{R}$; the two-element discrete space 2 can be considered as a quasi-Borel space, with M_2 the set of measurable functions $\mathbb{R} \to 2$, which are exactly the characteristic functions of the Borel sets. □

The categories **Diff** and **Qbs** both arise as categories of concrete sheaves on a site: the category of Cartesian spaces **CartSp** in the case of **Diff** and the category of standard Borel spaces **Sbs** in the case of **Qbs**. This process makes **Diff** and **Qbs** quasi-toposes. In particular they are both complete, cocomplete, Cartesian closed, and contain fully and faithfully **CartSp** and **Sbs** respectively. In addition, **CartSp** is a non-full subcategory of **Sbs** as every smooth function is measurable. Studying their contructions and links further might help design a denotational semantics for a probabilistic language with differentiation operators, or for a language for differentiable programming with non-differentiable parts. For instance the notion of almost-everywhere differentiable function could be explored, and in particular piece-wise differentiable function.

4.2.3 Proposed work

For this line of work, some ambitious goals would be to:

- Develop new categorical tools for the study of smoothness and randomness. Then use it to prove correctness of a variety of algorithms phrased in a functionnal language. Generalise these algorithms and find new optimizations coming from categorical insights.
- Better understand the use of density functions. They are often useful when designing algorithms as a calculational tool but maybe a more abstract point of view would be preferable and would unravel new ways of doing these algorithms. In particular, when the density is intractable as in implicit generative models like GAN [Goodfellow et al., 2014, Radford et al., 2015].
- Develop a good formalism for non-parametric learning. Recent work [Liutkus et al., 2018] has used optimal transport theory and partial differential equations theory for generative modelling. This is very likely to just be the tip of iceberg and a theory for non-parametric generative modelling generalising this work would lead to new exciting results in ML. In particular, this might be a key step toward general artifical intelligence (AGI) [Goertzel and Pennachin, 2007].

4.3 Proposition 3: Reparametrizations

4.3.1 Idea: reparametrization

A reparametrization of a distribution μ on X consists in a pair of a deterministic mesurable function $f : Y \to X$ and a distribution ν such that $\mu = f_*\nu$. The last operation is the pushforward of ν by f and defined on measurable elements $A \subseteq X$ by $f_*\nu(A) = \nu(f^{-1}(A))$.

In practice, μ is a complicated distribution of interest and ν a simpler distribution from which it is easy or at least easier to sample from. The equation $\mu = f_*\nu$ means that to sample following μ can be achieved by sampling following ν and then applying f. Also, as distributions are often in practice given by their density functions (likelihood functions), reparametrization is more tricky as densities are ill-behaved w.r.t. pushforward. For instance, a sufficient condition for this to work is if the cumulative distributive function (CDF) is in closed form.

4.3.2 Reparametrization tricks

The reparametrization tricks (see e.g. [Lee et al., 2018, Kingma and Welling, 2014, Nalisnick and Smyth, 2016, Ruiz et al., 2016]) are well-chosen reparametrizations that are used in different situations. We summarize the main ones in the following three categories:

- one-liners
- backpropagation
- variance reduction

One liners give us simple toolds to generate random variates in one line of code [Shakir, 2015]. Three popular appproaches are: inversion methods, which require the inverse CDF as the transformation; polar methods. The Box-Muller transform is derived this way. It takes two samples from the uniform distribution on $[0, 1]$ and maps them to two standard, normally distributed samples. Finally, coordinate transformation methods, e.g. location-scale transformations. For instance one can sample from a Gaussian $\mathcal{N}(\mu, RR^T)$ by sampling from the standard Gaussian $\varepsilon N(0, 1)$ and using the transformation $\varepsilon \mapsto \mu + R\varepsilon$.

Reparametrization tricks [rep, 2016] are used to backpropagate through a random node. The idea is to use the reparametrization to isolate the source of randomness from the parameters, and it then becomes clear how to backpropagate. For instance, when computing $\nabla_\sigma(z), zN(\mu, \sigma^2)$, we can use the reparametrizationfrom above $z = \mu + \sigma\varepsilon, \varepsilon N(0, 1)$ and the gradient becomes

$$\nabla_\sigma(z) = \nabla_\sigma(\mu + \sigma\varepsilon) = 0 + 1 * \varepsilon + \sigma * 0 = \varepsilon$$

There are a number of recent uses of random variate reparametrization in ML to develop scalable VI in deep generative models and Gaussian processes [rep, 2016]. Unbiased estimates of gradients are computed and reparametrization contribute to low variance estimates. I have not seen good intuitive explanation for this variance reduction techniques yet. The idea seems to be that these estimators provide an implicit tracking of dependencies between parameters —a provenance tracking— which contributes to the low variance.

4.3.3 Proposed work

I thus propose to try to generalise the reparametrization tricks using category theory and probabilistic programming, in the same way that probabilistic programming already provides a convenient formalism to specify probabilistic models for observed data. There are already some known

generalisation of differentible non-centered parametrizations (DNCP) [Kingma and Welling, 2014] and recent work on reparametrisation tricks [Lee et al., 2018]. Category theory might help explain them in a clearer and more canonical way. Furthermore, re-parametrizations in VI are often asked to be differentiable functions. Following the previous line of work I proposed, this might enable to relax the conditions to almost everywhere differentiable functions, and help us understand the right contexts in which reparametrization can be performed and its benefits.

The goals that can be achieved through this are:

- Discover a new reparametrization trick or a family of them. This will widen their applicability and improve the state-of-the-art performance. For instance, the widely used β and γ distributions do not have their inverse CDF in closed form and more involved techniques are used for doing reparametrisation for them [Ruiz et al., 2016, Nalisnick and Smyth, 2016]. Implementations in collaboration will be done and experiments carried to test efficiency.
- Build new VAEs or even new family of algorithms like VI. This would in particular broaden the field of learning in ML. A lot of ideas from VI are for instance being imported for reinforcement learning [Sutton et al., 1998].
- Find ways to better encapsulate the reparametrization tricks in deep probabilistic programming languages. Make a prototype idealized new programming language with programs that are easier to read, write, maintain and experiment with.

5 Summary and Risk Assessment

5.1 Summary

I surveyed the state of the art in differentiable programming and of some common extensions of automatic differentiation that are used in gradient-descent algorithms to make it more performent. I highlighted that these extensions naturally tighten the gap between differentiable programming and probabilistic programming, as probabilities can be used to improve gradient descent, e.g. stochastic gradient descent. Conversely gradient descent can be used in Bayesian inference, e.g. for variational inference.

I proposed three possible works I plan to pursue during my PhD. First, denotational semantics for rich higher-order differentiable languages, extending my work [Huot et al., 2020]. Second, I hinted one possible way the similarities between probabilistic programming and differentiable programming could be further explored via category theory. The category of diffeological spaces which serves as a model for a higher-order language of smooth functions and the category of quasi-Borel spaces which serves as a model for a higher-order language of random elements are categories of concrete sheaves.

Lastly, I proposed to generalize reparametrization tricks, which are powerful variance reduction techniques used for instance in variational inference, using category theory. These tricks also allow backpropagation on parameters with noise, allow more efficient sampling.

5.2 Risk Assessment

The first direction which consists on building on my previous work [Huot et al., 2020] is more clearly defined and therefore less risky. As a result, my priority will be continuing my work on AD, and in parallel or subsequently explore the links between probabilistic programming and differentiable programming, as well as generalisation and automation of the reparametrization tricks. I currently plan to allocate two thirds of my time of the Work 1, and the remaining third on the exploration of Work 2. I have some preliminary results in these, especially proving a Jet-based method [Betancourt, 2018] for efficient computation of higher-order derivatives. Thus I first plan to focus more on these, and explore the third Work proposal as opportunities arise or after I think enough is done on the Work 1.

References

[rep, 2016] (2016). How does the reparameterization trick for VAEs work and why is it important? `https://stats.stackexchange.com/questions/199605/how-does-the-reparameterization-trick-for-vaes-work-and-why-is-it-important`.

[Abadi et al., 2016] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 265–283.

[Abadi and Plotkin, 2019] Abadi, M. and Plotkin, G. D. (2019). A simple differentiable programming language. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–28.

[Abdul Hamid et al., 2011] Abdul Hamid, N., Mohd Nawi, N., and Ghazali, R. (2011). The effect of adaptive gain and adaptive momentum in improving training time of gradient descent back propagation algorithm on classification problems.

[Agrawal et al., 2019] Agrawal, A., Modi, A. N., Passos, A., Lavoie, A., Agarwal, A., Shankar, A., Ganichev, I., Levenberg, J., Hong, M., Monga, R., et al. (2019). Tensorflow eager: A multi-stage, Python-embedded DSL for machine learning. *arXiv preprint arXiv:1903.01855*.

[Allen-Zhu, 2017] Allen-Zhu, Z. (2017). Katyusha: The first direct acceleration of stochastic gradient methods. *The Journal of Machine Learning Research*, 18(1):8194–8244.

[Baez and Hoffnung, 2011] Baez, J. and Hoffnung, A. (2011). Convenient categories of smooth spaces. *Transactions of the American Mathematical Society*, 363(11):5789–5825.

[Baydin and Pearlmutter, 2014] Baydin, A. G. and Pearlmutter, B. A. (2014). Automatic differentiation of algorithms for machine learning. *arXiv preprint arXiv:1404.7456*.

[Baydin et al., 2017] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., and Siskind, J. M. (2017). Automatic differentiation in machine learning: a survey. *The Journal of Machine Learning Research*, 18(1):5595–5637.

[Becker et al., 1988] Becker, S., Le Cun, Y., et al. (1988). Improving the convergence of back-propagation learning with second order methods. In *Proceedings of the 1988 connectionist models summer school*, pages 29–37.

[Bengio et al., 1994] Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.

[Betancourt, 2018] Betancourt, M. (2018). A geometric theory of higher-order automatic differentiation. *arXiv preprint arXiv:1812.11592*.

[Bettencourt et al., 2019] Bettencourt, J., Johnson, M. J., and Duvenaud, D. (2019). Taylor-Mode Automatic Differentiation for Higher-Order Derivatives in JAX.

[Bingham et al., 2019] Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. (2019). Pyro: Deep universal probabilistic programming. *The Journal of Machine Learning Research*, 20(1):973–978.

[Borgström et al., 2016] Borgström, J., Dal Lago, U., Gordon, A. D., and Szymczak, M. (2016). A lambda-calculus foundation for universal probabilistic programming. *ACM SIGPLAN Notices*, 51(9):33–46.

[Bottou, 2010] Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

[Bottou, 2012] Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer.

[Box and Tiao, 2011] Box, G. E. and Tiao, G. C. (2011). *Bayesian inference in statistical analysis*, volume 40. John Wiley & Sons.

[Brunel et al., 2019] Brunel, A., Mazza, D., and Pagani, M. (2019). Backpropagation in the Simply Typed Lambda-calculus with Linear Negation. *arXiv preprint arXiv:1909.13768*.

[Carpenter et al., 2017] Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of statistical software*, 76(1).

[Carpenter et al., 2015] Carpenter, B., Hoffman, M. D., Brubaker, M., Lee, D., Li, P., and Betancourt, M. (2015). The Stan math library: Reverse-mode automatic differentiation in C++. *arXiv preprint arXiv:1509.07164*.

[Chen et al., 2014] Chen, T., Fox, E., and Guestrin, C. (2014). Stochastic gradient hamiltonian monte carlo. In *International conference on machine learning*, pages 1683–1691.

[Chen et al., 2015] Chen, T., He, T., Benesty, M., Khotilovich, V., and Tang, Y. (2015). Xgboost: extreme gradient boosting. *R package version 0.4-2*, pages 1–4.

[Cockett and Lemay, 2019] Cockett, J. R. B. and Lemay, J.-S. (2019). Integral categories and calculus categories. *Mathematical Structures in Computer Science*, 29(2):243–308.

[Cruttwell et al., 2019] Cruttwell, G., Gallagher, J., and MacAdam, B. (2019). Towards formalizing and extending differential programming using tangent categories. In *Proc. ACT 2019*.

[Cruttwell, 2017] Cruttwell, G. S. (2017). Cartesian differential categories revisited. *Mathematical Structures in Computer Science*, 27(1):70–91.

[Friedman, 2002] Friedman, J. H. (2002). Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4):367–378.

[Gaunt et al., 2016] Gaunt, A. L., Brockschmidt, M., Singh, R., Kushman, N., Kohli, P., Taylor, J., and Tarlow, D. (2016). Terpret: A probabilistic programming language for program induction. *arXiv preprint arXiv:1608.04428*.

[Girolami and Calderhead, 2011] Girolami, M. and Calderhead, B. (2011). Riemann manifold langevin and hamiltonian monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.

[Goertzel and Pennachin, 2007] Goertzel, B. and Pennachin, C. (2007). *Artificial general intelligence*, volume 2. Springer.

[Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

[Goodman et al., 2012] Goodman, N., Mansinghka, V., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. (2012). Church: a language for generative models. *arXiv preprint arXiv:1206.3255*.

[Gordon et al., 2014] Gordon, A. D., Graepel, T., Rolland, N., Russo, C., Borgstrom, J., and Guiver, J. (2014). Tabular: a schema-driven probabilistic programming language. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 321–334.

[Griewank et al., 1989] Griewank, A. et al. (1989). On automatic differentiation. *Mathematical Programming: recent developments and applications*, 6(6):83–107.

[Heunen et al., 2017] Heunen, C., Kammar, O., Staton, S., and Yang, H. (2017). A convenient category for higher-order probability theory. In *2017 32nd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–12. IEEE.

[Hoffman et al., 2013] Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347.

[Hoffman and Gelman, 2014] Hoffman, M. D. and Gelman, A. (2014). The No-U-Turn sampler: adaptively setting path lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623.

[Huot and Staton, 2019] Huot, M. and Staton, S. (2019). Quantum channels as a categorical completion. In *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 1–13. IEEE.

[Huot et al., 2020] Huot, M., Staton, S., and Vákár, M. (2020). Correctness of Automatic Differentiation via Diffeologies and Categorical Gluing. *arXiv preprint arXiv:2001.02209*.

[Iglesias-Zemmour, 2013] Iglesias-Zemmour, P. (2013). *Diffeology*, volume 185. American Mathematical Soc.

[Johnson and Zhang, 2013] Johnson, R. and Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323.

[Kelly et al., 2016] Kelly, R., Pearlmutter, B. A., and Siskind, J. M. (2016). Evolving the Incremental {\lambda} calculus into a model of forward automatic differentiation (ad). *arXiv preprint arXiv:1611.03429*.

[Kingma and Welling, 2014] Kingma, D. and Welling, M. (2014). Efficient gradient-based inference through transformations between bayes nets and neural nets. In *International Conference on Machine Learning*, pages 1782–1790.

[Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

[Kingma and Welling, 2013] Kingma, D. P. and Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

[Kucukelbir et al., 2017] Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. (2017). Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1):430–474.

[LeCun, 2018] LeCun, Y. (2018). Deep learning est mort. vive differentiable programming.

[Lee et al., 2019] Lee, W., Yu, H., Rival, X., and Yang, H. (2019). Towards verified stochastic variational inference for probabilistic programs. *Proceedings of the ACM on Programming Languages*, 4(POPL):1–33.

[Lee et al., 2018] Lee, W., Yu, H., and Yang, H. (2018). Reparameterization gradient for non-differentiable models. In *Advances in Neural Information Processing Systems*, pages 5553–5563.

[Liutkus et al., 2018] Liutkus, A., Şimşekli, U., Majewski, S., Durmus, A., and Stöter, F.-R. (2018). Sliced-Wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions. *arXiv preprint arXiv:1806.08141*.

[Mandt et al., 2017] Mandt, S., Hoffman, M. D., and Blei, D. M. (2017). Stochastic gradient descent as approximate Bayesian inference. *The Journal of Machine Learning Research*, 18(1):4873–4907.

[Manzyuk, 2012] Manzyuk, O. (2012). A simply typed λ-calculus of forward automatic differentiation. *Electronic Notes in Theoretical Computer Science*, 286:257–272.

[Manzyuk et al., 2012] Manzyuk, O., Pearlmutter, B. A., Radul, A. A., Rush, D. R., and Siskind, J. M. (2012). Confusion of tagged perturbations in forward automatic differentiation of higher-order functions. *arXiv preprint arXiv:1211.4892*.

[Martens, 2010] Martens, J. (2010). Deep learning via Hessian-free optimization. In *ICML*, volume 27, pages 735–742.

[Nalisnick and Smyth, 2016] Nalisnick, E. and Smyth, P. (2016). Stick-breaking variational autoencoders. *arXiv preprint arXiv:1605.06197*.

[Olah, 2015] Olah, C. (2015). Neural networks, types, and functional programming.

[Paisley et al., 2012] Paisley, J., Blei, D., and Jordan, M. (2012). Variational Bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*.

[Paszke et al., 2017] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic differentiation in pytorch.

[Pearlmutter and Siskind, 2008] Pearlmutter, B. A. and Siskind, J. M. (2008). Reverse-mode AD in a functional framework: Lambda the ultimate backpropagator. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(2):7.

[Qian, 1999] Qian, N. (1999). On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151.

[Radford et al., 2015] Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

[Ranganath et al., 2014] Ranganath, R., Gerrish, S., and Blei, D. (2014). Black box variational inference. In *Artificial Intelligence and Statistics*, pages 814–822.

[Rehman and Nawi, 2011] Rehman, M. Z. and Nawi, N. M. (2011). The effect of adaptive momentum in improving the accuracy of gradient descent back propagation algorithm on classification problems. In *International Conference on Software Engineering and Computer Systems*, pages 380–390. Springer.

[Rezende et al., 2014] Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*.

[Ruiz et al., 2016] Ruiz, F. R., AUEB, M. T. R., and Blei, D. (2016). The generalized reparameterization gradient. In *Advances in neural information processing systems*, pages 460–468.

[Saul and Jordan, 1996] Saul, L. K. and Jordan, M. I. (1996). Exploiting tractable substructures in intractable networks. In *Advances in neural information processing systems*, pages 486–492.

[Ścibior et al., 2017] Ścibior, A., Kammar, O., Vákár, M., Staton, S., Yang, H., Cai, Y., Ostermann, K., Moss, S. K., Heunen, C., and Ghahramani, Z. (2017). Denotational validation of higher-order Bayesian inference. *Proceedings of the ACM on Programming Languages*, 2(POPL):60.

[Shaikhha et al., 2018] Shaikhha, A., Fitzgibbon, A., Vytiniotis, D., Jones, S. P., and Koch, C. (2018). Efficient differentiable programming in a functional array-processing language. *arXiv preprint arXiv:1806.02136*.

[Shakir, 2015] Shakir, M. (2015). Machine learning trick of the Day (4): Reparameterisation Tricks. `http://blog.shakirm.com/2015/10/machine-learning-trick-of-the-day-4-reparameterisation-tricks/#381_1`.

[Shang et al., 2018] Shang, F., Zhou, K., Liu, H., Cheng, J., Tsang, I. W., Zhang, L., Tao, D., and Jiao, L. (2018). VR-SGD: A simple stochastic variance reduction method for machine learning. *IEEE Transactions on Knowledge and Data Engineering*, 32(1):188–202.

[Shao and Zheng, 2009] Shao, H. and Zheng, G. (2009). A new BP algorithm with adaptive momentum for FNNs training. In *2009 WRI Global Congress on Intelligent Systems*, volume 4, pages 16–20. IEEE.

[Siskind and Pearlmutter, 2005] Siskind, J. M. and Pearlmutter, B. A. (2005). Perturbation confusion and referential transparency: Correct functional implementation of forward-mode AD.

[Socher et al., 2011] Socher, R., Lin, C. C., Manning, C., and Ng, A. Y. (2011). Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136.

[Souriau, 1980] Souriau, J.-M. (1980). Groupes différentiels. In *Differential geometrical methods in mathematical physics*, pages 91–128. Springer.

[Sutton et al., 1998] Sutton, R. S., Barto, A. G., et al. (1998). *Introduction to reinforcement learning*, volume 135. MIT press Cambridge.

[Swanston et al., 1994] Swanston, D., Bishop, J., and Mitchell, R. J. (1994). Simple adaptive momentum: new algorithm for training multilayer perceptrons. *Electronics Letters*, 30(18):1498–1500.

[Szu and Hartley, 1987] Szu, H. and Hartley, R. (1987). Fast simulated annealing. *Physics letters A*, 122(3-4):157–162.

[Tran et al., 2016] Tran, D., Kucukelbir, A., Dieng, A. B., Rudolph, M., Liang, D., and Blei, D. M. (2016). Edward: A library for probabilistic modeling, inference, and criticism. *arXiv preprint arXiv:1610.09787*.

[Tsallis and Stariolo, 1996] Tsallis, C. and Stariolo, D. A. (1996). Generalized simulated annealing. *Physica A: Statistical Mechanics and its Applications*, 233(1-2):395–406.

[van Merrienboer et al., 2018a] van Merrienboer, B., Breuleux, O., Bergeron, A., and Lamblin, P. (2018a). Automatic differentiation in ml: Where we are and where we should be going. In *Advances in neural information processing systems*, pages 8757–8767.

[van Merrienboer et al., 2018b] van Merrienboer, B., Moldovan, D., and Wiltschko, A. (2018b). Tangent: Automatic differentiation using source-code transformation for dynamically typed array programming. In *Advances in Neural Information Processing Systems*, pages 6256–6265.

[Wang et al., 2018] Wang, F., Decker, J., Wu, X., Essertel, G., and Rompf, T. (2018). Backpropagation with continuation callbacks: foundations for efficient and expressive differentiable programming. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pages 10201–10212. Curran Associates Inc.

[Wang et al., 2019] Wang, F., Zheng, D., Decker, J., Wu, X., Essertel, G. M., and Rompf, T. (2019). Demystifying differentiable programming: Shift/reset the penultimate backpropagator. *Proceedings of the ACM on Programming Languages*, 3(ICFP):1–31.

[Welling and Teh, 2011] Welling, M. and Teh, Y. W. (2011). Bayesian learning via stochastic gradient Langevin dynamics. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 681–688.

[Wood et al., 2014] Wood, F., Meent, J. W., and Mansinghka, V. (2014). A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032.