Differential Equations for Continuous-Time Deep Learning

Lars Ruthotto*

January 9, 2024

Abstract

This short, self-contained article seeks to introduce and survey continuous-time deep learning approaches that are based on neural ordinary differential equations (neural ODEs). It primarily targets readers familiar with ordinary and partial differential equations and their analysis who are curious to see their role in machine learning. Using three examples from machine learning and applied mathematics, we will see how neural ODEs can provide new insights into deep learning and a foundation for more efficient algorithms.

Deep learning has been behind incredible breakthroughs, such as voice recognition, image classification, and text generation. While these successes are undeniable, our mathematical understanding of deep learning is still developing. More rigorous insight is needed to overcome fundamental challenges, including the interpretability, robustness, and bias of deep learning, and lower its environmental and computational costs.

Let us define deep learning informally as machine learning methods that use feed-forward neural networks with many (i.e., more than a handful) layers. While most traditional approaches use a finite number of layers, we will focus on more recent approaches that conceptually use infinitely many layers. We will explain those approaches by defining differential equations whose dynamics are modeled by trainable neural network components and whose time roughly corresponds to the depth of the network.

Using three examples from machine learning and applied mathematics, we will see how continuous-depth neural network architectures, defined by ordinary differential equations (ODEs), can provide new insights into deep learning and a foundation for more efficient algorithms. Even though many deep learning approaches used in practice today do not rely on differential equations, I find many opportunities for mathematical research in this area. As we shall see, phrasing the problem continuously in time enables one to borrow numerical techniques and analysis to gain more insight into deep learning, design new approaches, and crack open the black box of deep learning. We shall also see how neural ODEs can approximate solutions to high-dimensional, nonlinear optimal control problems.

This article targets readers familiar with ordinary and partial differential equations and their analysis and who are curious to see their role in machine learning. Therefore, rather than drawing a complete picture of state-of-the-art machine learning, which is shooting at moving targets, we seek to provide foundational insights and motivate further study. To this end, we will inevitably take shortcuts and sacrifice being up-to-date for clarity. Even though designing efficient numerical algorithms is critical to translating the theoretical advantages of the continuous-time viewpoint into practical learning approaches, we will keep this topic for another day and provide some references for the interested reader.

While giving a complete picture of the research activity at this interface between differential equations and deep learning is beyond the scope of this paper, we seek to capture the key ideas and provide a head start to those eager to learn more. This paper aims to illustrate the foundations and some of the benefits of continuous-time deep learning using a few hand-picked examples related to the author's activity in the area. This article is an unedited and extended version of an article under review at AMS Notices and contains a few additional references.

*Lars Ruthotto is a Winship Distinguished Research Associate Professor at Emory University. His email address is lruthotto@emory.edu.

Deep Neural Networks in Continuous Time

Before we begin, let us introduce the main mathematical notation of deep learning for this paper and motivate continuous-time architectures.

Throughout this paper, $F_\theta : \mathbb{R}^n \to \mathbb{R}^m$ denotes a neural network with the subscript $\theta \in \mathbb{R}^p$ representing its parameters, often called weights. We will see that there are different ways of defining F_θ. For example, a common choice is the L-layer feed-forward network that maps the input $x_0 = x \in \mathbb{R}^n$ to the output $F_\theta(x) = x_L$ via the layers

$$x_{l+1} = \sigma_\ell \left(W_l x_l + b_l \right), \ \forall l = 0, \dots, L-1. \quad (1)$$

Here, the activation functions σ_l are applied elementwise, and the parameters of the lth layer, θ_l, consist of the weight matrix W_l and the weight vector b_l. To define F_θ, one must choose the number of layers, activation functions, and the sizes of the weight matrices and vectors, collectively called hyperparameters. Except for the number of columns in W_0, which must be n, and the number of rows in W_{L-1}, b_{L-1}, which must be m, the remaining sizes can be adjusted arbitrarily.

With an effective way to choose hyperparameters and identify the model weights, deep networks are perhaps the most efficient high-dimensional function approximators today. Their versatility has enabled their use across various tasks. For example, neural networks have been used in supervised learning to fit given inputs to corresponding outputs, in reinforcement learning to predict optimal actions, and in generative modeling to match simple latent distributions to a complex distribution available only through samples. The success of neural networks across these tasks is rooted in their approximation properties. For example, it can be shown that networks with one hidden layer are universal approximators. Since we will use these networks as an important building block later, let us define them as

$$f_\theta(x) = W_1 \tanh(W_0 x + b_0) + b_1, \quad (2)$$

whose parameter, θ, consists of the weight matrices $W_0 \in \mathbb{R}^{k \times n}, W_1 \in \mathbb{R}^{m \times k}$, weight vectors $b_0 \in \mathbb{R}^k, b_1 \in \mathbb{R}^m$, and whose activation is the hyperbolic tangent function. In other words, for every $\epsilon > 0$, there is a width, k, such that they approximate continuous functions to a given accuracy $\epsilon > 0$. Since the width needed to achieve the desired accuracy can be impractically large, most applications today prefer narrower but deeper architectures. This is supported by theoretical results such as in [KL19], which show that as L grows, the model becomes more expressive and can be a universal approximator even with a finite width.

Increasing the depth of neural networks such as the one in (1) to realize approximation results is easier said than done. In practice, it often becomes more and more challenging to identify model weights that accurately approximate the function of interest. For example, it is difficult to approximate the identity function with a network like (1).

Residual neural networks (ResNets) provide an alternative way to define very deep networks and considerably improved the state-of-the-art in computer vision applications recently [HZRS1603]. Their key innovation is often called a skip connection that turns, for example, (2) into the residual layer

$$r_\theta(x) = x + f_\theta(x). \quad (3)$$

Such a ResNet layer can learn the identity map by the trivial choice $f_\theta \equiv 0$. Consequently, increasing the network depth by adding residual layers often improves the approximation result since the weights of the new layers can be chosen to approximate the identity.

One way to motivate deep neural networks that are continuous in time is to view $r_\theta(x)$ in (3) as a forward Euler approximation of $z(1)$ where z solves the initial value problem

$$\frac{d}{dt} z = f_{\theta(t)}(z), \quad t \in (0, 1], \quad z(0) = x. \quad (4)$$

Here, t is an artificial time, and with the notation $\theta(t)$, we seek to suggest that the weights can be modeled as functions of time; see [E1703, HR17]. This viewpoint was popularized in the machine learning community by the work [CRBD18], which also coined the term Neural ODEs and demonstrated several new use cases. It is important to remember that ResNets and Neural ODEs are different: one is discrete, and the other is continuous. Their relation has been discussed and analyzed in more detail in [MPP$^+$20, SAP22]. We also note that when the features, x, represent functions (e.g., voice, image, or video data) the above model can mimic partial differential equations [RH18].

Supervised Learning in Continuous Time

In this section, we show how using continuous-time models for supervised learning leads to learning problems that can be analyzed and solved using tools from optimal control.

In supervised learning, the goal is to learn F_θ that approximates the relation between labeled input-output pairs $(x, y) \sim D$ assumed to be independent samples from some data distribution D. Once the hyperparameters of the network are chosen, learning the weights θ is typically phrased as a minimization problem, such as

$$\min_\theta \mathbb{E}_{(x,y)\sim D}[\ell(F_\theta(x), y)], \tag{5}$$

with some loss function ℓ whose definition depends on the task; for example, for data fitting, one can consider the regression loss $\ell(v, y) = \frac{1}{2}\|v - y\|^2$. The optimization problem is challenging and interesting in its own right, and we refer to [BCN16] for an excellent monograph.

A simple way to define a continuous-time model is to define F_θ as an affine transformation of the terminal state of a neural ODE, that is,

$$F_\theta(x) = Wz(1) + b \tag{6}$$

$$\frac{d}{dt}z = f_{\theta(t)}(z), \quad t \in (0, 1], \quad z(0) = x. \tag{7}$$

This model can be interpreted as the affine model (given by the parameters W and b) applied to the features evolved by the neural ODE, whose dynamics are governed by the weight function θ. Note that the affine function can be omitted when $m = n$.

Since neural ODEs as in 6 yield an invertible transformation of the data space, it is unsurprising that many functions cannot be approximated by the model in (6). In Figure 1, we demonstrate this in one example that can also illustrate the role of the ODE in the supervised learning problem. Shown here is a classification example obtained by minimizing a logistic regression loss function. Each data point x is associated with a label y, either blue or red. The goal is to learn a function F_θ corresponding to the training data. The center column of the figure shows the propagated features, which are the $z(1)$ associated with each example, as well as the hyperplane parameterized by W, b that seeks to divide the features. The rightmost column visualizes the predictions of the classifier. While the predictions are nearly perfect in both rows, upon close inspection, the limitations of the ODE shine through in the top row, and it can be seen that the network was unable to transform the blue and red points to become linearly separable, which requires a non-invertible transformation. The need for that is alleviated by simply padding the input features with one zero and embedding them into three dimensions. This phenomenon and the importance of augmentation are elaborated in [DDT19].

When F_θ is defined as in (6) the minimization problem (5) becomes an optimal control problem

$$\begin{aligned} \min_{\theta,W,b,z} \ & \mathbb{E}_{(x,y)\sim D}[\ell(Wz(1) + b, y)], \\ \text{s.t.} \ & \frac{d}{dt}z = f_{\theta(t)}(z), \quad t \in (0, 1] \\ & z(0) = x. \end{aligned} \tag{8}$$

The relation between learning θ and solving an optimal control problem has been used to gain insights and obtain more efficient algorithms. Universal approximation results of continuous-time models are derived in [LLS22] by analyzing their flow maps. On the computational side, it is worth mentioning the approach in [LCTW18], which uses control theory to derive new learning algorithms, and [BCE+19], which studies the continuous and discrete versions of the learning problem and proposes schemes that can learn time discretizations and embed other constraints.

With an architecture continuous in time, it is also possible to provide a PDE perspective of the supervised learning problem (8). Following the presentation in [WYSO20], let us take a macroscopic viewpoint and consider the space of examples rather than individual data points. To this end, we model the neural network predictions as a function $u : \mathbb{R}^n \times [0, 1] \to \mathbb{R}^m$ whose evolution is governed by the transport PDE with velocity f_θ. In doing so, we formulate supervised learning as a PDE-constrained optimization problem

$$\begin{aligned} \min_{u,\theta,W,b} \ & \mathbb{E}_{(x,y)\sim D}[\ell(u(x, 1), y)], \\ \text{s.t.} \ & \partial_t u(z, t) + f_{\theta(t)}(z)^\top \nabla u(z, t) = 0 \\ & u(z, 0) = Wx + b. \end{aligned} \tag{9}$$

To verify that the problems are equivalent, note that (6) defines the characteristic curves of the transport equation and therefore

$$\begin{aligned} u(x, 1) &= u(z(0), 1) \\ &= Wz(1) + b \\ &= F_\theta(x) \approx y. \end{aligned}$$

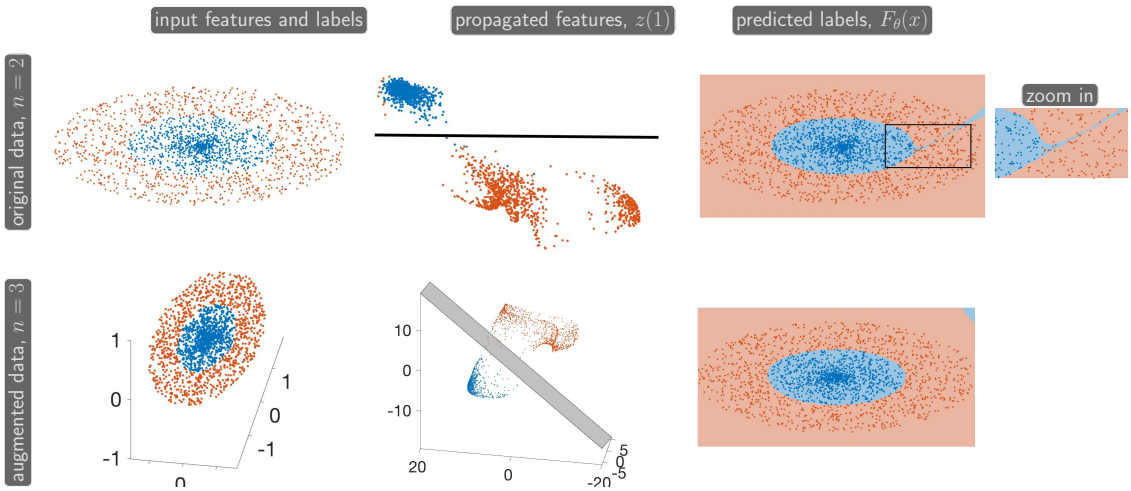

Figure 1: Illustration of a continuous-time model for binary classification. Left column: input features and labels. Center column: propagated features given by the final state of ODE and hyperplane given by W, b. Right column: Labels predicted by the neural network. The rows show two instances of the problem using the original two-dimensional and augmented features (padded with one zero), respectively. While both models agree closely around the samples, we highlight some errors of the original model that arise from the restriction to invertible maps in $n = 2$. This example demonstrates that augmenting overcomes the need for a non-invertible mapping. Note that the models may not be reliable in regions with no data points, for example, in the top right corner of the domain.

Since the dimensionality of the feature space is usually larger than two or three, problem (9) is intractable. However, this viewpoint can provide new insights and motivate improved learning algorithms. For example, [WYSO20] showed that adding some amount of viscosity to the PDE constraint in (9) can increase the robustness of classifiers to random perturbations of the inputs. They also proposed an efficient method based on the Feyman-Kac formula to scale to high dimensions. The interpretation also allows us to build bridges to optical flow and image registration, which may yield further understanding in the future.

Continuous-Time Generative Models

This section illustrates the advantages of continuous-time models for building flexible generative models.

In generative modeling, one learns complicated, often high-dimensional, data distributions from examples. The goal typically is to enable sampling and sometimes includes estimating the densities of given examples. Rather than mapping input points to corresponding output points, as in supervised learning, a generative model maps a tractable reference distribution to a target distribution. In deep generative modeling, this mapping is called a generator, and it is represented by a deep neural network.

While the choice of reference distribution is arbitrary (as long as it is easy to sample from), most commonly, it is a standard Gaussian. To motivate the use of continuous-time models, we will set the dimension of the latent distribution, n, equal to the data distribution, m. We also assume both distributions are proper in $\mathbb{R}^n$, which enables the use of normalizing flows ; see [KPB20] for a recent review. When this assumption is violated (as is common in practice), other generative models, such as variational autoencoders or generative adversarial networks, are typically superior; a general introduction to generative modeling is given in [RH21]. We illustrate the generative modeling problem in Figure 2.

Under these assumptions, the idea of normalizing flows is to learn a diffeomorphic generator that maps reference to target; that is, we seek to find an invertible F_θ such that both F_θ and F_θ^{-1} are continuously differentiable. Since the reference distribution is a standard Gaussian, its density, which we denote by π_X, is easy to compute. To estimate the density of a point y from the unknown target distribution under

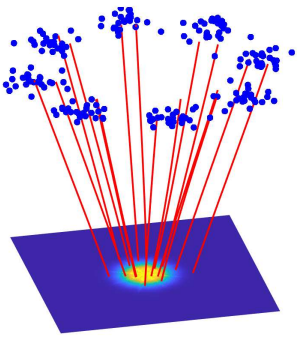

Figure 2: Illustrating the generative modeling problem. Given samples from the target distribution (represented by blue dots), we try to find an invertible transformation (represented by red lines) to a simple target distribution (a standard Gaussian).

the current generator F_θ, we can use the change of variable formula

$$\pi_Y(y) = \pi_X(F_\theta^{-1}(y)) \cdot \det \nabla F_\theta^{-1}(y). \tag{10}$$

Maximum likelihood training aims to find a parameter θ that maximizes the expected likelihood over all the samples. One typically considers minimizing the expected negative log-likelihood

$$\mathbb{E}_y \left[\frac{1}{2} \|F_\theta^{-1}(y)\|^2 - \log\det\left(\nabla F_\theta^{-1}(y)\right) \right], \tag{11}$$

where the first term is (up to an additive constant) the negative log-likelihood of the standard normal π_X. Even though the functional is convex in F_θ^{-1}, it is not convex in θ once we approximate it with a neural network. Therefore, numerical optimization schemes are required to compute an approximate minimizer.

The most critical question for normalizing flows is choosing the neural network architecture F_θ. For example, even though the classical multi-layer perception (1) can approximate any generator, it is generally not invertible, so it cannot be trained using (11). Trading off the expressiveness of the model with invertibility and computational considerations has led to various approaches. A key idea to build normalizing flows is to concatenate finitely many layers designed to have easy-to-compute inverses and Jacobian log determinants. However, this construction can limit expressiveness, and often, many layers are needed to approximate mappings in high dimensions. As discussed in more detail in [RH21], one can sometimes increase the approximation power of F_θ^{-1} by sacrificing the computational efficiency of evaluating the generator F_θ or vice versa.

Approaches that define the generator using a continuous-time model are known under the term continuous normalizing flows [GCB+18]. When defining $F_\theta(x) = z(1)$ as the terminal state of the neural ODE (4) and f_θ is sufficiently regular so that the ODE solution is defined uniquely, we can, at least formally, compute the inverse of the generator by integrating backward in time and defining $F_\theta^{-1}(y) = w(0)$ where

$$\frac{d}{dt} w = f_{\theta(t)}(w), \quad t \in [0,1), \quad w(1) = y. \tag{12}$$

One should always be careful when reversing the time in an ODE since, in general, it should not be assumed that the ODE is stable both forward and backward. However, in practice, using fairly arbitrary neural networks to represent f_θ has been found to induce negligible errors.

Let us also comment on the evaluation of the Jacobian log-determinant. When we define $F_\theta^{-1}(y) = w(0)$ as above, the instantaneous change of variables formula from [CRBD18, Appendix A] implies

$$\log\det(\nabla F_\theta^{-1}(y)) = \int_0^1 \mathrm{tr} \nabla f_{\theta(t)}(w) dt. \tag{13}$$

Numerically integrating the trace of the Jacobian using quadrature rules is often computationally more efficient than computing its log determinant. The computation of the log determinant can also be combined with the numerical ODE solver used to compute the inverse of the generator.

The training of the continuous-time generator F_θ can now be phrased as an optimal control problem

$$\begin{aligned} \min_{\theta,w} \ & \mathbb{E}_y \left[\frac{1}{2} \|w(0)\|^2 - \int_0^1 \mathrm{tr} \nabla f_{\theta(t)}(w) dt \right] \\ \text{s.t.} \ & \frac{d}{dt} w = f_{\theta(t)}(w), \ t \in [0,1), \\ & w(1) = y. \end{aligned} \tag{14}$$

It turns out that this control problem admits infinitely many solutions. As we shall see in the following section, it is possible for two different networks $f_\theta^{(1)}$ and $f_\theta^{(2)}$ to yield the same generator but follow different trajectories; impatient readers may skip ahead to Figure 3. Even though one often cannot see the differences in the created samples, realizing the non-uniqueness allows one to bias the search toward generators with more regular trajectories. It

also bridges generative modeling and optimal transport, which has a rich theory and long history.

The idea of penalizing transport costs has been investigated and shown practical benefits in [YK20, FJNO2011, OFLR21]. Since optimal transport in high dimensions is difficult, a tractable approach is to penalize transport costs, for example, by adding the functional

$$P_{\mathrm{OT}}[w, f_\theta] = \int_0^1 \frac{\alpha}{2} \|f_{\theta(t)}(w)\|^2 dt \tag{15}$$

to the objective function in (14). Here, the parameter α balances matching the distributions (for $\alpha \ll 1$) and minimizing the transport costs (for $\alpha \gg 0$). It is possible to show that we obtain the optimal transport map for an appropriate choice of α and that trajectories become straight. This regularity can translate to practical benefits since the ODEs for the generator and its inverse become trivial to solve.

Adding transport costs to the generative modeling problem also provides exciting opportunities to analyze the training problem. To give a glimpse into this area, we note that the training problem can be written on the macroscopic level as the PDE-constrained optimization problem

$$\begin{aligned} \min_{\rho,\theta} \int_{\mathbb{R}^n} \int_0^1 \frac{\alpha}{2} \|f_{\theta(t)}(x)\|^2 \rho(t,x) dt - \log \rho(1,x) \pi_Y(x) dx \\ \text{s.t. } \partial_t \rho(t,x) + \nabla \cdot \big(f_{\theta(t)}(x) \rho(t,x)\big) = 0, \\ \rho(0,x) = \pi_X(x). \end{aligned} \tag{16}$$

Here, the PDE constraint for $t \in (0,1]$ is given by the continuity equation. The formulation above is a relaxed version of the classical dynamic optimal transport formulation [BB00]. To be concrete, above the terminal constraint $\rho(1,x) = \pi_Y$ is relaxed, and deviations are penalized by the second term of the objective. Similar to the dynamic OT problem, (16) can be reformulated into a convex variational problem, which can provide further insight but becomes impractical when the growth of n requires nonlinear function approximators.

Neural ODEs for Potential Mean Field Games

This section showcases Neural ODE's promise for overcoming the limitations of other numerical methods for simulating interactions within large populations of agents playing a non-cooperative game.

To show how neural ODEs arise naturally in the mean field limit of many games, let us generalize (16) to include objective functions that contain more general cost terms in, for example, using the objective functional

$$\begin{aligned} \mathcal{J}[\rho, f_\theta] = \int_0^1 \int_{\mathbb{R}^n} L(x, f_\theta) \rho(t,x) dx dt \\ + \int_0^1 \mathcal{F}(\rho(t,\cdot)) dt + \mathcal{G}(\rho(1,\cdot)), \end{aligned} \tag{17}$$

which consists of running cost given by the function $L : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ and the functional $\mathcal{F}$, and a terminal cost functional $\mathcal{G}$. Given an initial state of the population density π_X, finding the optimal strategy f_θ amounts to solving the mean field game

$$\begin{aligned} \min_{\rho,\theta} \mathcal{J}[\rho, f_\theta] \\ \text{s.t. } \partial_t \rho(t,x) + \nabla \cdot \big(f_{\theta(t)}(x) \rho(t,x)\big) = 0, \\ \rho(0,x) = \pi_X(x), \end{aligned} \tag{18}$$

where the continuity equation models the evolution of the population density ρ. This more general version of (16) can be used to model various non-cooperative differential games played by a large population of rational agents. Furthermore, the formulation allows one to analyze and solve a larger set of generative models beyond continuous normalizing flows [ZK23]. Some examples are listed in Table 1. Also, we provide two one-dimensional instances motivated by optimal transport and crowd motion to illustrate our problem setup and notation in Figure 3. For simplicity, we focus the observation on deterministic games but note that neural network techniques have also been proposed for stochastic MFGs governed by the Fokker Planck equation [LFL+21].

It is important to note that solving (18) in general remains a daunting task, especially when the dimension of the state space, n, is larger than three or four. In these cases, the curse of dimensionality affects traditional numerical methods that rely on meshes or grids to solve the continuity equation in (18). Unfortunately, many realistic use cases of MFGs arising in economics, social science, and other fields require considerably larger n to capture the state of the agents.

Deriving a neural ODE formulation for approximating the solution of a class of MFGs requires some calculus and theoretical tools. Here, we will briefly overview our approach in [ROL+20]. A key quantity for analyzing and solving the mean field game is its value function $\Phi : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$. An intuitive way to define it is via a microscopic perspective. Let $x \in \mathbb{R}^n$

	$L(x, f_\theta)$	$\mathcal{F}(\rho)$	$\mathcal{G}(\rho)$
optimal transport (OT)	$\frac{\alpha}{2}\|f_\theta\|^2$	$\int \rho \log(\rho) dx$	$\mathrm{KL}(\rho, \rho_Y)$
crowd motion	$\frac{\alpha}{2}\|f_\theta\|^2 + Q(x)$		
normalizing flow			$\mathbb{E}_y[-\log(\rho)]$
normalizing flow+OT	$\frac{\alpha}{2}\|f_\theta\|^2$		$\mathbb{E}_y[-\log(\rho)]$

Table 1: Examples of different potential mean field games that can be modeled via (18). We also recommend [ZK23] for a more exhaustive list, including score-based diffusion and Wasserstein gradient flows.

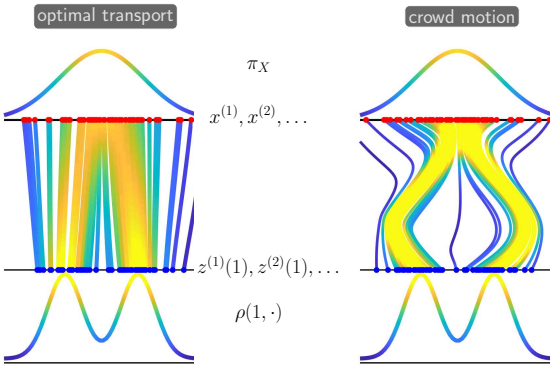

Figure 3: Illustration of potential mean field game versions of relaxed dynamic optimal transport (left) and crowd motion problems (right). Both cases use the standard Gaussian reference π_X (top) and the same Gaussian mixture as the target (bottom). As expected, in the optimal transport case, the trajectories are straight, whereas in the crowd motion case, the agents are curved to avoid an obstacle in the center of the domain. This example also shows that different dynamics can produce the same map F_θ.

be the state of an arbitrary agent at time $t \in [0, 1)$. Then, the value function Φ denotes the optimal cost to go for this agent and can be written as

$$\begin{aligned} \Phi(t, x) = &\min_{\theta, z} J_t[f_\theta, \rho, z], \\ &\text{s.t. } \frac{d}{dt} z = f_{\theta(s)}(z), \ s \in (t, 1] \\ &z(t) = x \end{aligned} \tag{19}$$

where ρ is the population density at the equilibrium, and we define the single-agent objective functional

$$\begin{aligned} J_t[f_\theta, \rho, z] = &G(z(1), \rho(1, z(1))) \\ &+ \int_t^T \left(L\left(z, f_{\theta(s)}(z)\right) + F(z, \rho(s, z)) \right) ds \end{aligned} \tag{20}$$

with F and G denoting the L^2 derivatives of $\mathcal{F}$ and $\mathcal{G}$, respectively. This makes this an MFG in potential form and ensures that solving the problem from the microscopic and macroscopic perspective leads to the same solution.

Evaluating J_t in (20) requires the agent to estimate the equilibrium density resulting from the collective behavior of the agents around its current trajectory. To this end, we solve the continuity equation in (18). Fortunately, characteristic curves of the continuity equation coincide with the trajectories of the agents. Hence, rather than solving the continuity equation everywhere to compute the density, we can update the densities along the trajectories as the agent's state evolves. Since most of the common choices listed in Table 1 require the log of the density, let us note that along the curves $z(\cdot)$ we have

$$\log \rho(t, z(t)) = \log \pi_X(x) - \int_0^t \mathrm{tr} \nabla f_{\theta(s)}(z) ds. \tag{21}$$

This allows us to eliminate the continuity equation in problem (19) without requiring a grid or a mesh. The resulting Lagrangian approach also has a crucial computational advantage since we can compute the trajectories and objective function values for several agents in parallel without the need to communicate.

The above observations and our experience from the previous section could also be used to obtain a neural ODE approach for potential mean field games. However, the learned solution may violate some theoretical properties. For example, the Pontryagin Maximum Principle shows that the optimal control, f^*, is related to the value function via the feedback form

$$f^*(x) = -\nabla_p H(x, \nabla \Phi(t, x)). \tag{22}$$

Here, the Hamiltonian H is the Fenchel dual of the running cost L defined by

$$H(x, p) = \sup_{f \in \mathbb{R}^n} \left\{ -p^\top f - L(x, f) \right\}. \tag{23}$$

For many choices of L that arise in practice, including the examples in Table 1, H can be computed analytically.

Even when the neural network can approximate the optimal policy, in our experience, finding weights such that properties such as (22) approximately hold is non-trivial. Clearly, choosing the network weights randomly will not yield an approximation that satisfies the feedback form, which means we must solve the learning problem well. Therefore, when H is available and straightforward to compute, we can approximate Φ_θ with a scalar-valued neural network and define the control implicitly via (22). Since the value function contains all the information about the MFG solution, approximating it directly can provide helpful insight.

Approximating the value function directly also allows us to incorporate more prior knowledge into the training problem. It is known that the value function solves the Hamilton Jacobi Bellman (HJB) equations

$$\begin{aligned} -\partial_t \Phi_\theta(t,x) + H(x, \nabla\Phi_\theta(t,x)) &= F(x, \rho(t,x)), \\ \Phi_\theta(1,x) &= G(x, \rho(1,x)). \end{aligned} \tag{24}$$

Here, the $-\partial_t$ emphasizes that this equation is backward in time. Lasry and Lions have shown that solving the above PDE in conjunction with the continuity equations is the necessary and sufficient condition for the problem [LL0703]. Even though solving (24) in high dimensions is affected by the curse of dimensionality, we can monitor the violations of the HJB equations along the trajectories and penalize them using some functional P_{HJB}.

To summarize the above observations, we can train the scalar deep neural network, Φ_θ, that approximates the value functions via the optimal control problem

$$\begin{aligned} &\min_\theta \mathbb{E}_{x\sim\pi_X}\left[J\left[f_\theta, \rho, z\right] + \beta P_{\text{HJB}}[\Phi_\theta, \rho, z]\right] \\ &\text{s.t. } \frac{d}{dt}\begin{pmatrix} z \\ \log(\rho(t,z)) \end{pmatrix} = \begin{pmatrix} f_\theta(z) \\ -\text{tr}\nabla f_\theta(z) \end{pmatrix} \\ &\qquad z(0) = x, \quad \log\rho(0,x) = \log\pi_X(x). \end{aligned} \tag{25}$$

Discussion and Outlook

We demonstrated how differential equations can be used to build continuous-time deep learning approaches. We highlighted a few ways this could lead to new insights and more efficient algorithms for supervised machine learning, generative modeling, and solving high-dimensional mean field games.

The key step is to transform the feature space incrementally using an ODE whose dynamics are represented by a deep neural network. Conceptually, this leads to infinitely deep networks whose artificial time loosely corresponds to the depth of the network. This relation is not precise since, depending on the choice of f_θ, the dynamics of the ODE can be scaled arbitrarily and even depend on trainable parameters. One can consider continuous-time architectures as infinitely deep, which sets them apart from traditional networks consisting of finitely many layers.

Earlier works on continuous-time learning that bring in ideas from differential equations include, for example, [RMKK$^+$92]. Some of the ideas in this work resemble the ones popularized by [CRBD18], but the latter contained other novel ideas, for example, the use of differential equations for generative models later extended in [GCB$^+$18]. The renewed interest in continuous-time models is probably related to the growth of computational resources, advances in numerical methods for solving ODEs and optimal control problems arising in their training, and larger datasets.

We inevitably omitted many important topics to keep the presentation short and coherent. Important examples of continuous-time architectures not governed by ODEs are the controlled differential equations [KMFL20] and non-local networks driven by fractional differential equations [AKLV20]. The above viewpoint can be extended to PDE architectures for input features that can be seen as grid functions.

More could also be said about numerical methods for continuous-time deep learning. An important question is whether first to optimize and then discretize (as, for example, in [CRBD18, GCB$^+$18]) or first to discretize and then optimize (as, for example, done in [OFLR21]). In a first-optimize-then-discretize setting, one solves the adjoint ODE to compute the gradient of the loss function with respect to the weights, the key ingredient for optimization algorithms. While this allows some flexibility in choosing the numerical integrators for the forward and adjoint equation (e.g., one can use different step sizes), the adjoint method requires storing or recomputing the entire trajectory of the features, which is computationally infeasible. In a discretize-then-optimize approach, one selects a numerical time integrator and integration points to discretize the neural ODE (4) and obtains a finite-dimensional optimization problem. Differentiating the discretized loss

function with respect to the weights is possible using automatic differentiation (also known as backpropagation) or analytically using the chain rule. The choice of the time integrator is crucial and provides opportunities to design novel network architectures that resemble ResNets but can be tailored to the network model; for example, one can mimic hyperbolic systems and use symplectic time integrators to ensure forward and backward stability [HR17] and save memory costs. An excellent in-depth discussion of these two paradigms in the context of neural ODEs is provided in [GKB19]. Some additional numerical results for time-series regression and generative modeling can be found in [OR20].

Acknowledgments

I thank {Levon Nurbekyan, Deepanshu Verma} for their careful reading of initial drafts and fruitful discussions. I am also grateful for the two anonymous reviewers who made excellent suggestions that improved this paper. My work on this article and related projects was partly supported by NSF awards DMS 1751636, DMS 2038118, AFOSR grant FA9550-20-1-0372, and US DOE Office of Advanced Scientific Computing Research Field Work Proposal 20-023231.

References

[AKLV20] Harbir Antil, Ratna Khatri, Rainald Löhner, and Deepanshu Verma, *Fractional deep neural network via constrained optimization*, Machine Learning: Science and Technology **2** (2020), no. 1, 015003.

[BB00] J D Benamou and Y Brenier, *A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem*, Numerische Mathematik (2000).

[BCE+19] Martin Benning, Elena Celledoni, Matthias J Ehrhardt, Brynjulf Owren, and Carola-Bibiane Schönlieb, *Deep learning as optimal control problems: Models and numerical methods*, Journal of Computational Dynamics **6** (2019), no. 2, 171–198.

[BCN16] Léon Bottou, Frank E Curtis, and Jorge Nocedal, *Optimization Methods for Large-Scale Machine Learning*, SIAM Review **60** (2016), no. 2, 223 311, available at 1606.04838.

[CRBD18] Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud, *Neural ordinary differential equations*, Advances in neural information processing systems, 2018.

[DDT19] Emilien Dupont, Arnaud Doucet, and Yee Whye Teh, *Augmented neural ODEs*, Advances in neural information processing systems, 2019.

[E1703] Weinan E, *A Proposal on Machine Learning via Dynamical Systems*, Communications in Mathematics and Statistics **5** (201703), no. 1, 1 11.

[FJNO2011] Chris Finlay, Joern-Henrik Jacobsen, Levon Nurbekyan, and Adam Oberman, *How to Train Your Neural ODE: the World of Jacobian and Kinetic Regularization*, International conference on machine learning, 202011, pp. 3154 –3164.

[GCB+18] Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud, *FFJORD: Free-form continuous dynamics for scalable reversible generative models*, International conference on learning representations, 2018.

[GKB19] Amir Gholami, Kurt Keutzer, and George Biros, *Anode: Unconditionally accurate memory-efficient gradients for neural odes*, arXiv preprint arXiv:1902.10298 (2019).

[HR17] Eldad Haber and Lars Ruthotto, *Stable architectures for deep neural networks*, Inverse Problems **34** (2017), no. 1, 014004, available at 1705.03341.

[HZRS1603] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Identity Mappings in Deep Residual Networks*, 201603, pp. 630 645.

[KL19] Patrick Kidger and Terry Lyons, *Universal Approximation with Deep Narrow Networks*, arXiv (2019), available at 1905.08539.

[KMFL20] Patrick Kidger, James Morrill, James Foster, and Terry Lyons, *Neural controlled differential equations for irregular time series*, Advances in Neural Information Processing Systems **33** (2020), 6696–6707.

[KPB20] Ivan Kobyzev, Simon Prince, and Marcus A. Brubaker, *Normalizing flows: An introduction and review of current methods*, IEEE Transactions on Pattern Analysis and Machine Intelligence **43** (2020), 3964–3979.

[LCTW18] Qianxiao Li, Long Chen, Cheng Tai, and E Weinan, *Maximum principle based algorithms for deep learning*, Journal of Machine Learning Research **18** (2018), no. 165, 1–29.

[LFL+21] Alex Tong Lin, Samy Wu Fung, Wuchen Li, Levon Nurbekyan, and Stanley J Osher, *Alternating the population and control neural networks to solve high-dimensional stochastic mean-field games*, Proceedings of the National Academy of Sciences **118** (2021), no. 31, e2024713118.

[LL0703] Jean-Michel Lasry and Pierre-Louis Lions, *Mean field games*, Japanese Journal of Mathematics **2** (200703), 229–260.

[LLS22] Qianxiao Li, Ting Lin, and Zuowei Shen, *Deep learning via dynamical systems: An approximation perspective*, Journal of the European Mathematical Society **25** (2022), no. 5, 1671–1709.

[MPP+20] Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama, *Dissecting neural odes*, Advances in neural information processing systems, 2020, pp. 3952–3963.

[OFLR21] Derek Onken, Samy Wu Fung, Xingjian Li, and Lars Ruthotto, *OT-Flow: Fast and accurate continuous normalizing flows via optimal transport*, Proceedings of the AAAI Conference on Artificial Intelligence, 2021, pp. 9223–9232.

[OR20] Derek Onken and Lars Ruthotto, *Discretize-optimize vs. optimize-discretize for time-series regression and continuous normalizing flows*, arXiv preprint arXiv:2005.13420 (2020).

[RH18] Lars Ruthotto and Eldad Haber, *Deep neural networks motivated by partial differential equations*, Journal of Mathematical Imaging and Vision **62** (2018), 352 –364.

[RH21] ———, *An introduction to deep generative modeling*, GAMM-Mitteilungen **44** (2021), no. 2, e202100008.

[RMKK+92] Ramiro Rico-Martinez, K Krischer, IG Kevrekidis, MC Kube, and JL Hudson, *Discrete-vs. continuous-time nonlinear signal processing of cu electrodissolution data*, Chemical Engineering Communications **118** (1992), no. 1, 25–48.

[ROL+20] Lars Ruthotto, S J Osher, Wuchen Li, Levon Nurbekyan, and Samy Wu Fung, *A machine learning framework for solving high-dimensional mean field game and mean field control problems*, Proceedings of the National Academy of Sciences **117** (2020), no. 17, 9183 –9193.

[SAP22] Michael E Sander, Pierre Ablin, and Gabriel Peyré, *Do Residual Neural Networks discretize Neural Ordinary Differential Equations?*, arXiv (2022), available at `2205.14612`.

[WYSO20] Bao Wang, Binjie Yuan, Zuoqiang Shi, and Stanley J. Osher, *Enresnet: Resnets ensemble via the feynman–kac formalism for adversarial defense and beyond*, SIAM Journal on Mathematics of Data Science **2** (2020), no. 3, 559–582, available at `https://doi.org/10.1137/19M1265302`.

[YK20] Liu Yang and George Em Karniadakis, *Potential flow generator with L_2 optimal transport regularity for generative models*, IEEE Transactions on Neural Networks and Learning Systems **33** (2020), no. 2, 528–538.

[ZK23] Benjamin J Zhang and Markos A Katsoulakis, *A mean-field games laboratory for generative modeling*, arXiv (2023), available at `2304.13534`.