Neural Flows: Efficient Alternative to Neural ODEs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Neural ordinary differential equations describe how values change in time. This is the reason why they gained importance in modeling sequential data, especially when the observations are made at irregular intervals. In this paper we propose an alternative by directly modeling the solution curves — the flow of an ODE — with a neural network. This immediately eliminates the need for expensive numerical solvers while still maintaining the modeling capability of neural ODEs. We propose several flow architectures suitable for different applications by establishing precise conditions on when a function defines a valid flow. Apart from computational efficiency, we also provide empirical evidence of favorable generalization performance via applications in time series modeling, forecasting, and density estimation.

1 Introduction

Ordinary differential equations (ODEs) are among the most important tools for modeling complex systems, both in natural and social sciences. They describe the *instantaneous change* in the system, which is often an easier way to model physical phenomena, than specifying the whole system itself. For example, the change of the pendulum angle or the change in population can be naturally expressed in the differential form. Similarly, neural ODEs [9] describe how a quantity of interest, e..g, a vector $\boldsymbol{x}$ changes with time: $\dot{\boldsymbol{x}} = f(t, \boldsymbol{x}(t))$, f being a neural network. Starting at some initial value $\boldsymbol{x}(t_0)$ we can find the result of this dynamic at any t_1:

$$\boldsymbol{x}(t_1) = \boldsymbol{x}(t_0) + \int_{t_0}^{t_1} f(t, \boldsymbol{x}(t))\,\mathrm{d}t = \mathrm{ODESolve}(\boldsymbol{x}(t_0), f, t_0, t_1). \tag{1}$$

It is sufficient for f to be continuous in t and Lipschitz continuous in $\boldsymbol{x}$ to have a unique solution, by the Picard–Lindelöf theorem [12]. This mild condition is already satisfied by a large family of neural networks. In most practically relevant scenarios, the integral in Equation 1 has to be solved numerically, requiring a trade-off between computation cost and numerical precision. Much of the follow up work to [9] focused on retaining expressive dynamics while requiring fewer solver evaluations [21, 36].

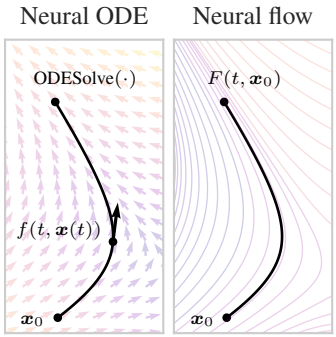

Figure 1: (Left) ODE requires numerical solver. (Right) Our approach returns solutions directly.

In the machine learning context we are given a set of initial conditions (often at $t_0 = 0$) and a loss function for the solution evaluated at time t_1. An example is modeling time series where we evolve the latent state in continuous time and use it to predict the observed measurements [15]. However, unlike applications in other domains, such as a pendulum, the function f is completely

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

unknown and needs to be learned from data. Thus, neural networks are used to model f, for their ability to capture complex dynamics; however note that this comes at the cost of the ODE being non-interpretable.

Since solving an ODE is expensive, we want to find a way to keep the desired properties of neural ODEs at a much smaller computation cost. If we take a step back, we see that neural ODEs take initial values as inputs and return non-intersecting solution curves (Figure 1). In this paper we propose to model the solution curves directly, with a neural network, instead of specifying the derivative. That is, given an initial condition we return the solution with a single forward pass through our network. Straight away, this leads to improvements in computation performance because we avoid using ODE solvers altogether. We show how our method can be used as a faster alternative to ODEs in existing models [7, 15, 33, 64], while improving the modeling performance. In the following, we derive the conditions that our method needs to satisfy and propose different architectures that implement them.

2 Neural flows

In this section, we present our method, *neural flows*, that directly models the solution curve of an ODE with a neural network. For simplicity, let us briefly assume that the initial condition $\boldsymbol{x}_0 = \boldsymbol{x}(t_0)$ is specified at $t_0 = 0$. We handle the general case shortly. Then, Equation 1 can be written as $\boldsymbol{x}(t) = F(t, \boldsymbol{x}_0)$, where F is the solution to the initial value problem, $\dot{\boldsymbol{x}} = f(t, \boldsymbol{x}(t)), \boldsymbol{x}_0 = \boldsymbol{x}(0)$. We will model F with a neural network. For this, we first list the conditions that F must satisfy so that it is a solution to some ODE. Let $F : (0, T) \times \mathbb{R}^d \to \mathbb{R}^d$ be a smooth function satisfying:

i) $F(0, \boldsymbol{x}_0) = \boldsymbol{x}_0$, (initial condition)

ii) $F(t, \cdot)$ is invertible, $\forall t$. (uniqueness of the solution given the initial value $\boldsymbol{x}_0$; i.e., the curves specified by F corresponding to different initial values should not intersect for any t)

There is an exact correspondence between a function F with the above properties and an ODE defined with f such that the derivative $\frac{\mathrm{d}}{\mathrm{d}t} F(t, \boldsymbol{x}_0)$ matches $f(t, \boldsymbol{x}(t))$ everywhere, given $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ [46, Theorem 9.12]. In general, we can say that f defines a vector field and F defines a family of integral curves, also known as the *flow* in mathematics (not to be confused with normalizing flow). As F will be parametrized with a neural network, condition i) requires that its parameters must depend on t such that we have the identity map at $t = 0$.

Note that by providing $\boldsymbol{x}_0$ we define a smooth trajectory $F(\cdot, \boldsymbol{x}_0)$ — the solution to some ODE with the initial condition at $t_0 = 0$. If we relax the restriction $t_0 = 0$ and allow $\boldsymbol{x}_0$ to be specified at an arbitrary $t_0 \in \mathbb{R}$, the solution can be obtained with a simple procedure. We first go back to the case $t = 0$ where we obtain the corresponding "initial" value $\hat{\boldsymbol{x}}_0 := \boldsymbol{x}(0) = F^{-1}(t_0, \boldsymbol{x}_0)$. This then gives us the required solution $F(\cdot, \hat{\boldsymbol{x}}_0)$ to the original initial value problem. Thus, we often prefer functions with an analytical inverse.

Finally, we tackle implementing F. The second property instructs us that the function $F(t, \cdot)$ is a diffeomorphism on $\mathbb{R}^d$. We can satisfy this by drawing inspiration from existing works on normalizing flows and invertible neural networks [e.g., 16, 2]. In our case, the parameters must be conditioned on time, with identity at $t = 0$. As a starting example, consider a linear ODE $f(t, \boldsymbol{x}(t)) = \boldsymbol{A}\boldsymbol{x}(t)$, with $\boldsymbol{x}(0) = \boldsymbol{x}_0$. Its solution can be expressed as $F(t, \boldsymbol{x}_0) = \exp(\boldsymbol{A}t)\boldsymbol{x}_0$, where $\exp$ is the matrix exponential. Here, the learnable parameters $\boldsymbol{A}$ are simply multiplied by t to ensure condition i); and given fixed t, the network behaves as an invertible linear transformation. In the following we propose other, more expressive functions suitable for applications such as time series modeling.

2.1 Proposed flow architectures

ResNet flow. A single residual layer $\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + g(\boldsymbol{x}_t)$ [29] bears a resemblance to Equation 1 and can be seen as a discretized version of a continuous transformation which inspired the development of neural ODEs. Although plain ResNets are not invertible, one could use spectral normalization [25] to enforce a small Lipschitz constant of the network, which guarantees invertibility [65, Theorem

9.23]; see e.g., Behrmann et al. [2]. Thus, ResNets become a natural choice for modeling the solution curve F resulting in the following extension — ResNet flow:

$$F(t, \boldsymbol{x}) = \boldsymbol{x} + \varphi(t) g(t, \boldsymbol{x}), \tag{2}$$

where $\varphi : \mathbb{R} \to \mathbb{R}^d$. This satisfies properties i) and ii) from above when $\varphi(0) = \mathbf{0}$ and $|\varphi(t)_i| < 1$; and $g : \mathbb{R}^{d+1} \to \mathbb{R}^d$ is an arbitrary contractive neural network ($\mathrm{Lip}(g) < 1$). One simple choice for φ is a $\tanh$ function. The inverse of F can be found via fixed point iteration similar to [2].

GRU flow. Time series data is traditionally modeled with recurrent neural networks, e.g., with a GRU [10], such that the hidden state $\boldsymbol{h}_{t-1}$ is updated at fixed intervals with the new observation $\boldsymbol{x}_t$:

$$\boldsymbol{h}_t = \mathrm{GRUCell}(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t) = \boldsymbol{z}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \odot \boldsymbol{c}_t, \tag{3}$$

where $\boldsymbol{z}_t$ and $\boldsymbol{c}_t$ are functions of $\boldsymbol{h}_{t-1}$ and $\boldsymbol{x}_t$. De Brouwer et al. [15] derived the continuous equivalent of this architecture called GRU-ODE (see Appendix A.1). Given the initial condition $\boldsymbol{h}_0 = \boldsymbol{h}(t_0)$, they evolve the hidden state $\boldsymbol{h}(t)$ with an ODE, until they observe new $\boldsymbol{x}_{t'}$ at time t', when they use Equation 3 to update it:

$$\bar{\boldsymbol{h}}_{t'} = \mathrm{ODESolve}(\boldsymbol{h}_0, \text{GRU-ODE}, t_0, t'), \quad \boldsymbol{h}_{t'} = \mathrm{GRUCell}(\bar{\boldsymbol{h}}_{t'}, \boldsymbol{x}_{t'}). \tag{4}$$

Here, we will derive the flow version of GRU-ODE. If we rewrite Equation 3 by regrouping terms: $\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + (1 - \boldsymbol{z}_t) \odot (\boldsymbol{c}_t - \boldsymbol{h}_{t-1})$, we see that GRU update acts as a single ResNet layer.

Definition 1. *Let f_z, f_r, f_c be any arbitrary neural networks and let $z(t, \boldsymbol{h}) = \alpha \cdot \sigma(f_z(t, \boldsymbol{h}))$, $r(t, \boldsymbol{h}) = \beta \cdot \sigma(f_r(t, \boldsymbol{h}))$, $c(t, \boldsymbol{h}) = \tanh(f_c(t, r(t, \boldsymbol{h}) \odot \boldsymbol{h}))$. Further, let $\varphi : \mathbb{R} \to \mathbb{R}^d$ be a continuous function with $\varphi(0) = \mathbf{0}$ and $|\varphi(t)_i| < 1$. Then the continuous evolution of GRU state in time is defined as:*

$$F(t, \boldsymbol{h}) = \boldsymbol{h} + \varphi(t)(1 - z(t, \boldsymbol{h})) \odot (c(t, \boldsymbol{h}) - \boldsymbol{h}). \tag{5}$$

Theorem 1. *A neural network defined by Equation 5 specifies a flow when functions f_z, f_r and f_c are contractive maps, i.e., $\mathrm{Lip}(f_\cdot) < 1$, and $\alpha = 2/5$, $\beta = 4/5$.*

We prove Theorem 1 in Appendix A.3 by showing that the second summand on the right hand side in Equation 5 satisfies Lipschitz constraint making the whole network invertible. Note that GRU flow (Equation 5) acts as a replacement to ODESolve in Equation 4. Alternatively, we can append $\boldsymbol{x}_t$ to the input of f_z, f_r and f_c, which would give us a continuous-in-time version of GRU.

Coupling flow. The disadvantage of both ResNet flow and GRU flow is the missing analytical inverse. To this end, we propose a continuous-in-time version of an invertible transformation based on splitting the input dimensions into two disjoint sets A and B, $A \cup B = \{1, 2, \ldots, d\}$ [16]. We copy the values indexed by B and transform the rest conditioned on $\boldsymbol{x}^B$ which gives us the coupling flow:

$$F(t, \boldsymbol{x})^A = \boldsymbol{x}^A \exp(\psi(t, \boldsymbol{x}^B)\varphi_\psi(t)) + \tau(t, \boldsymbol{x}^B)\varphi_\tau(t), \tag{6}$$

where ψ, τ are arbitrary neural networks and $\varphi_\psi(0) = \varphi_\tau(0) = \mathbf{0}$. We can easily see that this satisfies condition i), and it is invertible by design regardless of t [16]. Since some values stay constant in a single layer, we apply multiple consecutive transformations, choosing different partitions A and B.

For all three models we can stack multiple layers $F = F_1 \circ \cdots \circ F_n$ and still define a proper flow since the composition of invertible functions is invertible, and consecutive identity maps give identity.

We can think of φ (including $\varphi_\psi, \varphi_\tau$) as a time embedding function that has to be zero at $t = 0$. Since it is a function of a single variable, we would like to keep the complexity low and avoid using general neural networks in favor of interpretable and expressive basis functions. A simple example is linear dependence on time $\varphi(t) = \boldsymbol{\alpha} t$, or $\tanh(\boldsymbol{\alpha} t)$ for ResNet flow. We use these in the experiments. An alternative, more powerful embedding consists of Fourier features $\varphi(t)_i = \sum_k \boldsymbol{\alpha}_k \sin(\boldsymbol{\beta}_k t)$.

2.2 On approximation capabilities

We are interested in how well neural flows can approximate neural ODEs. Historically, the Cauchy-Kovalevskaya theorem [13, p. 39] tells us that, if the derivative f is an analytic function, then the

solution curve F is analytic too. This hints that as long as we have expressive enough networks, we can approximate any ODE. Recently, it has been shown that both the affine coupling transformations [70] and neural ODEs [47, 71] belong to the same class of universal approximators for smooth invertible functions. The ResNet flow, as defined in Equation 2, is essentially an Euler discretization, meaning it is enough to stack appropriately many layers to uniformly approximate any ODE [47]. We can also analytically take the derivative $\frac{\mathrm{d}}{\mathrm{d}t}F$ to obtain a neural network that can be seen as a neural ODE. Other results [19, 76] consider limitations of neural ODEs in modeling general homeomorphisms, e.g., mapping $x \mapsto -x$. They also propose the solution by adding dimensions to the input $\boldsymbol{x}$. However, for this to work, the network needs to always output zeros on the corresponding padded dimensions which cannot be guaranteed. A better solution would be to relax the conditions that F needs to satisfy. For example, if we only require that the initial condition is satisfied, completely ignoring invertibility, we can remove the Lipschitz constraint in Equation 2. Since neural flows offer such flexibility, they might be of more practical relevance in these use-cases.

3 Applications

In this section we review two main applications of neural ODEs: modeling irregularly-sampled time series and density estimation. We describe the existing modeling approaches and propose extensions using neural flows. In Section 4 we will use models presented here to qualitatively and quantitatively compare neural flows with neural ODEs.

3.1 Continuous-time latent variable models

Autoregressive [57, 66] and state space models [31, 63] have achieved considerable success modeling regularly-sampled time series. However, many real-world applications do not have a constant sampling rate and may contain missing values, e.g., in healthcare we have very sparse measurements at irregular time intervals. Here we describe how our neural flow models can be used in such scenario.

Encoder. In this setting, we are given a sequence of observations $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, $\boldsymbol{x}_i \in \mathbb{R}^d$ at times $\boldsymbol{t} = (t_1, \ldots, t_n)$. To represent this type of data, previous RNN-based works relied on exponentially decaying hidden state [6], time gating [54], or simply adding time as an additional input [18]. More recently, various ODE-based models built on top of RNNs to evolve the hidden state between observations in continuous time, giving rise to, e.g., ODE-RNN [64], while outperforming previous approaches. Another model is GRU-ODE [15], which we already described in Equation 4. We proposed the GRU flow (Equation 5) that can be used as a straightforward replacement.

Lechner and Hasani [45] showed that simply evolving the hidden state with a neural ODE can cause vanishing or exploding gradients, a known issue in RNNs [3]. Thus, they propose using an LSTM-based [30] model instead. Following the notation from Equation 4, the update is as follows:

$$\bar{\boldsymbol{h}}_{t'} = \text{ODESolve}(\boldsymbol{h}_0, f, t_0, t'), \quad (\boldsymbol{c}_{t'}, \boldsymbol{h}_{t'}) = \text{LSTMCell}((\boldsymbol{c}_0, \bar{\boldsymbol{h}}_{t'}), \boldsymbol{x}_{t'}). \tag{7}$$

The difference to ODE-RNN [64] is introducing another hidden state that is not updated continuously in time, which in turn allows gradient propagation via internal LSTM gating. Here again, we simply replace the ODESolve with the ResNet or coupling flow to obtain a neural flow model.

Decoder. Once we have a hidden state representation $\boldsymbol{h}_i$ of the irregularly-sampled sequence up to $\boldsymbol{x}_i$, we are interested in making future predictions. The ODE based models continue evolving the hidden state using a numerical solver to get the representation at time t_{i+1}, with $\boldsymbol{h}_{i+1} = \text{ODESolve}(\boldsymbol{h}_i, f, t_i, t_{i+1})$. With neural flows we can simply pass the next time point t_{i+1} into F and get the next hidden state directly. In the following we show how the presented encoder-decoder model is used in both the smoothing and filtering approaches for irregular time series modeling.

Smoothing approach. The given sequence of observations $(\boldsymbol{X}, \boldsymbol{t})$ is modeled with latent variables or states $(\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n) \sim \mathbb{R}^h$, such that $\boldsymbol{x}_i \sim p(\boldsymbol{x}_i|\boldsymbol{z}_i)$, conditionally independent of other $\boldsymbol{x}_j$ [9, 64]. There is a predesignated prior state $\boldsymbol{z}_0$ at $t = 0$ from which the latent state is assumed to evolve continuously. More precisely, if z_0 is a sample from the initial latent state $\boldsymbol{z}_0$, then a latent state sample at any future time step t is given by $z_t = F(t, z_0)$.

Since the exact inference on the initial state $\boldsymbol{z}_0$, $p(\boldsymbol{z}_0|\boldsymbol{X}, \boldsymbol{t})$, is intractable, we proceed by doing approximate inference following the variational auto-encoder approach [9, 64]. We use an LSTM-based neural flow encoder (Equation 7 with ODESolve replaced by a flow) that processes $(\boldsymbol{X}, \boldsymbol{t})$ and outputs the approximate posterior parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ from the last state, $q(\boldsymbol{z}_0|\boldsymbol{X}, \boldsymbol{t}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$. The decoder returns all z_i deterministically at times $\boldsymbol{t}$ with $F(t, z_0)$, with initial condition $z_0 \sim q(\boldsymbol{z}_0|\boldsymbol{X}, \boldsymbol{t})$. For the latent state at an arbitrary t_i, the target is generated according to the model $\boldsymbol{x}_i \sim p(\boldsymbol{x}_i|\boldsymbol{z}_i)$. Given $p(\boldsymbol{z}_0) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{1})$, the overall model is trained by maximizing the evidence lower bound:

$$\text{ELBO} = \mathbb{E}_{z_0 \sim q(\boldsymbol{z}_0|\boldsymbol{X}, \boldsymbol{t}))}[\log p(\boldsymbol{X})] - \text{KL}[q(\boldsymbol{z}_0|\boldsymbol{X}, \boldsymbol{t})||p(\boldsymbol{z}_0)]. \tag{8}$$

Using continuous time models to process irregularly-sampled observations, as well as for evolving the latent state, brings up multiple advantages. First, the irregular time points are automatically taken care of without any explicit processing. Second, the decoder is not bound to discrete time but can make predictions at any, and as many time points as required, allowing us to do reconstruction, missing value imputation and forecasting. These properties hold whether we use neural flows or ODEs, but our approach is more computationally efficient, which matters as we scale to bigger data.

Filtering approach. In contrast to the previous approach, we can alternatively do the inference in an online fashion at each of the observed time points, i.e., estimating the posterior $p(\boldsymbol{z}_i|\boldsymbol{x}_{1:i}, \boldsymbol{t}_{1:i})$ after seeing observations until the current time step i. This is known as filtering. Here, the prediction for future time steps is done by evolving the posterior corresponding to the final observed time point $p(\boldsymbol{z}_n|\boldsymbol{X}, \boldsymbol{t})$ instead of the initial time point $p(\boldsymbol{z}_0|\boldsymbol{X}, \boldsymbol{t})$, as was done in the smoothing approach.

In this paper, we follow the general approach suggested by De Brouwer et al. [15] for capturing non-linear dynamics for irregular time series. We use GRU flow (instead of GRU-ODE) for evolving the hidden state $\boldsymbol{h}_i \in \mathbb{R}^h$. The hidden state is updated with the incoming observation $\boldsymbol{x}_i$, which is used to output the mean and variance of the approximate posterior $q(\boldsymbol{z}_i|\boldsymbol{x}_{1:i}, \boldsymbol{t}_{1:i})$. The log-likelihood cannot be computed exactly under this model so [15] suggest using a custom objective that is the analogue to Bayesian filtering (see Appendix A.2 for details). Unlike [15], which needs to solve the ODE for every observation, our method only needs a single pass through the network per observation.

3.2 Temporal point processes

Sometimes temporal data is measured irregularly *and* the times at which we observe the events come from some underlying process modeled with temporal point processes (TPPs). For example, we can use TPPs to model the times of messages between users. One example type of behavior we want to capture is excitation [28], e.g., observing one message increases the chance of seeing other soon after.

A realization of a TPP on an interval $[0, T]$ is an increasing sequence of arrival times $\boldsymbol{t} = (t_1, \ldots, t_n)$, $t_i \in [0, T]$, where n is a random variable. The model is defined with an intensity function $\lambda(t)$ that tells us how many events we expect to see in some bounded area [14]. The intensity has to be positive. We define the history $\mathcal{H}_{t_i}$ as the events that precede t_i, and further define the conditional intensity function $\lambda^*(t)$ which depends on history. For convenience, we can also work with inter-event times $\tau_i = t_i - t_{i-1}$, without losing generality. We train the model by maximizing the log-likelihood:

$$\log p(\boldsymbol{t}) = \sum_i^n \log \lambda^*(t_i) - \int_0^T \lambda^*(s)\, \mathrm{d}s. \tag{9}$$

Previous works used autoregressive models (e.g., RNNs) to represent the history with a fixed-size vector $\boldsymbol{h}_i$ [18]. The intensity function can correspond to a simple distribution [18] or a mixture of distributions [67]. Then the integral in Equation 9 can be computed exactly. Another possibility is modeling $\lambda(t)$ with an arbitrary neural network which requires Monte Carlo integration [52]. On the other hand, Jia and Benson [33] propose jump ODE model that evolves the hidden state $\boldsymbol{h}(t)$ with an ODE and updates the state with new observations, similar, e.g., ODE-LSTM from before. In this case, obtaining the hidden state and solving the integral in Equation 9 can be done in a single solver call.

Marked point processes. Often we are also interested in what type of an event happened at time point t_i. Thus, we can assign the observed type $\boldsymbol{x}_i$, also called mark, and model the arrival times

and marks jointly: $p(\boldsymbol{t}, \boldsymbol{X}) = p(\boldsymbol{t})p(\boldsymbol{X}|\boldsymbol{t})$. Written like this, we can keep the model for arrival times (Equation 9) and add a neural network that inputs the history $\boldsymbol{h}_i$ and the next time point t_{i+1} and outputs the probabilities for each mark type. Special case of $\boldsymbol{x}_i \in \mathbb{R}^d$ is covered in the next section.

3.3 Time-dependent density estimation

Normalizing flows (NFs) define densities with invertible transformations of random variables. That is, given a random variable $\boldsymbol{z} \sim q(\boldsymbol{z})$, $\boldsymbol{z} \in \mathbb{R}^d$ and an invertible function $F : \mathbb{R}^d \to \mathbb{R}^d$, we can compute the probability density function of $\boldsymbol{x} = F(\boldsymbol{z})$ with the change of variables formula [60]: $p(\boldsymbol{x}) = q(\boldsymbol{z})|\det J_F(\boldsymbol{z})|^{-1}$, where J_F is the Jacobian of F. As we can see, it is important to define a function F that is easily invertible and has a tractable determinant of the Jacobian. One example is the coupling NF [16], which we used to construct the coupling flow in Equation 6. Other tractable models include autoregressive [40, 59] and matrix factorization based NFs [4, 39].

In contrast to this, Chen et al. [9] define the transformation with an ODE: $f(t, \boldsymbol{z}(t)) = \frac{\partial}{\partial t}\boldsymbol{z}(t)$. This allows them to define the instantaneous change in log-density as well as the continuous equivalent to the change of variables formula, giving rise to the continuous normalizing flow (CNF):

$$\frac{\partial}{\partial t} \log p(\boldsymbol{z}(t)) = -\mathrm{tr}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right), \quad \log p(\boldsymbol{x}) = \log q(\boldsymbol{z}(t_0)) - \int_{t_0}^{t_1} \mathrm{tr}\left(\frac{\partial f}{\partial \boldsymbol{z}(t)}\right), \tag{10}$$

where $t_0 = 0$ and $t_1 = 1$ are usually fixed. The neural network f can be arbitrary as long as it gives unique ODE solutions. This offers an advantage when we need special structure of f that cannot be easily implemented with the discrete NFs, e.g., in physics we often require equivariant transformations [42]. Besides the cost of running the solver, calculating the trace at each step in Equation 10 becomes intractable as the dimension of data grows, so one resorts to stochastic estimation [26]. A similar approximation method is used for estimating the determinant in an invertible ResNet model [2]. Again, if we consider a linear ODE, we can easily show that calculating the trace and calculating the determinant of the corresponding flow is equivalent.

However, we are not interested in comparison between different normalizing flows for stationary densities [see e.g., 41], since *flow endpoints* t_0 and t_1 are always fixed; thus, our models would be reduced to the discrete NFs. Recently, Chen et al. [7] demonstrated how CNFs can evolve the densities in continuous time, with varying t_0 and t_1, which proves useful for spatio-temporal data. We will show how to do the same with our coupling flow, something that has not been explored before.

Spatio-temporal processes. We reuse the notation from Section 3.2 to denote the arrival times with $\boldsymbol{t}$ and marks with $\boldsymbol{X}$, $\boldsymbol{x}_i \in \mathbb{R}^d$, which are now continuous variables. Values $\boldsymbol{x}_i$ often correspond to locations of events, e.g., earthquakes [55] or disease outbreaks [53]. We use the temporal point processes from Section 3.2 to model $p(\boldsymbol{t})$, and are only left with the conditional density $p(\boldsymbol{X}|\boldsymbol{t})$. Chen et al. [7] propose several models for this, the first one being the time-varying CNF where $p(\boldsymbol{x}_i|t_i)$ is estimated by integrating Equation 10 from $t_0 = 0$ to observed t_i. Using our affine coupling flow as defined in Equation 6 we can write:

$$p(\boldsymbol{x}_i|t_i) = q(F^{-1}(t_i, \boldsymbol{x}_i))|\det J_{F^{-1}}(\boldsymbol{x}_i)|, \tag{11}$$

where q is the base density (defined with any NF) and the determinant is the product of the diagonal values of the Jacobian w.r.t. $\boldsymbol{x}_i$, which are simply $\exp$ terms from Equation 6 [16]. The density p evolves with time, the same way as in the CNF model, but without using the solver or trace estimation. To generate new realizations at t_i, we first sample from q to get $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$, then solve $F(t_i, \boldsymbol{x}_0)$.

An alternative model, attentive CNF [7], is more expressive compared to the time-varying CNF and more efficient than jump ODE models [7, 33]. The probability density of $\boldsymbol{x}_i$ depends on all the previous values $\boldsymbol{x}_{j<i}$ through the attention mechanism [74]. In our model, we represent all the previous points $\boldsymbol{x}_{j<i}$ with an attention encoder and define a conditional coupling NF $p(\boldsymbol{x}_i|t_i, \boldsymbol{x}_{j<i})$. We describe the full model in Appendix A.5. Both of the previous models can also use ResNet flow, but the benefits over ODEs vanish since the determinant and the inverse require iterative procedure.

	MuJoCo	Activity		Physionet	
	MSE	MSE	Accuracy	MSE	AUC
Neural ODE	8.403±0.142	6.390±0.136	0.756±0.013	**4.833±0.078**	0.777±0.012
Coupling flow	**4.217±0.147**	6.579±0.049	0.752±0.012	4.860±0.070	**0.788±0.004**
ResNet flow	5.147±0.171	**6.279±0.098**	**0.760±0.004**	4.903±0.125	0.784±0.010

Table 1: Test mean squared error (lower is better) and accuracy/area under curve (higher is better). Best result is bolded, result within one standard deviation is highlighted. Averaged over 5 runs.

4 Experiments

In this section we show that flow-based models can match or outperform ODEs at a smaller computation cost, both in latent variable time series modeling, as well as TPPs and time-dependent density estimation. To make fair comparison, we used recently introduced reparametrization trick for ODEs that allows faster mini-batching [7], and the semi-norm trick for faster backpropagation [37], making the models more competitive compared to the original works. In all experiments we split the data into train, validation and test set; train with early stopping and report results on test set. We use Adam optimizer [38]. For training we use two different machines, one with 3.4GHz processor and 32GB RAM and another with 61GB RAM and NVIDIA Tesla V100 GPU 16GB. All datasets are publicly available, we include them with the code to reproduce all the results in the supplementary material.

Synthetic data. We compare the performance of neural ODEs and neural flows on periodic signals and data generated from autonomous ODEs. Full setup and results are presented in Appendix B. In short, we observe that training with adaptive solvers [17] is slower compared to fixed-step solvers. In this case, however, we are not guaranteed invertibility [58], which can be an issue in, e.g., density estimation. With the same setup, our models are an order of magnitude faster. Finally, neural ODEs struggle with non-smooth signals while neural flows, which also only output smooth dynamics, perform much better. Neural flows are also better at extrapolating, although none of the models excel.

Smoothing approach. Following [64], we use three datasets: MuJoCo, Physionet, and Activity. MuJoCo is created from a simple physics simulation "Hopper" [69] by randomly sampling initial positions and velocities and letting dynamics evolve deterministically in time. There are 10000 sequences, with 100 time steps and 14 features. Activity contains 6554 time series of 3d positions of 4 sensors attached to an individual. The goal is to classify one of the 7 possible activities (e.g., walking, lying, etc.). Physionet [68] contains 8000 time series and 37 features of patients' measurements from the first 48 hours after being admitted to ICU. The goal is to predict the mortality.

We use the encoder-decoder model (Section 3.1) and maximize Equation 8. We use the same number of hidden layers and the same size of latent states for both the neural ODE, coupling flow and ResNet flow, giving approximately the same number of trainable parameters. The results in Table 1 show the reconstruction error and the accuracy of prediction. For better readability, we scale MSE scores. Neural flows outperform ODE models, and are within the confidence interval in Physionet reconstruction task. Our method offers more than two times speed-up during training (Figure 2), compared to an ODE using an Euler method. The gap is even larger for adaptive solvers. Note that Figure 2 shows an average time to run one training epoch which includes other operations, such as data fetching, state update etc. When comparing ODEs and flows alone, our method is much faster. Figure 2 shows that ODESolve contributes significantly to long training times.

Filtering approach. Following De Brouwer et al. [15], we use clinical database MIMIC-III [34], pre-processed to contain 21250 patients' time series, with 96 features. We also process newly released MIMIC-IV [24, 35] to obtain 17874 patients. The details are in Appendix D.2. The goal is to predict the next three measurements in the 12 hour interval after the observation window of 36 hours.

Table 2 shows that our GRU flow model (Equation 5) mostly outperforms GRU-ODE [15]. Additionally, we show that the ordinary ResNet flow with 4 stacked transformations (Equation 2) performs worse. The reason might be because it is missing GRU flow properties, such as boundedness. Similarly, an ODE with a regular neural network does not outperform GRU-ODE [15]. Finally, we report that the model with GRU flow requires 60% less time to run one training epoch.

	MIMIC-III		MIMIC-IV	
	MSE	NLL	MSE	NLL
GRU-ODE	0.507±0.005	**0.770±0.023**	0.379±0.005	0.748±0.045
ResNet flow	0.508±0.007	0.779±0.023	0.379±0.005	0.774±0.059
GRU flow	**0.499±0.004**	0.781±0.041	**0.364±0.008**	**0.734±0.054**

Table 2: Forecasting on healthcare data averaged over 5 runs (lower is better).

		MOOC		Reddit		Wiki	
	Discrete GRU	-0.4448	2.7563	-0.9299	1.8468	-0.5832	8.0527
Cont.	Jump ODE	0.8710	4.6118	0.1308	3.6654	-0.3115	10.6040
Cont.	Coupling flow	0.7694	5.5494	-0.1263	3.6312	-0.2807	9.7214
Cont.	ResNet flow	**-1.2379**	2.9466	**-1.2962**	2.3932	-1.2907	10.4368
Mix.	Jump ODE	-0.2626	3.0723	-1.0907	1.9057	**-1.3635**	**7.5537**
Mix.	Coupling flow	-0.4026	**2.5877**	-1.0933	**1.6817**	-1.2702	8.8018
Mix.	ResNet flow	-0.5664	3.0005	-1.0605	1.9491	-1.1937	8.5489

Table 3: Test NLL for TPP (left columns, per dataset) and marked TPP (right columns); full results in Appendix C. Cont. denotes models with continuous intensity, and Mix. with mixture distribution.

Temporal point processes. As we saw in Section 3.2, most of the TPP models consist of two parts: the encoder that processes the history, and the network that outputs the intensity. In the context of neural ODEs, we would like to answer: 1) whether having continuous state $\boldsymbol{h}(t)$ outperforms RNNs, and 2) if intertwining the hidden state evolution with the intensity outperforms other approaches. For this purpose we propose the following models. Jump ODE models $\boldsymbol{h}(t)$ and intensity $\lambda(t) = g(\boldsymbol{h}(t))$ continuously [33, 7], where g is a neural network. Neural flow model replaces the ODE with any of our proposed models and uses Monte Carlo integration to evaluate Equation 9. Note that this operation can be parallelized unlike solving an ODE. Further, we keep all the continuous time encoders (ODEs and flows) but after we observe the event, we output the stationary log-normal mixture for the next arrival time [67]. Finally, we use a discrete GRU with the same mixture decoder.

We use both synthetic and real-world data, following [56, 67]. We generate 4 synthetic datasets corresponding to homogeneous, renewal and self-correcting processes. For real-world data, we collect timesteps of forum posts (Reddit), interactions of students with an online course system (MOOC), and Wiki page edits [43]. The details of the data are in Appendix D.3.

We report the test negative log-likelihood on real-world data in Table 3, both for models trained with and without marks. Full results, including synthetic data can be found in the Appendix C. We note that all the models capture the synthetic data, although continuous intensity models struggle compared to those with the mixture distribution. We can see this is the case for real-world data too, where the mixture distribution usually outperforms the corresponding continuous intensity model. In general, neural flows are better than ODE-based models, with the exception of one ODE model on Wiki dataset. We can conclude that having a continuous encoder is preferred to a discrete RNN because it can capture the irregular time sequence better. However, there is no benefit in parametrizing the intensity function in a continuous fashion, especially since this is a much slower approach.

Table 8 in Appendix C shows the comparison of wall clock times. Comparing only continuous intensity models we can see that Monte Carlo integration is faster than solving an ODE. As expected, using the mixture distribution gives the best performance. Thus, our flow models offer more than an order of magnitude faster processing compared to ODEs with continuous intensity.

Spatial data. We compare the continuous normalizing flows with our continuous-time version of the coupling NF on time-dependent density estimation. We use two versions of each model: time-varying and attentive, as described in Section 3.3. Following Chen et al. [7], we use earthquake events in Japan (EQ) [73], Covid cases for the state of New Jersey [72], and locations of bike rentals (Bikes).

Results in Table 4 show the test NLL for spatial data. Our continuous coupling NF models perform better on all datasets. Since affine coupling is a simple transformation, we require bigger models with more parameters. At the same time, our models are still more than an order of magnitude faster.

	Bikes	Covid	EQ
Time-var. CNF	2.315	1.984	1.709
Attentive CNF	2.371	1.973	1.668
Time-var. coupling	**2.280**	**1.916**	1.633
Attentive coupling	2.330	1.926	**1.457**

Table 4: Test NLL for spatial datasets.

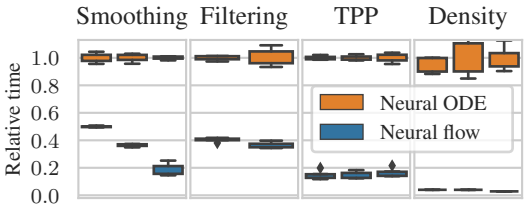

Figure 2: Comparing per-epoch wall-clock times.

5 Discussion

In this paper we presented neural flows as an efficient alternative to neural ODEs. We retain all the desirable properties of neural ODEs, without using numerical solvers. Our method outperforms the ODE based models in time series modeling and density estimation, at a much smaller computation cost. This brings the possibility to scale to larger datasets and models in the future.

Other related work. Early works on approximating the ODE solutions without numerical solvers used splines or radial basis functions [51, 48], or functions similar to modern ResNets [44]. More recently, [61] approximate the solution by minimizing the error of the solution points and of the boundary condition. Unlike these approaches, we do not approximate the solution to some given ODE but learn the solutions which corresponds to learning the unknown ODE. Also, our method guarantees that we always define a proper flow, as is required in certain applications.

ResNets were initially recognized as a discretization of dynamical systems [49, 75] and were used to tackle infinite depth [1, 50], stability [11, 27] and invertibility [5, 32]. We take a different approach and propose modified ResNets, among other, avoiding any iterative procedure. ResNets also lead to neural ODEs which have memory efficient backpropagation as one of the main features [20, 9]. Further, to combat solver inefficiency, many improvements have been proposed, such as adding regularization [21, 36, 23], improving training [22, 77, 37] and having faster inference [62].

Limitations. Defining a flow automatically defines an ODE, but since many ODEs do not have closed-form solutions, we cannot always find the *exact* flow corresponding to a particular ODE. This is usually not an issue since in most applications, such as those presented in Section 3, it is sufficient for both neural ODEs and neural flows to approximate an unknown dynamic. However, if we restrict ourselves to autonomous ODEs (fixed vector field in time), we cannot define a general neural flow that satisfies this condition. We further discuss this in Appendix A.6 and present a potential solution that involves a simple regularization.

Since neural ODEs reuse the same function f in the solver, essentially defining *implicit layers*, they can be more parameter efficient. Sometimes we might need more parameters to represent the same dynamic, as we observed in the density estimation task. But even here, the results show neural flows are more efficient. In the special setting with limited memory, we can resort to existing solutions [8].

Future work. In this work we designed neural flow models as invertible functions that satisfy initial condition using simple dependence on time. Although these models already outperform neural ODEs, it would be interesting to see if there are other ways to define a neural flow, and whether these architectures can outperform the ones we proposed here.

We applied our method to the main applications of neural ODEs: time series modeling and density estimation. In the future we hope to see neural flows adapted for other use cases as well.

Broader impact. We introduced a new method to replace neural ODEs. As such, it has a wide variety of potential applications, some of which we explored in this paper. We used several healthcare datasets and hope to see further applications of our method in this domain. At the same time, it is important to pay attention to data privacy and fairness when building such models, especially for sensitive applications, such as healthcare. One of the main benefits of our method is the reduced computation cost, which may imply energy savings.

References

[1] S. Bai, J. Z. Kolter, and V. Koltun. Deep equilibrium models. In *NeurIPS*, 2019.

[2] J. Behrmann, W. Grathwohl, R. T. Chen, D. Duvenaud, and J.-H. Jacobsen. Invertible residual networks. In *ICML*, 2019.

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

[4] R. v. d. Berg, L. Hasenclever, J. M. Tomczak, and M. Welling. Sylvester normalizing flows for variational inference. In *UAI 2018*, 2018.

[5] B. Chang, L. Meng, E. Haber, L. Ruthotto, D. Begert, and E. Holtham. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI*, 2018.

[6] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):1–12, 2018.

[7] R. T. Q. Chen, B. Amos, and M. Nickel. Neural spatio-temporal point processes. In *ICLR*, 2021.

[8] T. Chen, B. Xu, C. Zhang, and C. Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[9] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *NeurIPS*, 2018.

[10] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. In *Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation (SSST-8)*, 2014.

[11] M. Ciccone, M. Gallieri, J. Masci, C. Osendorfer, and F. Gomez. NAIS-Net: Stable deep networks from non-autonomous differential equations. In *NeurIPS*, 2018.

[12] E. A. Coddington and N. Levinson. *Theory of ordinary differential equations*. Tata McGraw-Hill Education, 1955.

[13] R. Courant and D. Hilbert. *Methods of Mathematical Physics: Partial Differential Equations*. John Wiley & Sons, 2008.

[14] D. Daley and D. Vere-Jones. An introduction to the theory of point processes: Volume I: Elementary theory and methods. 2007.

[15] E. De Brouwer, J. Simm, A. Arany, and Y. Moreau. GRU-ODE-Bayes: Continuous modeling of sporadically-observed time series. In *NeurIPS*, 2019.

[16] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using Real NVP. In *ICLR*, 2017.

[17] J. R. Dormand and P. J. Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

[18] N. Du, H. Dai, R. Trivedi, U. Upadhyay, M. Gomez-Rodriguez, and L. Song. RMTPP: Embedding event history to vector. In *KDD*, 2016.

[19] E. Dupont, A. Doucet, and Y. W. Teh. Augmented neural ODEs. In *NeurIPS*, 2019.

[20] P. E. Farrell, D. A. Ham, S. W. Funke, and M. E. Rognes. Automated derivation of the adjoint of high-level transient finite element programs. *SIAM Journal on Scientific Computing*, 35(4): C369–C393, 2013.

[21] C. Finlay, J.-H. Jacobsen, L. Nurbekyan, and A. M. Oberman. How to train your neural ODE. In *ICML*, 2020.

[22] A. Gholami, K. Keutzer, and G. Biros. ANODE: Unconditionally accurate memory-efficient gradients for neural ODEs. In *IJCAI*, 2019.

[23] A. Ghosh, H. S. Behl, E. Dupont, P. H. Torr, and V. Namboodiri. Steer: Simple temporal regularization for neural odes. In *NeurIPS*, 2020.

[24] A. L. Goldberger, L. A. N. Amaral, L. Glass, J. M. Hausdorff, P. C. Ivanov, R. G. Mark, J. E. Mietus, G. B. Moody, C.-K. Peng, and H. E. Stanley. Physiobank, Physiotoolkit, and Physionet: Components of a new research resource for complex physiologic signals. *Circulation*, 101(23): e215–e220, 2000.

[25] H. Gouk, E. Frank, B. Pfahringer, and M. J. Cree. Regularisation of neural networks by enforcing Lipschitz continuity. *Machine Learning*, 110(2):393–416, 2021.

[26] W. Grathwohl, R. T. Chen, J. Bettencourt, I. Sutskever, and D. Duvenaud. FFJORD: Free-form continuous dynamics for scalable reversible generative models. In *ICLR*, 2019.

[27] E. Haber and L. Ruthotto. Stable architectures for deep neural networks. *Inverse Problems*, 34 (1):014004, 2017.

[28] A. G. Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971.

[29] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE CCVPR*, 2016.

[30] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

[31] R. Hyndman, A. Koehler, K. Ord, and R. Snyder. *Forecasting with exponential smoothing. The state space approach*. 2008. doi: 10.1007/978-3-540-71918-2.

[32] J. Jacobsen, A. W. M. Smeulders, and E. Oyallon. i-revnet: Deep invertible networks. In *ICLR*, 2018.

[33] J. Jia and A. R. Benson. Neural jump stochastic differential equations. In *NeurIPS*, 2019.

[34] A. Johnson, T. Pollard, L. Shen, H. L. Li-Wei, M. Feng, M. Ghassemi, B. Moody, P. Szolovits, L. A. Celi, and R. G. Mark. MIMIC-III, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[35] A. Johnson, L. Bulgarelli, T. Pollard, S. Horng, L. A. Celi, and R. Mark. MIMIC-IV. 2021.

[36] J. Kelly, J. Bettencourt, M. J. Johnson, and D. Duvenaud. Learning differential equations that are easy to solve. In *NeurIPS*, 2020.

[37] P. Kidger, R. T. Chen, and T. Lyons. "Hey, that's not an ODE": Faster ODE adjoints via seminorms. In *ICML*, 2021.

[38] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

[39] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *NeurIPS*, 2018.

[40] D. P. Kingma, T. Salimans, R. Jozefowicz, X. Chen, I. Sutskever, and M. Welling. Improved variational inference with inverse autoregressive flow. In *NeurIPS*, 2016.

[41] I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[42] J. Köhler, L. Klein, and F. Noé. Equivariant flows: exact likelihood generative learning for symmetric densities. In *ICML*, 2020.

[43] S. Kumar, X. Zhang, and J. Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1269–1278, 2019.

[44] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

[45] M. Lechner and R. Hasani. Learning long-term dependencies in irregularly-sampled time series. In *NeurIPS*, 2020.

[46] J. M. Lee. *Introduction to Smooth Manifolds*. Springer, 2012.

[47] Q. Li, T. Lin, and Z. Shen. Deep learning via dynamical systems: An approximation perspective. *arXiv preprint arXiv:1912.10382*, 2019.

[48] Li Jianyu, Luo Siwei, Qi Yingjian, and Huang Yaping. Numerical solution of differential equations by radial basis function neural networks. In *IJCNN*, 2002.

[49] Q. Liao and T. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *arXiv preprint arXiv:1604.03640*, 2016.

[50] Y. Lu, A. Zhong, Q. Li, and B. Dong. Beyond finite layer neural networks: Bridging deep architectures and numerical differential equations. In *ICML*, 2018.

[51] A. J. Meade Jr and A. A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12): 1–25, 1994.

[52] H. Mei and J. M. Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. In *NeurIPS*, pages 6754–6764, 2017.

[53] S. Meyer, J. Elias, and M. Höhle. A space–time conditional intensity model for invasive meningococcal disease occurrence. *Biometrics*, 68(2):607–616, 2012.

[54] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased LSTM: Accelerating recurrent network training for long or event-based sequences. In *NeurIPS*, 2016.

[55] Y. Ogata and D. Vere-Jones. Inference for earthquake models: a self-correcting model. *Stochastic processes and their applications*, 17(2):337–347, 1984.

[56] T. Omi, N. Ueda, and K. Aihara. Fully neural network based model for general temporal point processes. In *NeurIPS*, 2019.

[57] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. In *SSW*, 2016.

[58] K. Ott, P. Katiyar, P. Hennig, and M. Tiemann. When are neural ODE solutions proper ODEs? *arXiv preprint arXiv:2007.15386*, 2020.

[59] G. Papamakarios, T. Pavlakou, and I. Murray. Masked autoregressive flow for density estimation. In *NeurIPS*, 2017.

[60] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *arXiv:1912.02762*, 2019.

[61] M. L. Piscopo, M. Spannowsky, and P. Waite. Solving differential equations with neural networks: Applications to the calculation of cosmological phase transitions. *Phys. Rev. D*, 2019.

[62] M. Poli, S. Massaroli, A. Yamashita, H. Asama, and J. Park. Hypersolvers: Toward fast continuous-depth models. In *NeurIPS*, 2020.

[63] S. S. Rangapuram, M. W. Seeger, J. Gasthaus, L. Stella, Y. Wang, and T. Januschowski. Deep state space models for time series forecasting. In *NeurIPS*, 2018.

[64] Y. Rubanova, R. T. Chen, and D. Duvenaud. Latent ODEs for irregularly-sampled time series. In *NeurIPS*, 2019.

[65] W. Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1976.

[66] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3):1181–1191, 2020.

[67] O. Shchur, M. Biloš, and S. Günnemann. Intensity-free learning of temporal point processes. In *ICLR*, 2020.

[68] I. Silva, G. Moody, D. J. Scott, L. A. Celi, and R. G. Mark. Predicting in-hospital mortality of icu patients: The physionet/computing in cardiology challenge 2012. In *2012 Computing in Cardiology*, pages 245–248. IEEE, 2012.

[69] Y. Tassa, Y. Doron, A. Muldal, T. Erez, Y. Li, D. d. L. Casas, D. Budden, A. Abdolmaleki, J. Merel, A. Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[70] T. Teshima, I. Ishikawa, K. Tojo, K. Oono, M. Ikeda, and M. Sugiyama. Coupling-based invertible neural networks are universal diffeomorphism approximators. In *NeurIPS*, 2020.

[71] T. Teshima, K. Tojo, M. Ikeda, I. Ishikawa, and K. Oono. Universal approximation property of neural ordinary differential equations. In *NeurIPS 2020 Workshop on Differential Geometry meets Deep Learning*, 2020.

[72] The New York Times. Coronavirus (Covid-19) data in the United States, 2020. URL https://github.com/nytimes/covid-19-data.

[73] U.S. Geological Survey. Earthquake catalogue (accessed May 15, 2021), 2020. URL https://earthquake.usgs.gov/earthquakes/search/.

[74] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[75] E. Weinan. A proposal on machine learning via dynamical systems. *Communications in Mathematics and Statistics*, 5(1):1–11, 2017.

[76] H. Zhang, X. Gao, J. Unterman, and T. Arodz. Approximation capabilities of neural ODEs and invertible residual networks. In *International Conference on Machine Learning*, pages 11086–11095, 2020.

[77] J. Zhuang, N. Dvornek, X. Li, S. Tatikonda, X. Papademetris, and J. Duncan. Adaptive checkpoint adjoint method for gradient estimation in neural ODE. In *ICML*, 2020.

Checklist

i) For all authors...

(a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 4 for experimental results.

(b) Did you describe the limitations of your work? [Yes] See Section 5.

(c) Did you discuss any potential negative societal impacts of your work? [Yes] We discuss broader impact in Section 5. We did not find any direct negative impact of our method.

(d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]

ii) If you are including theoretical results...

(a) Did you state the full set of assumptions of all theoretical results? [Yes] See Section 2.

(b) Did you include complete proofs of all theoretical results? [Yes] For example, Theorem 1 is proved in Appendix.

iii) If you ran experiments...

(a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In the supplementary material.

(b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In the main text, and the supplementary material.

(c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] See, e.g., Tables 1 and 2, and Appendix C.

(d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section 4.

iv) If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...

(a) If your work uses existing assets, did you cite the creators? [Yes] See Section 4 and supplementary material.

(b) Did you mention the license of the assets? [Yes] All datasets and codes we used are publicly available. We include the download links in the supplementary material.

(c) Did you include any new assets either in the supplemental material or as a URL? [Yes] We include our code in the supplementary material.

(d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes] MIMIC-III [34] and MIMIC-IV [35] are health records de-identified in accordance to HIPAA standards. Johnson et al. [34] state that the individual patient consent was waived.

(e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [Yes] Data we used does not contain personally identifiable information, partly because it was de-identified (e.g., MIMIC), or because it never contained it (e.g., earthquake data). No datasets contain offensive content.

v) If you used crowdsourcing or conducted research with human subjects...

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]