REVIEW: ORDINARY DIFFERENTIAL EQUATIONS FOR DEEP LEARNING

A PREPRINT

Xinshi Chen

October 28, 2019

Contents

1 Introduction

Deep neural network is one of the most popular models in many machine learning tasks. To better understand and improve the behavior of neural network, a recent line of works bridged the connection between ordinary differential equations (ODEs) and deep neural networks (DNNs). The connections are made in two folds:

- View DNN as ODE discretization;
- View the training of DNN as solving an optimal control problem.

The former connection motivates people either to design neural architectures based on ODE discretization schemes or to replace DNN by a continuous model charaterized by ODEs. Several works demonstrated distinct advantages of using continuous model (ODE) instead of traditional DNN in some specific applications [Grathwohl et al., 2019, Zhang et al., 2018, Chen et al., 2019]. The latter connection is inspiring. Based on Pontryagin's maximum principle, which is popular in the optimal control literature, some developed new optimization methods for training neural networks and some developed algorithms to train the infinite-deep continuous model with low memory-cost.

This paper is organized as follows: In Section 2, the relation between neural architecture and ODE discretization is introduced. Some architectures are not motivated by ODE, but they are latter found to be associated with some specific discretization schemes. Some architectures are designed based on ODE discretization and expected to achieve some special properties. Section 3 formulate the optimization problem where a traditional neural network is replaced by a continuous model (ODE). The formulated optimization problem is an optimal control problem. Therefore, two different types of controls will also be discussed in this section. In Section 4, we will discuss how we can utilize the optimization methods that are popular in optimal control literature to help the training of machine learning problems. Finally, two applications of using continuous model will be shown in Section 5 and 6 to demonstrate some of its advantages over traditional neural networks.

2 ODE for Neural Architecture

A line of works [Liao and Poggio, 2016, Li et al., 2017, Chen et al., 2018, Lu et al., 2018, Ruthotto and Haber, 2018, Weinan et al., 2019, Chang et al., 2019] has mentioned the relation between residual networks and discretization schemes of ODEs. More specifically, for an ODE in the form

$$dX(t)/dt = f(X, W), \tag{1}$$

its approximation using forward Euler discretization scheme is

$$X_{t+1} = X_t + hf(X_t, W_t), \tag{2}$$

which can be regarded as a generalization of residual networks [Liao and Poggio, 2016, Chen et al., 2018, Lu et al., 2018, Ruthotto and Haber, 2018, Chang et al., 2019]. Lu et al. [2018] showed some previous effective networks including PolyNet [Zhang et al., 2017] and FractalNet [Larsson et al., 2017] which are not motivated by ODE can also be interpreted as different discretizations of differential equations. After that, Lu et al. [2018] proposed LM-ResNet which corresponds to linear multi-step discretization scheme, Chang et al. [2019] proposed AntisymmetricRNN by utilizing stability analysis of ODE, Ruthotto and Haber [2018] proposed 2ndOrderCNN and HamiltonianCNN. A summary is given in table 1.

Table 1: Different deep networks and their associated ODE discretization schemes.

architecture	formula	discretization
ResNet [He et al., 2016]	$X_{t+1} = X_t + f(X_t)$ refer to only one previous point	Forward Euler
PolyNet [Zhang et al., 2017]	$X_{t+1} = X_t + f(X_t) + f(f(X_t))$ $\approx (I - f)^{-1}(X_t)$ implict scheme, stabler than forward Euler	Backward Euler
FractalNet [Larsson et al., 2017]	$f_{k+1}(X_t) = [(f_k \circ f_k)(X_t)] \bigoplus [\text{conv}(X_t)]$ take some intermediate steps to obtain a higher order method	Runge-Kutta
LM-ResNet [Lu et al., 2018]	$X_{t+1} = (1 - h_t)X_t + h_t X_{t-1} + f(X_t)$ use information from the previous step to calculate the next value	Linear multistep
2ndOrderCNN [Ruthotto and Haber, 2018]	$X_{t+1} = 2X_t - X_{t-1} + h_t^2 f(X_t)$ one type of multistep methods	Leapfrog method
AntisymmetricRNN [Chang et al., 2019]	$H_t = H_{t-1} + \varepsilon\sigma\left((W - W^\top)H_{t-1} + VX_t + b\right)$ Jacobian matrix has eigenvalues of zero real part	Forward Euler
RevNet [Gomez et al., 2017] HamiltonCNN [Ruthotto and Haber, 2018]	$Y_t = X_t + f(X_{t+1}), Y_{t+1} = X_{t+1} + g(Y_t)$ $Y_{t+1} = Y_t + f(X_t), X_{t+1} = X_t - g(Y_{t+1})$	a system of ODE

Despite from these interesting connections, a question arises: why are these architectures expected to be better? Although there are experimental evidences in these works which reveal some benefits of using these architectures, there is no clear explanation of the reason. The connection to ODE discretization also does not help much on our understanding, because numerical discretization schemes are typically designed to improve the accuracy of approximating the solution

of ODEs (e.g. Runge-Kutta and multi-step methods), while a neural network is typically not targeted at solving certain ODEs.

Among these works, there are two interesting angles. First, for the RevNet and HamiltonianCNN which are reversible, it allows us to avoid storage of intermediate network states, thus achieving higher memory efficiency. This is particularly important for very deep networks where memory limitation can hinder training. Second, Haber and Ruthotto [2017], Chang et al. [2019] introduced an intriguing idea of utilizing the stability theory of ODE. A stable network can be more robust to the perturbation (or noise) of the inputs. Unfortunately, the architecture proposed by Haber and Ruthotto [2017], Chang et al. [2019] are based on a set of conditions that in general can not ensure stability. Nevertheless, this ideas is still inspiring.

3 ODE for Continuous Modeling

Instead of using discretization of ODE to design neural architecture, one can directly use a continuous model. Many machine learning tasks are learning some function mappings $F : \mathbb{R}^d \to \mathbb{R}$ from data. The function mapping F is usually a neural network with unknown parameters. Now, one can consider the mapping from $\boldsymbol{x}$ to $F(\boldsymbol{x})$ as an evolution from initial state $X(0) = \boldsymbol{x}$ to the finial state $X(T)$, where the dynamics of $X(t)$ for $0 \le t \le T$ can be modeled by a differential equation, $dX/dt = f$. Learning the mapping F becomes learning the flow velocity $f : \Omega \to \mathbb{R}^d$ which determines continuous propagation of $X(t)$.

With the traditional neural network model replaced by a continuous ODE, a supervised learning problem can be formulated as

$$\min_{W} \; L\left(X(T)\right) + \int_0^T R(W)\, dt \tag{3}$$

$$\text{s.t.} \; \frac{dX(t)}{dt} = f(X(t), W, t), \quad X(0) = \boldsymbol{x}_0, \quad 0 \le t \le T, \tag{4}$$

where $L(\cdot)$ is the estimation loss function, R is the running cost or regularity, and W controls the propagation of $X(t)$. The control W is similar to the parameters/weights of a neural network. Next, we will discuss two questions:

1. How to solve the optimal W?
2. What is the benefit of using a continuous model? Why not use traditional neural networks?

For the second question, several works demonstrated distinct advantages of using continuous model instead of traditional DNN in some specific applications [Grathwohl et al., 2019, Zhang et al., 2018, Chen et al., 2019], which are summarized in Section 5 and Section 6. For the first question, the ODE constraint optimization problem in Eq. (3) and Eq. (4) can be regarded as a standard optimal control problem, where the unknown W is referred to **control** in optimal control literature. Optimal control has a rich history, so there are many tools that can be utilized for machine learning problems. Li et al. [2017], Weinan et al. [2019] first introduced the idea of regarding a supervised learning problem as an optimal control problem. They consider training the parameters of a neural network as solving an optimal control W from a collection of admissible controls.

In optimal control literature, the control W can be modeled in two forms. One is the feed-back form $W : (X(t), t) \mapsto W(X(t), t) \in \mathcal{W}$, which is called **closed-loop control**. The other form is $W : t \mapsto W(t) \in \mathcal{W}$ (or $W : (X(0), t) \mapsto W(X(0), t) \in \mathcal{W}$), which is called **open-loop control**. Closed-loop control is in general a stronger charaterization of the solution of the control problem. It can actively adapt to the encountered states $X(t)$. In contrast, open-loop control will determine the whole path $W(t)$ for all $t \in [0, T]$ at the initial time $t = 0$, so it is not adapted to $X(t)$. See Figure 1 for a visual explanation.

Closed-loop control is useful for *stochastic* optimal control problems (i.e. the flow f is stochastic), because there exists uncertainty and W can adapted to the most updated information. However, the closed-loop control $W(X(t), t)$ is hard to solve. Typically, one needs to consider the **value function** $V(t, \boldsymbol{x}) := \inf_{W \in \mathcal{W}} \int_t^T R(W)\, dt + L(X(T)),\ \forall (t, \boldsymbol{x}) \in [0, T] \times \mathcal{X}$, which satisfies the **Hamilton-Jacobi-Bellman** (HJB) *equantion*:

$$-V_t + \sup_{W \in \mathcal{W}} H(t, \boldsymbol{x}, -V_{\boldsymbol{x}}, W) = 0, \quad \forall (t, \boldsymbol{x}) \in [0, T) \times \mathcal{X} \tag{5}$$

and $V(T, \boldsymbol{x}) = L(\boldsymbol{x})$, where $H : [0, T] \times \mathcal{X} \times \mathbb{R}^d \times \mathcal{W} \to \mathbb{R}$ is called the **Hamiltonian** and defined as

$$H(t, \boldsymbol{x}, \boldsymbol{p}, W) := \boldsymbol{p} \cdot f(\boldsymbol{x}, W, t) - R(W). \tag{6}$$

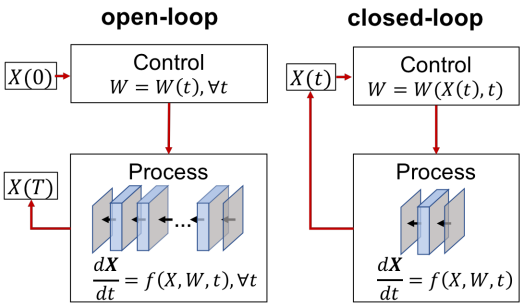

Figure 1: *Left*: An open-loop control makes a sequence of decisions when the initial state is observed. *Right*: A closed-loop control makes decisions based on the most updated information.

Typically, an optimal closed-loop control is obtained by solving the HJB equation, which is in general hard to solve numerically. Although the solution of HJB equation is well-studied, the algorithms are ususally not scalable. Thus it is hard to apply them for high-dimensional machine learning problems. Conversely, many works are trying to use deep learning tools to solve high-dimensional differential equations [Hutzenthaler et al., 2019, Grohs et al., 2018, Han et al., 2018, Freno and Carlberg, 2019, Weinan et al., 2017] and also high-dimensional optimal control problems [Jentzen et al., 2018, Reisinger and Zhang, 2019]. In fact, reinforcement learning is solving stochastic optimal control problems. It generalizes and extends ideas from optimal control to non-traditional control problems.

One possibility to make the closed-loop control easier to solve is to use a biased model. For example, one can parameterize the closed-loop control as $W = \phi(X(t), t; \theta)$ where ϕ is a model (e.g. a neural network) and $\theta \in \Theta$ is its learnable parameters. Then the weights output from ϕ will be adaptive to the current state $X(t)$. This is similar to HyperNetwork: a widely known approach of using one network to generate the weights for another network.

To conclude, closed-loop control is hard to solve in general, and existing algorithms may not be useful for machine learning problems. Meanwhile, for deterministic problems, an open-loop control is good enough. In fact, in a deterministic system, the optimal closed-loop control and the optimal open-loop control will give the same control law and thus the same optimality [Dreyfus, 1964]. Therefore, for the remaining of this paper, we will focus on open-loop control and continue to investigate the two questions posted above.

4 Optimization Methods

In this section, we will discuss existing methods of solving the optimal open-loop control, and how to utilize it for machine learning problems.

4.1 Pontryagin's Maximum Principle

For solving the open-loop control, Pontryagin's Maximum Principle (PMP) [Pontryagin et al., 1962, Boltyanskii et al., 1960] gives a set of first order necessary conditions for the optimal pair (X^*, W^*). In PMP, there is an important concept: the **adjoint process** $P(t) = -\frac{\partial L(X^*(T))}{\partial X^*(t)}$. It can represent the gradient of loss with respect to the state $X^*(t)$. Its evolution is determined by the ODE in Eq. (8). The following is the statement of PMP for deterministic problem.

Theorem 4.1. (Pontryagin's Maximum Principle) *Let $W^*(\cdot) \in \mathcal{W}$ be the optimal control and X^* the optimal controlled state trajectory. Suppose* $\operatorname{ess\,sup}_{t\in[0,T]} \|W^*(t)\|_\infty < \infty$. *Then there exists an adjoint process $P : [0, T] \to \mathbb{R}^d$ such that $\forall t \in [0, T]$,*

$$\dot{X}^*(t) = \nabla_{\boldsymbol{p}} H(t, X^*(t), P(t), W^*(t)), \quad X^*(0) = \boldsymbol{x}_0, \tag{7}$$

$$\dot{P}(t) = -\nabla_{\boldsymbol{x}} H(t, X^*(t), P(t), W^*(t)), \quad P(T) = -\nabla_{\boldsymbol{x}} L(X^*(T)), \tag{8}$$

$$W^*(t) = \arg\max_{W\in\mathcal{W}} H(t, X^*(t), P(t), W), \quad \forall t \in [0, T], \tag{9}$$

where H is the Hamiltonian defined in Eq. (6).

Eq. (9) can not be implemented without discretization. There is also a discrete version of PMP, where the state X is updated iteratively by $X_{k+1} = g(X_k, W_k, k)$. The corresponding discrete PMP states the following conditions:

$$X^*_{t+1} = g(X^*_t, W^*_t, t), \quad X^*_0 = \boldsymbol{x}^i, \tag{10}$$

$$P_t = -\nabla_{\boldsymbol{x}} H(t, X^*_t, P_{t+1}, W^*_t), \quad P_T = -\nabla_{\boldsymbol{x}} L(X^*_T), \tag{11}$$

$$W^*_t = \arg\max_{W \in \mathcal{W}} H(t, X^*_t, P_{t+1}, W), \quad \forall t = 0, \dots, T-1, \tag{12}$$

where H is the discrete Hamiltonian

$$H(t, \boldsymbol{x}, \boldsymbol{p}, W) := \boldsymbol{p} \cdot g(\boldsymbol{x}, W, t) - \delta R(W). \tag{13}$$

Hence, by solving PMP, one can obtain the optimal open-loop control. There are many ways of solving PMP. However, to solve large scale problems, Method of successive approximation (MSA) is a more favorable algorithm.

Algorithm 1 discrete MSA

1: Initialize W^0.
2: **for** $k = 0$ to M **do**
3: Given $X_0 = \boldsymbol{x}_0$, compute $X_{t+1} = g(X_t, W^{(k)}_t, t)$, for $t = 0$ to $T-1$;
4: Given $P_T = -\nabla_{\boldsymbol{x}} L(X_T)$, compute $P_t = -\nabla_{\boldsymbol{x}} H(t, X_t, P_{t+1}, W^{(k)}_t)$ for $t = T-1$ to 0;
5: Set $W^{(k+1)}_t = \arg\max_{W' \in \mathcal{W}} H(t, X_t, P_{t+1}, W')$ for each $t = 0, \dots, T-1$.
6: **end for**

There is an interesting fact that if one replaces line 5 in algorithm 1 by one gradient step $W^{(k+1)}_t \leftarrow W^{(k+1)}_t + \nabla_W H(t, X_t, P_{t+1}, W^{(k)}_t)$, then it is equivalent to gradient descent [Li et al., 2017].

To conclude, PMP provides another optimization method other than stochastic gradient descent. However, what is the benefit of using PMP instead of SGD? (1) The first obvious advantage is that PMP does not require gradient with respect to W. Therefore, it can be applied to train discrete variables. There are also experiments showing that very sparse weights are obtained while using PMP to train discrete neural networks [Li and Hao, 2018]. (2) Thanks to the rich history of optimal control, PMP has the advantages that rigorous error estimates and convergence results can be established, which allows it easier to modify and improve the algorithm. For example, Li et al. [2017] modified the Hamiltonian and showed a favorable initial convergence rate per iteration.

4.2 Compute Gradient for Continuous Model

For a high-dimensional *continuous* model, it is expensive to use PMP to solve W because of the final step in Eq. (9). However, one can simplify the model by setting a time-invariant control $W(t) \equiv W$ and make use of the adjoint process to compute the gradient. Chen et al. [2018] proposed this idea and provided an algorithm which only requires $O(1)$ memory. A summary is given in Figure 2. Chen et al. [2018] also showed by experiments that compared to ResNet,

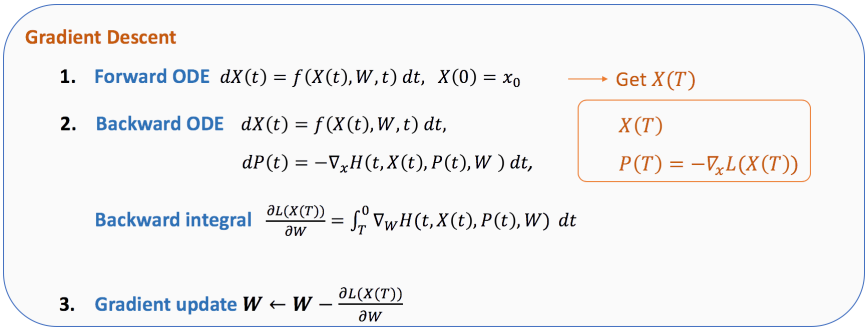

Figure 2: Use adjoint process to compute gradient with low memory cost.

this continuous model can achieve comparable performance with a much smaller number of parameters $W(t) \equiv W$. In the next two sections, we will discuss two more applications to show the advantages of continuous model in some important machine learning problems.

5 Application: Generative Model

Suppose the data points $\{\boldsymbol{x}^i\}_{i=1}^N$ are collected from a particular distribution p_x. The goal of generative modeling is to estimate the distribution p_x given a collection of data points. A common way is to start from a simple distribution $Z \sim p_z$ (e.g. standard Gaussian $\mathcal{N}(0,1)$), and then transform the variable Z using a function mapping $X = \phi_\theta(Z)$, where ϕ_θ can be a neural network with parameters θ [Dinh et al., 2014, Kingma and Dhariwal, 2018, Dinh et al., 2017]. θ will be learned using the data points. However, to compute the transformed density, one needs to compute

$$\log p_x(\boldsymbol{x}) = \log p_z(\boldsymbol{z}) - \log \det \left| \frac{\partial \phi_\theta(\boldsymbol{z})}{\partial \boldsymbol{z}} \right|. \tag{14}$$

The determinant of Jacobian requires $\mathcal{O}(d^3)$ computation, so existing methods have to restrict the architecture of ϕ_θ to make this term tractable.

Zhang et al. [2018] and Grathwohl et al. [2019] used continuous ODE to replace the function mapping ϕ_θ. That is, sampling from an initial distribution $X(0) \sim p_z$, through a continuous propagation determined by the ODE $\frac{dX}{dt} = f(X, W, t)$, the solution $X(T)$ is the sample from the transformed distribution. The advantage of this modeling is that the change of density also follows an ODE

$$\frac{d \log p_x(\boldsymbol{x}, t)}{dt} = -\nabla_x \cdot f, \tag{15}$$

where the computation of $-\nabla_x \cdot f$ is $\mathcal{O}(d^2)$. Through a stochastic estimator, Grathwohl et al. [2019] reduced this computation cost to $\mathcal{O}(d)$. By applying theorems from optimal transport, Zhang et al. [2018] proposed to model the flow f as the gradient of a neural network $f := \nabla \phi_\theta$.

6 Research Work: Particle Flow Bayes' Rule

6.1 Introduction and Related Work

Bayesian inference is a core machine learning problem. In many data analysis tasks, it is important to estimate unknown quantities $\boldsymbol{x} \in \mathbb{R}^d$ from observations $\mathcal{O}_m := \{o_1, \cdots, o_m\}$. Given prior knowledge $\pi(\boldsymbol{x})$ and likelihood functions $p(o_t|\boldsymbol{x})$, the essence of Bayesian inference is to compute the posterior $p(\boldsymbol{x}|\mathcal{O}_m) \propto \pi(\boldsymbol{x}) \prod_{t=1}^m p(o_t|\boldsymbol{x})$ by Bayes' rule. For many nontrivial models, the prior might not be conjugate to the likelihood, making the posterior not in a closed form. Therefore, computing the posterior often results in intractable integration and poses significant challenges. Typically, one resorts to approximate inference methods such as sampling (e.g., MCMC) [Andrieu et al., 2003] or variational inference [Wainwright et al., 2008].

In many real problems, observations arrive sequentially online, and Bayesian inference needs be performed recursively,

$$\overbrace{p(\boldsymbol{x}|\mathcal{O}_{m+1})}^{\text{updated posterior}} \propto \overbrace{p(\boldsymbol{x}|\mathcal{O}_m)}^{\text{current posterior}} \overbrace{p(o_{m+1}|\boldsymbol{x})}^{\text{likelihood}}. \tag{16}$$

That is the estimation of $p(\boldsymbol{x}|\mathcal{O}_{m+1})$ should be computed based on the estimation of $p(\boldsymbol{x}|\mathcal{O}_m)$ obtained from the last iteration and the presence of the new observation o_{m+1}. It therefore requires algorithms which allow for efficient online inference. In this case, both standard MCMC and variational inference become inefficient, since the former requires a complete scan of the dataset in each iteration, and the latter requires solving an optimization for every new observation. Thus, sequential Monte Carlo (SMC) [Doucet et al., 2001, Balakrishnan et al., 2006] or stochastic approximations, such as stochastic gradient Langevin dynamics [Welling and Teh, 2011] and stochastic variational inference [Hoffman et al., 2013], are developed to improve the efficiency. However, SMC suffers from the path degeneracy problems in high dimensions [Daum and Huang, 2003], and rejuvenation steps are designed but may violate the online sequential update requirement [Canini et al., 2009, Chopin et al., 2013]. Stochastic approximation methods are prescribed algorithms that cannot exploit the structure of the problem for further improvement.

Chen et al. [2019] designed a continuous particle flow operator $\mathcal{F}$ to realize the Bayes update, called **meta particle flow** (MPF). In the MPF framework (Fig. 3), a prior distribution $\pi(\boldsymbol{x})$, or, the current posterior $\pi_m(\boldsymbol{x}) := p(\boldsymbol{x}|\mathcal{O}_m)$ will be approximated by a set of N equally weighted particles $\mathcal{X}_m = \{\boldsymbol{x}_m^1, \ldots, \boldsymbol{x}_m^N\}$; and then, given an observation o_{m+1}, the flow operator $\mathcal{F}(\mathcal{X}_m, \boldsymbol{x}_m^n, o_{m+1})$ will transport each particle $\boldsymbol{x}_m^n$ to a new particle $\boldsymbol{x}_{m+1}^n$ corresponding to the updated posterior $p(\boldsymbol{x}|\mathcal{O}_{m+1})$.

In a high-level, the MPF operator $\mathcal{F}$ as a continuous deterministic flow, which propagates the locations of particles and the values of their probability density simultaneously through a dynamical system described by ordinary differential equations (ODEs).

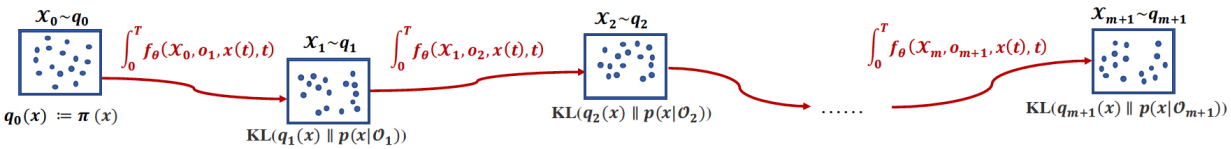

Figure 3: Sequential Bayesian inference as a deterministic flow of particles.

6.2 Bayesian Inference as Particle Flow

Starting with N particles $\mathcal{X}_0 = \{\boldsymbol{x}_0^1, \ldots, \boldsymbol{x}_0^N\}$ sampled i.i.d. from a prior distribution $\pi(\boldsymbol{x})$, given an observation o_1, the operator $\mathcal{F}$ will transport the particles to $\mathcal{X}_1 = \{\boldsymbol{x}_1^1, \ldots, \boldsymbol{x}_1^N\}$ to estimate the posterior $p(\boldsymbol{x}|\mathcal{O}_1) \propto \pi(\boldsymbol{x})p(o_1|\boldsymbol{x})$. The transformation is modeled as the solution of an initial value problem of an ordinary differential equation (ODE). That is, $\forall n$,

$$\begin{cases} \frac{d\boldsymbol{x}}{dt} = f(\mathcal{X}_0, o_1, \boldsymbol{x}(t), t), \ \forall t \in (0, T] \\ \boldsymbol{x}(0) = \boldsymbol{x}_0^n \end{cases} \implies \text{gives } \boldsymbol{x}_1^n = \boldsymbol{x}(T),$$

where the flow velocity f takes the observation o_1 as the input, and determines both the direction and the speed of the change of $\boldsymbol{x}(t)$. In the above ODE model, each particle $\boldsymbol{x}_0^n$ sampled from the prior gives an initial value $\boldsymbol{x}(0)$, and then the flow velocity f will evolve the particle continuously and deterministically. At the terminal time T, we will take the result of the evolution $\boldsymbol{x}(T)$ as the transformed particle $\boldsymbol{x}_1^n$ for estimating the posterior.

Applying this ODE-based transformation sequentially as new observations $o_2, o_3, \ldots$ arrive, a recursive particle flow Bayes operator is defined as

$$\begin{aligned} \boldsymbol{x}_{m+1}^n &= \mathcal{F}(\mathcal{X}_m, o_{m+1}, \boldsymbol{x}_m^n) \\ &:= \boldsymbol{x}_m^n + \textstyle\int_0^T f(\mathcal{X}_m, o_{m+1}, \boldsymbol{x}(t), t)\, dt. \end{aligned} \tag{17}$$

6.2.1 Property of Continuous Deterministic Flow

The continuous transformation of $\boldsymbol{x}(t)$ described by the differential equation $d\boldsymbol{x}/dt = f$ defines a *deterministic* flow for each particle. Let $q(\boldsymbol{x}, t)$ be the probability density of the continuous random variable $\boldsymbol{x}(t)$. The change of this density is also determined by f. More specifically, the evolution of the density follows the widely known **continuity equation** [Batchelor, 2000]:

$$\frac{\partial q(\boldsymbol{x}, t)}{\partial t} = -\nabla_x \cdot (qf), \tag{18}$$

where $\nabla_x \cdot$ is the divergence operator. Continuity equation is the mathematical expression for the law of *local conservation of mass* - mass can neither be created nor destroyed, nor can it "teleport" from one place to another.

Given continuity equation, one can describe the change of log-density by another differential equation, for which we state it as Theorem 6.1.

Theorem 6.1. *If $d\boldsymbol{x}/dt = f$, then the change in log-density follows the differential equation*

$$\frac{d \log(q(\boldsymbol{x}, t))}{dt} = -\nabla_x \cdot f. \tag{19}$$

Note: d/dt is material derivative (total derivative) and $\partial/\partial t$ is partial derivative. dq/dt defines the rate of change of q in a given particle as it moves along its trajectory $\boldsymbol{x} = \boldsymbol{x}(t)$ in the flow, while $\partial q/\partial t$ means the rate of change of q at a particular point $\boldsymbol{x}$ that is fixed in the space.

With theorem 6.1, we can compute the log-density of the particles associated with the Bayes operator $\mathcal{F}$ by integrating across $(0, T]$ for each n:

$$\log q_{m+1}(x_{m+1}^n) = \log q_m(x_m^n) - \textstyle\int_0^T \nabla_x \cdot f\, dt. \tag{20}$$

6.3 Existence of Shared Flow Velocity

Does a shared flow velocity f exist for different Bayesian inference tasks involving different priors and observations? If it does, what is the form of this function? These questions are non-trivial even for simple Gaussian case.

For instance, let the prior $\pi(x) = \mathcal{N}(0, \sigma_x)$ and the likelihood $p(o|x) = \mathcal{N}(x, \sigma)$ both be one dimensional Gaussian distributions. Given an observation $o = 0$, the posterior distribution of x is also a Gaussian distributed as $\mathcal{N}(0, (\sigma \cdot$

$\sigma_x)/(\sigma+\sigma_x))$. This means that the dynamical system $d\boldsymbol{x}/dt = f$ needs to push a zero mean Gaussian distribution with covariance σ_x to another zero mean Gaussian distribution with covariance $(\sigma \cdot \sigma_x)/(\sigma+\sigma_x)$ for any σ_x. It is not clear whether such a shared flow velocity function f exists and what is the form for it.

To resolve the existence issue, we consider the Langevin dynamics, which is a *stochastic* process

$$d\boldsymbol{x}(t) = \nabla_x \log p(\boldsymbol{x}|\mathcal{O}_m)p(o_{m+1}|\boldsymbol{x})\,dt + \sqrt{2}d\boldsymbol{w}(t), \tag{21}$$

where $d\boldsymbol{w}(t)$ is a standard Brownian motion. This stochastic flow is very different in nature comparing to the deterministic flow in Section 6.2.1, where a fixed location $\boldsymbol{x}(0)$ will always end up as the same location $\boldsymbol{x}(t)$. Nonetheless, we established their connection and proved the following theorem.

Theorem 6.2. *There exists a fixed and deterministic flow velocity f of the form*

$$\nabla_x \log p(\boldsymbol{x}|\mathcal{O}_m)p(o_{m+1}|\boldsymbol{x}) - w^*(p(\boldsymbol{x}|\mathcal{O}_m), p(o_{m+1}|\boldsymbol{x}), t), \tag{22}$$

such that its density $q(x,t)$ the same evolution as the density of Langevin dynamics.

6.4 Parametrization

In Section 6.3, we introduce a shared flow velocity, which is in form of $f(p(\boldsymbol{x}|\mathcal{O}_m), p(o_{m+1}|\boldsymbol{x}), \boldsymbol{x}(t), t)$ as indicated by Eq. (22). We design the parameterization of f based on this expression.

(i) $p(\boldsymbol{x}|\mathcal{O}_m) \Rightarrow \mathcal{X}_m$: In our particle flow framework, since we do not have full access to the density $p(\boldsymbol{x}|\mathcal{O}_m)$ but have samples $\mathcal{X}_m = \{\boldsymbol{x}_m^1, \ldots, \boldsymbol{x}_m^N\}$ from it, we can use these samples as surrogates for $p(\boldsymbol{x}|\mathcal{O}_m)$. A related example is feature space embedding of distributions [Smola et al., 2007], $\mu_{\mathcal{X}}(p) := \int_{\mathcal{X}} \phi(\boldsymbol{x})p(\boldsymbol{x})\,d\boldsymbol{x} \approx \frac{1}{N}\sum_{n=1}^N \phi(\boldsymbol{x}^n),\ \boldsymbol{x}^n \sim p$.

Ideally, if $\mu_{\mathcal{X}}$ is an injective mapping from the space of probability measures over $\mathcal{X}$ to the feature space, the resulting embedding can be treated as a sufficient statistic of the density and any information we need from $p(\boldsymbol{x}|\mathcal{O}_m)$ can be preserved. Hence, we represent $p(\boldsymbol{x}|\mathcal{O}_m)$ by $\frac{1}{N}\sum_{n=1}^N \phi(\boldsymbol{x}_m^n)$, where $\phi(\cdot)$ is a nonlinear feature mapping to be learned. Since we use a neural version of $\phi(\cdot)$, this representation can also be regarded as a DeepSet [Zaheer et al., 2017].

(ii) $p(o_{m+1}|\boldsymbol{x}) \Rightarrow (o_{m+1}, \boldsymbol{x}(t))$: In both Langevin dynamics and Eq. (22), the only term containing the likelihood is $\nabla_x \log p(o_{m+1}|\boldsymbol{x})$. Consequently, we can use this term as an input to f. In the case when the likelihood function is fixed, we can also simply use the observation o_{m+1}, which results in similar performance in our experiments.

Overall we will parameterize the flow velocity as

$$f = \boldsymbol{h}\left(\frac{1}{N}\sum_{n=1}^N \phi(\boldsymbol{x}_m^n), o_{m+1}, \boldsymbol{x}(t), t\right), \tag{23}$$

where $\boldsymbol{h}$ is a neural network and $\theta \in \Theta$ are parameters of $\boldsymbol{h}$ and ϕ. From now on, we will write $f = f_\theta(\mathcal{X}_m, o_{m+1}, \boldsymbol{x}(t), t)$, where $\theta \in \Theta$ is independent of t. In the next section, we will propose a meta learning framework for learning these parameters.

6.5 Meta Learning

Since we want to learn a generalizable Bayesian inference operator, we need to create multiple inference tasks and design the corresponding training and testing algorithm. We will discuss these details below.

Multi-task Framework. We are interested in a training set $\mathcal{D}_{train}$ containing multiple inference tasks. Each task is a tuple

$$\mathcal{T} := (\pi(\boldsymbol{x}), p(\cdot|\boldsymbol{x}), \mathcal{O}_M := \{o_1, \ldots, o_M\}) \in \mathcal{D}_{train}$$

which consists of a prior distribution, a likelihood function and a sequence of M observations. As we explained in previous sections, we want to learn a Bayesian operator $\mathcal{F}$ that can be applied recursively to hit the targets $p(\boldsymbol{x}|\mathcal{O}_1), p(\boldsymbol{x}|\mathcal{O}_2), \cdots$ sequentially. Therefore, each task can also be interpreted as a sequence of M sub-tasks:

$$\tau := (p(\boldsymbol{x}|\mathcal{O}_m), p(\cdot|\boldsymbol{x}), o_{m+1}) \in (\pi(\boldsymbol{x}), p(\cdot|\boldsymbol{x}), \mathcal{O}_M).$$

Therefore, each task is a sequential Bayesian inference and each sub-task corresponds to one step Bayesian update.

Cumulative Loss Function. For each sub-task we define a loss $\mathrm{KL}(q_m(\boldsymbol{x})||p(\boldsymbol{x}|\mathcal{O}_{m+1}))$, where $q_m(\boldsymbol{x})$ is the distribution transported by $\mathcal{F}$ at m-th stage and $p(\boldsymbol{x}|\mathcal{O}_{m+1})$ is the target posterior (see Fig. 3 for illustration). Meanwhile, the loss for the corresponding sequential task will be $\sum_{m=1}^M \mathrm{KL}(q_m(\boldsymbol{x})||p(\boldsymbol{x}|\mathcal{O}_m))$, which sums up the sub-loss of

all intermediate stages. Since its optimality is independent of normalizing constants of $p(\boldsymbol{x}|\mathcal{O}_m)$, it is equivalent to minimize the negative evidence lower bound (ELBO)

$$\mathcal{L}(\mathcal{T}) = \sum_{m=1}^{M} \sum_{n=1}^{N} \left(\log q_m(\boldsymbol{x}_m^n) - \log p(\boldsymbol{x}_m^n, \mathcal{O}_m)\right). \tag{24}$$

The above expression is an empirical estimation using particles $\boldsymbol{x}_m^n$. In each iteration during training phase, we will randomly sample a task from $\mathcal{D}_{train}$ and compute the gradient of the above cumulative loss function to update the parameters of our MPF operator $\mathcal{F}$.

6.6 Experiments

Experiments on multivariate Gaussian model, hidden Markov model and Bayesian logistic regression are conducted to demonstrate the generalization ability of MPF as a Bayesian operator and also the posterior estimation performance of MPF as an inference method. Results can be found in Chen et al. [2019].

7 Conclusion

In this paper, we draw a comprehensive connection between neural architecture and ODE discretization, and between supervised learning and optimal control, and between PMP and SGD. Such connection enables us to design new models or algorithms for machine learning problems. More importantly, it opens new avenues to attack problems associated with machine learning.

References

Christophe Andrieu, Nando De Freitas, Arnaud Doucet, and Michael I Jordan. An introduction to mcmc for machine learning. *Machine learning*, 50(1-2):5–43, 2003.

Suhrid Balakrishnan, David Madigan, et al. A one-pass sequential monte carlo method for bayesian analysis of massive datasets. *Bayesian Analysis*, 1(2):345–361, 2006.

G.K. Batchelor. *Kinematics of the Flow Field*, pages 71–130. Cambridge Mathematical Library. Cambridge University Press, 2000.

Vladimir Grigor'evich Boltyanskii, Revaz Valer'yanovich Gamkrelidze, and Lev Semenovich Pontryagin. The theory of optimal processes. i. the maximum principle. Technical report, TRW SPACE TECHNOLOGY LABS LOS ANGELES CALIF, 1960.

Kevin Canini, Lei Shi, and Thomas Griffiths. Online inference of topics with latent dirichlet allocation. In *Artificial Intelligence and Statistics*, pages 65–72, 2009.

Bo Chang, Minmin Chen, Eldad Haber, and Ed H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks. In *International Conference on Learning Representations*, 2019.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

Xinshi Chen, Hanjun Dai, and Le Song. Meta particle flow for sequential bayesian inference. In *International Conference on Machine Learning*, 2019.

Nicolas Chopin, Pierre E Jacob, and Omiros Papaspiliopoulos. Smc2: an efficient algorithm for sequential analysis of state space models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 75(3):397–426, 2013.

Fred Daum and Jim Huang. Curse of dimensionality and particle filters. In *2003 IEEE Aerospace Conference Proceedings (Cat. No. 03TH8652)*, volume 4, pages 4_1979–4_1993. IEEE, 2003.

Laurent Dinh, David Krueger, and Yoshua Bengio. Nice: Non-linear independent components estimation. *International Conference on Learning Representations Workshop*, 2014.

Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. In *International Conference on Learning Representations*, 2017.

Arnaud Doucet, Nando De Freitas, and Neil Gordon. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pages 3–14. Springer, 2001.

Stuart E Dreyfus. Some types of optimal control of stochastic systems. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 2(1):120–134, 1964.

Brian A Freno and Kevin T Carlberg. Machine-learning error models for approximate solutions to parameterized systems of nonlinear equations. *Computer Methods in Applied Mechanics and Engineering*, 348:250–296, 2019.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pages 2214–2224, 2017.

Will Grathwohl, Ricky T. Q. Chen, Jesse Bettencourt, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models. In *International Conference on Learning Representations*, 2019.

Philipp Grohs, Fabian Hornung, Arnulf Jentzen, and Philippe Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

Eldad Haber and Lars Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34(1):014004, 2017.

Jiequn Han, Arnulf Jentzen, and E Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *arXiv preprint arXiv:1901.10854*, 2019.

Arnulf Jentzen, Diyora Salimova, and Timo Welti. A proof that deep artificial neural networks overcome the curse of dimensionality in the numerical approximation of kolmogorov partial differential equations with constant diffusion and nonlinear drift coefficients. *arXiv preprint arXiv:1809.07321*, 2018.

Durk P Kingma and Prafulla Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.

Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. Fractalnet: Ultra-deep neural networks without residuals. In *International Conference on Learning Representations*, 2017.

Qianxiao Li and Shuji Hao. An optimal control approach to deep learning and applications to discrete-weight neural networks. In *International Conference on Machine Learning*, pages 2991–3000, 2018.

Qianxiao Li, Long Chen, Cheng Tai, and E Weinan. Maximum principle based algorithms for deep learning. *The Journal of Machine Learning Research*, 18(1):5998–6026, 2017.

Qianli Liao and Tomaso Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

Yiping Lu, Aoxiao Zhong, Quanzheng Li, and Bin Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *International Conference on Machine Learning*, pages 3282–3291, 2018.

Lev Semenovich Pontryagin, EF Mishchenko, VG Boltyanskii, and RV Gamkrelidze. The mathematical theory of optimal processes. 1962.

Christoph Reisinger and Yufei Zhang. Rectified deep neural networks overcome the curse of dimensionality for nonsmooth value functions in zero-sum games of nonlinear stiff systems. *arXiv preprint arXiv:1903.06652*, 2019.

Lars Ruthotto and Eldad Haber. Deep neural networks motivated by partial differential equations. *arXiv preprint arXiv:1804.04272*, 2018.

Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *International Conference on Algorithmic Learning Theory*, pages 13–31. Springer, 2007.

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

E Weinan, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017.

E Weinan, Jiequn Han, and Qianxiao Li. A mean-field optimal control formulation of deep learning. *Research in the Mathematical Sciences*, 6(1):10, 2019.

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

Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Ruslan R Salakhutdinov, and Alexander J Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.

Linfeng Zhang, E Weinan, and Lei Wang. Monge-amp\ere flow for generative modeling. *arXiv preprint arXiv:1809.10188*, 2018.

Xingcheng Zhang, Zhizhong Li, Chen Change Loy, and Dahua Lin. Polynet: A pursuit of structural diversity in very deep networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 718–726, 2017.