NEURAL ODE PROCESSES

Anonymous authors
Paper under double-blind review

ABSTRACT

Neural Ordinary Differential Equations (NODEs) use a neural network to model the instantaneous rate of change in the state of a system. However, despite their apparent suitability for dynamics-governed time-series, NODEs present a few disadvantages. First, they are unable to adapt to incoming data-points, a fundamental requirement for real-time applications imposed by the natural direction of time. Second, time-series are often composed of a sparse set of measurements, which could be explained by many possible underlying dynamics. NODEs do not capture this uncertainty. To this end, we introduce Neural ODE Processes (NDPs), a new class of stochastic processes determined by a distribution over Neural ODEs. By maintaining an adaptive data-dependent distribution over the underlying ODE, we show that our model can successfully capture the dynamics of low-dimensional systems from just a few data-points. At the same time, we demonstrate that NDPs scale up to challenging high-dimensional time-series with unknown latent dynamics such as rotating MNIST digits.

1 INTRODUCTION

Many time-series that arise in the natural world, such as the state of a harmonic oscillator, the populations in an ecological network or the spread of a disease, are the product of some underlying dynamics. Sometimes, as in the case of a video of a swinging pendulum, these dynamics are latent and do not manifest directly in the observation space (pixel space, in this case). Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018), which use a neural network to parametrise the velocity of an ODE, have become a natural choice for capturing the dynamics of such time-series (Çağatay Yıldız et al., 2019; Rubanova et al., 2019; Norcliffe et al., 2020; Kidger et al., 2020; Morrill et al., 2020).

However, despite their fundamental connection to dynamics-governed time-series, NODEs present certain limitations that hinder their adoption in these settings. Firstly, NODEs cannot adjust predictions as more data is collected without retraining the model. This ability is particularly important for real-time applications, where it is desirable that models adapt to incoming data points as time passes and more data is collected. Secondly, without a larger number of regularly spaced measurements, there is usually a range of plausible underlying dynamics that can explain the data. NODEs, however, only offer a single maximum likelihood prediction and are unable to learn stochastic dynamics. As many real-world time-series are comprised of sparse sets of measurements, often irregularly sampled, the model can fail to represent the diversity of suitable solutions.

To address these limitations, we introduce Neural ODE Processes (NDPs), a new class of stochastic processes governed by stochastic data-adaptive dynamics. Our probabilistic Neural ODE formulation relies on and extends the framework provided by Neural Processes (NPs), and runs parallel to other attempts to incorporate application-specific inductive biases in this class of models such as Attentive NPs (Kim et al., 2019), ConvCNPs (Gordon et al., 2019), and MPNPs (Day et al., 2020). We demonstrate that NDPs can adaptively capture many potential dynamics of low-dimensional systems when faced with limited amounts of data. Additionally, we show that our approach scales to high-dimensional time series with latent dynamics such as rotating MNIST digits (Casale et al., 2018).

2 BACKGROUND AND FORMAL PROBLEM STATEMENT

Problem Statement We consider modelling random functions $F : \mathcal{T} \to \mathcal{Y}$, where $\mathcal{T} = [t_0, \infty)$ represents time and $\mathcal{Y} \subset \mathbb{R}^d$ is a compact subset of $\mathbb{R}^d$. We assume F has a distribution $\mathcal{D}$, induced by another distribution $\mathcal{D}'$ over some underlying dynamics that govern the time-series. Given a specific instantation $\mathcal{F}$ of F, let $\mathrm{C} = \{(t_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}})\}_{i \in I_{\mathrm{C}}}$ be a set of samples from $\mathcal{F}$ with some indexing set I_{C}. We refer to C as the context points, as denoted by the superscript C. For a given context C, the task is to predict the values $\{\boldsymbol{y}_j^{\mathrm{T}}\}_{j \in I_{\mathrm{T}}}$ that $\mathcal{F}$ takes at a set of target times $\{t_j^{\mathrm{T}}\}_{j \in I_{\mathrm{T}}}$, where I_{T} is another index set. We call $\mathrm{T} = \{(t_j^{\mathrm{T}}, \boldsymbol{y}_j^{\mathrm{T}})\}$ the target set. Conventionally, as in Garnelo et al. (2018), the target set forms a superset of the context set and we have $\mathrm{C} \subseteq \mathrm{T}$.

During training, we assume access to a dataset of (potentially irregular) time-series sampled from F. We are interested in learning the underlying distribution over the dynamics as well as the induced distribution over functions. We note that when the dynamics are not latent and manifest directly in the observation space $\mathcal{Y}$, the distribution over ODE trajectories and the distribution over functions coincide.

Neural ODEs NODEs are a class of models that parametrize the velocity $\dot{\boldsymbol{z}}$ of a state $\boldsymbol{z}$ with the help of a neural network $\dot{\boldsymbol{z}} = f_\theta(\boldsymbol{z}, t)$. Given the initial time t_0 and target time t_i^{T}, NODEs predict the corresponding state $\hat{\boldsymbol{y}}_i^{\mathrm{T}}$ by performing the following integration and decoding operations:

$$\boldsymbol{z}(t_0) = h_1(\boldsymbol{y}_0), \qquad \boldsymbol{z}(t_i^{\mathrm{T}}) = \boldsymbol{z}(t_0) + \int_{t_0}^{t_i^{\mathrm{T}}} f_\theta(\boldsymbol{z}(t), t)dt, \qquad \hat{\boldsymbol{y}}_i^{\mathrm{T}} = h_2(\boldsymbol{z}(t_i^{\mathrm{T}})), \tag{1}$$

where h_1 and h_2 can be neural networks. When the dimensionality of $\boldsymbol{z}$ is greater than that of $\boldsymbol{y}$ and h_1, h_2 are linear, the resulting model is an Augmented Neural ODE (Dupont et al., 2019) with input layer augmentation (Massaroli et al., 2020). The extra dimensions offer the model additional flexibility as well as the ability to learn higher-order dynamics (Norcliffe et al., 2020).

When applied to time-series prediction, NODEs are typically trained by minimising the mean squared error (MSE) between $\hat{\boldsymbol{y}}_i^{\mathrm{C}}$ and $\boldsymbol{y}_i^{\mathrm{C}}$ over all time-steps $\{t_i^{\mathrm{C}}\}$ with $i \in I$. They provide only a maximum-likelihood prediction and accounting for additional data requires further training or retraining entirely.

Neural Processes (NPs) NPs model a random function $F : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X} \subseteq \mathbb{R}^{d_1}$ and $\mathcal{Y} \subseteq \mathbb{R}^{d_2}$. The NP represents a given instantiation $\mathcal{F}$ of F through the global latent variable $\boldsymbol{z}$, which parametrises the variation in F. Thus, we have $\mathcal{F}(\boldsymbol{x}_i) = g(\boldsymbol{x}_i, \boldsymbol{z})$. For a given context set $\mathrm{C} = \{(\boldsymbol{x}_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}})\}$ and target set $\boldsymbol{x}_{1:n}$, $\boldsymbol{y}_{1:n}$, the generative process is given by:

$$p(\boldsymbol{y}_{1:n}, \boldsymbol{z} | \boldsymbol{x}_{1:n}, \mathrm{C}) = p(\boldsymbol{z}|\mathrm{C}) \prod_{i=1}^{n} \mathcal{N}(\boldsymbol{y}_i | g(\boldsymbol{x}_i, \boldsymbol{z}), \sigma^2), \tag{2}$$

where $p(\boldsymbol{z})$ is chosen to be a multivariate standard normal distribution and $\boldsymbol{y}_{1:n}$ is a shorthand for the sequence $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$. The model can be trained using an amortised variational inference procedure that naturally gives rise to a *permutation-invariant encoder* $q_\theta(\boldsymbol{z}|\mathrm{C})$, which stores the information about the context points. Conditioned on this information, the *decoder* $g(\boldsymbol{x}, \boldsymbol{z})$ can make predictions at any input location $\boldsymbol{x}$. We note that while the domain $\mathcal{X}$ of the random function F is arbitrary, in this work we are interested only in stochastic functions with domain on the real line (time-series). Therefore, from here our notation will reflect that, using t as the input instead of $\boldsymbol{x}$. The output $\boldsymbol{y}$ remains the same.

3 NEURAL ODE PROCESSES

Model Overview We introduce Neural ODE Processes (NDPs), a class of dynamics-based models that learn to approximate random functions defined over time. To that end, we consider an NP whose context Z is used to determine a distribution over ODEs. Concretely, Z gives a distribution over the initial position (and optionally – the initial velocity) and, at the same time, stochastically conditions its instantaneous velocity. The positions given by the ODE trajectories at any time t_i^{T} are then decoded to give the predictions at t_i^{T}. In what follows, we offer a detailed description of each component of the model. A schematic of the model can be seen in Figure 1.

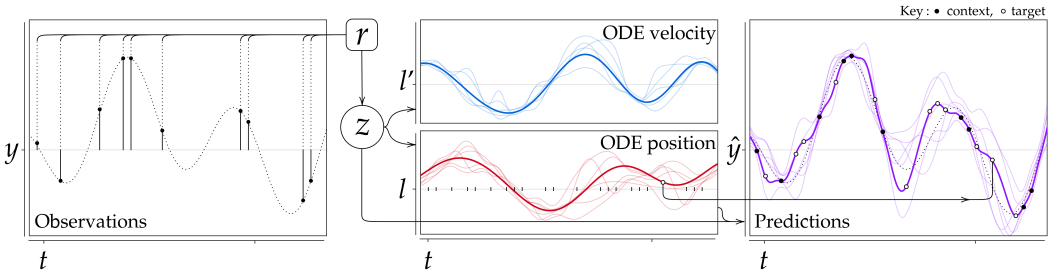

Figure 1: Schematic diagram of Neural ODE Processes. *Left:* Observations from a time series, the context set •, are encoded and aggregated to form $\boldsymbol{r}$ which parametrises the global latent variable $\mathbf{z}$. *Middle:* Samples from the global latent variable, $\boldsymbol{z}$, are used to initialise and condition the latent Neural ODE, with each sample producing a plausible, coherent trajectory. *Right:* Predictions at a target time, t_i^{T}, are made by decoding the state of the ODE, $\boldsymbol{l}(t_i^{\mathrm{T}})$ together with t_i^{T} and, optionally, the sample $\boldsymbol{z}$ used by that trajectory. An example is represented by the ∘ connected from the ODE position plot to the Predictions plot. *Middle, right:* dark lines relate to the same sample, fainter lines to other samples.

3.1 Generative Process

We first describe the generative process behind NDPs. A graphical model perspective of this process is also included in Figure 2.

Encoder and Aggregator Consider a given context set $\mathrm{C} = \{(t_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}})\}_{i \in I}$ of points coming from a time-series. As in NPs, we are first interested in producing a distribution over functions induced by the conditional prior over $Z \sim q(\boldsymbol{z}|\mathrm{C})$. To represent this distribution, the NDP encoder produces a representation $\boldsymbol{r}_i = f_e((t_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}}))$ for each context pair $(t_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}})$. The function f_e is parametrised as a neural network, fully connected or convolutional, depending on the nature of $\boldsymbol{y}$. An aggregator combines all the representations $\boldsymbol{r}_i$ to form a global representation, $\boldsymbol{r}$, that parametrises the distribution of the global latent context, $Z \sim q(\boldsymbol{z}|\mathrm{C}) = \mathcal{N}\big(\boldsymbol{z}|\mu_{\boldsymbol{z}}(\boldsymbol{r}), \mathrm{diag}(\sigma_{\boldsymbol{z}}(\boldsymbol{r}))\big)$. As the aggregator must preserve order invariance, we choose to take the element-wise mean.

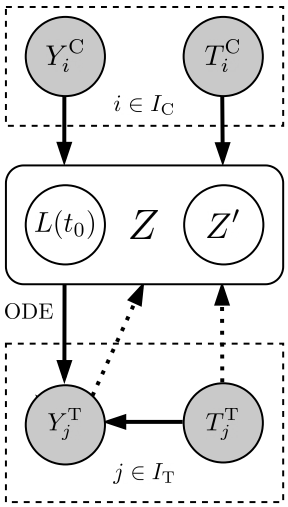

Figure 2: Graphical model of NDPs. The dark nodes denote observed random variables, while the light nodes denote hidden random variables. I_{C} and I_{T} represent the indexing sets for the context and target points, respectively. Full arrows show the generative process. Dotted arrows indicate inference.

Latent ODE Unlike NPs, which directly decode Z to obtain a distribution over functions, we are interested in capturing the dynamics that govern the time-series and exploiting the temporal nature of the data. To that end, we allow the latent context to evolve according to a Neural ODE (Chen et al., 2018) controlled by Z.

We begin by splitting the random context $Z \sim p(\boldsymbol{z}|\mathrm{C})$ into an initial latent position $L(t_0)$ and a sub-context Z' such that $Z = (L(t_0), Z')$. Splitting Z factorises the uncertainty in the underlying dynamics into an uncertainty over the initial condition (given by $L(t_0)$) and an uncertainty over the ODE derivative (given by Z').

Then, using the target times, $t_{1:n}^{\mathrm{T}} = (t_1^{\mathrm{T}}, ..., t_N^{\mathrm{T}})$, the latent state at a given time is found by evolving a Neural ODE:

$$\boldsymbol{l}(t_i^{\mathrm{T}}) = \boldsymbol{l}(t_0) + \int_{t_0}^{t_i^{\mathrm{T}}} f_\theta(\boldsymbol{l}(t), t, \boldsymbol{z}')dt, \tag{3}$$

where f_θ is a neural network that models the derivative of $\boldsymbol{l}$. As explained above, we allow $\boldsymbol{z}'$ to modulate the derivative of this ODE. Ultimately, for fixed initial conditions, this results in an uncertainty over the ODE trajectories.

Decoder To obtain a prediction at a time t_i^{T}, we decode the random state of the ODE at time t_i^{T}, given by $L(t_i^{\mathrm{T}})$. Assuming that the outputs are noisy, for a given sample $\boldsymbol{l}(t_i^{\mathrm{T}})$ from this stochastic state, the decoder g produces a distribution over $Y_{t_i}^{\mathrm{T}} \sim p\big(\boldsymbol{y}_i^{\mathrm{T}} | g(\boldsymbol{l}(t_i^{\mathrm{T}}), t_i)\big)$ parametrised by the decoder output. Concretely, for regression tasks, we take the target output to be normally distributed with constant (or optionally learned) variance $Y_{t_i}^{\mathrm{T}} \sim \mathcal{N}\big(\boldsymbol{y}_i | g(\boldsymbol{l}(t_i), t_i), \sigma^2\big)$. When $Y_{t_i}^{\mathrm{T}}$ is a random vector formed of independent binary random variables (e.g. a black and white image), we use a Bernoulli distribution $Y_{t_i}^{\mathrm{T}} \sim \prod_{j=1}^{\dim(Y)} \text{Bernoulli}\big(g(\boldsymbol{l}(t_i), t_i)_j\big)$.

Putting everything together, for a set of observed context points C, the generative process of NDPs is given by:

$$p(\boldsymbol{y}_{1:n}, \boldsymbol{z} | t_{1:n}, \mathrm{C}) = p(\boldsymbol{z}|\mathrm{C}) \prod_{i=1}^{n} p\big(\boldsymbol{y}_i | g(\boldsymbol{l}(t_i), t_i)\big), \tag{4}$$

where we highlight once again that $\boldsymbol{l}(t_i)$ also implicitly depends on $\boldsymbol{z}$.

We remark that **NDPs generalise NPs defined over time**. If the latent NODE learns the trivial velocity $f_\theta(\boldsymbol{l}(t), t, \boldsymbol{z}') = 0$, the random state $L(t) = L(t_0)$ remains constant at all times t. In this case, the distribution over functions is directly determined by $Z \sim p(\boldsymbol{z}|\mathrm{C})$, as in a regular NP. For even greater flexibility, one could also supply the sub-context $\boldsymbol{z}'$ to the decoder $g(\boldsymbol{l}(t), t, \boldsymbol{z}')$. This shows that, in principle, NDPs are at least as expressive as NPs. Therefore, NDPs could be a sensible choice even for applications where the time-series are not completely determined by some underlying dynamics, but are also influenced by other generative factors.

3.2 Learning and Inference

Since the true posterior is intractable because of the highly non-linear generative process, the model is trained using an amortised variational inference procedure. The variational lower-bound on the likelihood of the target values $\boldsymbol{y}_{m+1:n}$ at $t_{m+1:n}$ given the known context $t_{1:m}, \boldsymbol{y}_{1:m}$ is as follows:

$$\log p(\boldsymbol{y}_{m+1:n} | t_{1:n}, \boldsymbol{y}_{1:m}) \geq \mathbb{E}_{q(z|t_{1:n}, \boldsymbol{y}_{1:n})} \left[\sum_{i=m+1}^{n} \log p(\boldsymbol{y}_i | \boldsymbol{z}, t_i) + \log \frac{q(\boldsymbol{z}|t_{1:m}, \boldsymbol{y}_{1:m})}{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})} \right], \tag{5}$$

where q gives variational posterior (the encoder described in Section 3.1). The full derivation can be found in Appendix A. We use the reparametrisation trick to backpropagate the gradients of this loss. During training, we sample random contexts of different sizes to allow the model to become sensitive to the size of the context and the location of its points. We train using mini-batches composed of multiple contexts. For that, we use an extended ODE that concatenates the independent ODE states of each sample in the batch and integrates over the union of all the times in the batch (Rubanova et al., 2019). Pseudo-code for this training procedure is also given in Appendix B.

3.3 Model Variations

Here we present the different ways to implement the model. The majority of the variation is in the architecture of the decoder. However, it is possible to vary the encoder such that $f_e((t_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}}))$ can be a multi-layer-perceptron, or additionally contain convolutions.

Neural ODE Process (NDP) In this setup the decoder is a linear layer, which takes a concatenation of the latent state and a function of the latent state, time and sub-context as input: $g(\boldsymbol{l}(t_i^{\mathrm{T}}), t_i^{\mathrm{T}}, \boldsymbol{z}') = \boldsymbol{W}(\boldsymbol{l}(t_i^{\mathrm{T}}) \parallel h(\boldsymbol{l}(t_i^{\mathrm{T}}), t_i^{\mathrm{T}}, \boldsymbol{z}')) + \boldsymbol{b}$, where h can be a multi-layer-perceptron or additionally contain convolutional layers. This type of model is particularly suitable for high-dimensional time-series where the dynamics are fundamentally latent. However, its additional representation power given by h make it a good default choice for most settings.

Second Order Neural ODE Process (ND2P) This variation has the same decoder architecture as NDP, however the latent ode evolves according to a second order ODE. The latent state, $\boldsymbol{l}$, is split in half into a "position", $\boldsymbol{l}_1$ and "velocity", $\boldsymbol{l}_2$, with $\dot{\boldsymbol{l}_1} = \boldsymbol{l}_2$ and $\dot{\boldsymbol{l}_2} = f_\theta(\boldsymbol{l}_1, \boldsymbol{l}_2, t, \boldsymbol{z}')$. This model is designed for time-series where the dynamics are second-order. This is often the case given the use of Newton's 2nd Law to describe physical laws (Çağatay Yıldız et al., 2019; Norcliffe et al., 2020).

NDP Latent-Only (NDP-L) The decoder is a simple linear layer of the latent state $g(\boldsymbol{l}(t_i^{\mathrm{T}})) = \boldsymbol{W}(\boldsymbol{l}(t_i^{\mathrm{T}})) + \boldsymbol{b}$. This is based on the assumption that the latent state, which evolves according to an ODE, contains all the required information without any non-linearities. This would be suitable for simple functions generated by ODEs, for example, sines and exponentials. This decoder implicitly contains the time and context, because the latent state was evolved according to an ODE that depended on time and context given by Equation 3.

ND2P Latent-Only (ND2P-L) This model combines the assumption of second-order underlying dynamics and the idea of the latent state containing all of the information. The decoder is a linear layer of the latent state as in NDP-L, whose phase space dynamics are constrained as in ND2P.

3.4 Neural ODE Processes as Stochastic Processes

The Kolmogorov Extension Theorem states that *exchangeability* and *consistency* are necessary and sufficient conditions for a collection of joint marginal distributions to define a stochastic process Øksendal (2003); Garnelo et al. (2018). We define these conditions and show that the NDP model satisfies them. The proofs can be found in Appendix C.

Definition 3.1 (Exchangeability)**.** Exchangeability refers to the invariance of the joint distribution $\rho_{t_{1:n}}(\boldsymbol{y}_{1:n})$ under permutations of $\boldsymbol{y}_{1:n}$. That is, for a permutation π of $\{1, 2, ..., n\}$, $\pi(t_{1:n}) = (t_{\pi(1)}, ..., t_{\pi(n)})$ and $\pi(\boldsymbol{y}_{1:n}) = (\boldsymbol{y}_{\pi(1)}, ..., \boldsymbol{y}_{\pi(n)})$, the joint probability distribution $\rho_{t_{1:n}}(\boldsymbol{y}_{1:n})$ is invariant if $\rho_{t_{1:n}}(\boldsymbol{y}_{1:n}) = \rho_{\pi(t_{1:n})}(\pi(\boldsymbol{y}_{1:n}))$.

Proposition 3.1. *NDPs satisfy the exchangeability condition.*

Definition 3.2 (Consistency)**.** Consistency says if a part of a sequence is marginalised out, then the joint probability distribution is the same as if it was only originally taken from the smaller sequence:

$$\rho_{t_{1:m}}(\boldsymbol{y}_{1:m}) = \int \rho_{t_{1:n}}(\boldsymbol{y}_{1:n}) d\boldsymbol{y}_{m+1:n}. \tag{6}$$

Proposition 3.2. *NDPs satisfy the consistency condition.*

It is important to note that the stochasticity comes from sampling the full context $\boldsymbol{z}$. There is no stochasticity within the ODE, such as Brownian motion, though stochastic ODEs have previously been explored (Liu et al., 2019; Li et al., 2020; Tzen & Raginsky, 2019; Jia & Benson, 2020). For any given context $\boldsymbol{z}$, both the latent state trajectory and the observation space trajectory are fully determined.

3.5 Running Time Complexity

For a model with n context points and m target points, an NP has running time complexity $O(n+m)$, since the model only has to encode each context point and decode each target point. However, a Neural ODE Process has added complexity due to the integration process. Firstly, the integration itself has runtime complexity $O(\text{NFE})$, where NFE is the number of function evaluations. In turn, the worst-case NFE depends on the minimum step size δ the ODE solver has to use and the maximum time we are interested in, which we denote by $t_{\max}$. Secondly, for settings where the target times are not already ordered, an additional $O\big(m \log(m)\big)$ term is added for sorting them. This ordering is required by the ODE solver.

Therefore, given that $m \geq n$ and assuming a constant $t_{\max}$ exists, the worst-case complexity of NDPs is $O\big(m \log(m)\big)$. For applications where the times are already sorted (e.g. real-time applications), the complexity falls back to the original $O\big(n + m\big)$. In either case, NDPs scale well with the size of the input. We note, however, that the integration steps $t_{\max}/\delta$ could result in a very large constant, hidden by the big-O notation. Nonetheless, modern ODE solvers use adaptive step sizes that adjust to the data that has been supplied and this is unlikely to be a problem in practice.

In our experiments, when sorting is used, we notice the NDP models are between 1 and 1.5 orders of magnitude slower to train than NPs in terms of wall-clock time. At the same time, this limitation of the method is traded-off by a significantly faster loss decay per epoch and superior final performance. We provide a table of time ratios from our 1D experiments, from section 4.1, in Appendix D.

4 EXPERIMENTS

To test the proposed advantages of NDPs we carried out various experiments on time series data. Additional details for every task considered can be found in Appendix G.

Model details For the low-dimensional experiments in Sections 4.1 and 4.2, we use an MLP architecture for the encoder and decoder. For the high-dimensional experiments in Section 4.3, we use a convolutional architecture for both. We train the models using RMSprop (Tieleman & Hinton, 2012) with learning rate 1×10^{-3}. Additional model details can be found in Appendix F.

4.1 ONE DIMENSIONAL REGRESSION

We begin with a set of 1D regression tasks of differing complexity—sine waves, exponentials, straight lines and damped oscillators—that can be described by ODEs. For each task, the functions are determined by a set of parameters (amplitude, shift, etc) with pre-defined ranges. To generate the distribution over functions, we sample these parameters from a uniform distribution over their respective ranges. We use 490 time-series for training and evaluate on 10 separate test time-series. Each series contains 100 points. We repeat this procedure across 5 different random seeds to compute the standard error. Additional details can be found in Appendix G.1.

First, we visualise in the top-left corner of Figure 3 how NPs and NDPs adapt on the sine task to incoming data points. When a single data-point has been supplied, NPs have wrongly collapsed the distribution over functions to a set of almost horizontal lines. On the other hand, NDPs correctly capture a wider range of possible trajectories. Even when a large number of points has been supplied, the NP posterior does not collapse on a perfect fit, whereas NDPs correctly capture the true sine curve. In the other plots of Figure 3, we visualise the test-set MSE as a function of the training epoch. It can be seen that NDPs train in fewer iterations to a lower test loss due to the additional encoded structure of the latent ode state. At the same time, NDPs achieve this despite having approximately 10% fewer parameters than NPs.

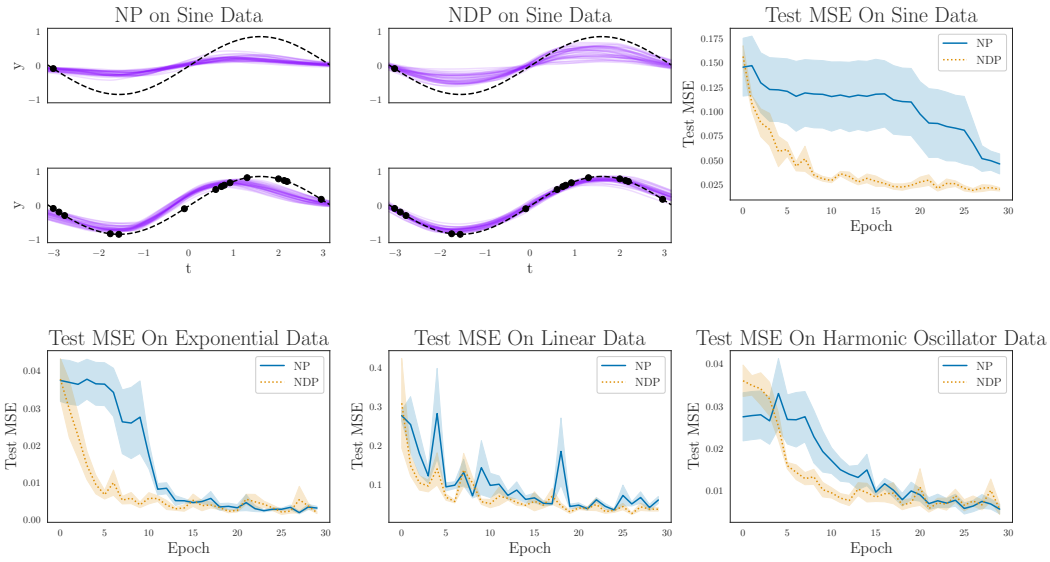

Figure 3: NPs and NDPs on the 1D datasets, we present the loss plots during training and example posteriors on the sine data. On the sine data, we see that NDPs are able to give a larger range of functions when only one point is available, and a smaller, more accurate range as more points in the time series are observed. On the right and below we see that quantitatively NDPs are also able to train to a lower loss in fewer epochs. This is expected for functions that are generated by ODEs. Both models were trained for 30 epochs.

Further to this, we carry out an ablation study, training all model variants on all the 1D datasets. Final test MSE losses are provided in Table 1 and the training plots are presented in Appendix G.1.

Table 1: Final MSE Loss on 1D regression tasks with standard error (lower is better). **Bold** indicates that the model performance is within error at the 95% confidence level, with underline indicating the best estimate for top-performer. NPs perform similarly to NDPs and their variants on the exponential and oscillator tasks, where y-values are close to zero, . For the sine and linear tasks, where y values vary significantly over the time range, NPs perform worse than NDPs and their variants.

	MSE $\times 10^{-2}$			
Model	**Sine**	**Linear**	**Exponential**	**Oscillators**
NP	5.93 ± 0.96	5.85 ± 0.70	$\mathbf{0.29 \pm 0.03}$	$\mathbf{0.64 \pm 0.06}$
NDP	$\mathbf{2.09 \pm 0.12}$	$\mathbf{3.76 \pm 0.32}$	$\mathbf{0.31 \pm 0.08}$	$\mathbf{0.72 \pm 0.08}$
ND2P	2.75 ± 0.19	$\mathbf{4.37 \pm 1.14}$	$\mathbf{0.25 \pm 0.04}$	$\mathbf{0.55 \pm 0.03}$
NDP-L	$\mathbf{2.51 \pm 0.24}$	4.77 ± 0.67	0.40 ± 0.04	0.72 ± 0.04
ND2P-L	$\mathbf{2.64 \pm 0.30}$	$\mathbf{3.16 \pm 0.46}$	0.39 ± 0.05	0.66 ± 0.03

We find that NDPs either strongly outperform NPs (sine, linear), or their standard errors overlap (exponential, oscillators). For the exponential and harmonic oscillator tasks, where the models perform similarly, many points are close to zero in each example and as such it is possible to achieve a low MSE score by producing outputs that are also around zero. In contrast, the sine and linear datasets have a significant variation in the y-values over the range, and we observe that NPs perform considerably worse than the NDP models on these tasks.

The difference between the NDP and the best of the model variants is not significant across the set of tasks. As such, we continue to use only NDPs for the remainder of the paper as this is the least constrained model variant: they have unrestricted latent phase-space dynamics, unlike the second-order counterparts, and a more expressive decoder architecture, unlike the latent-only variants. In addition, NDPs train in a faster wall clock time than the other variants, as shown in Appendix D.

4.2 Predator-Prey Dynamics

The Lotka-Volterra Equations are used to model the dynamics of a two species system, where one species predates on the other. The populations of the prey, u, and the predator, v, are given by the differential equations:

$$\frac{du}{dt} = \alpha u - \beta uv, \qquad \frac{dv}{dt} = \delta uv - \gamma v. \tag{7}$$

for positive real parameters, $\alpha, \beta, \delta, \gamma$. Intuitively, when prey is plentiful, the predator population increases ($+\delta uv$), and when there are many predators, the prey population falls ($-\beta uv$). The populations exhibit periodic behaviour, with the phase-space orbit determined by the conserved quantity $V = \delta u - \gamma \ln(u) + \beta v - \alpha \ln(v)$. Thus for any predator-prey system there exists a range of stable functions describing the dynamics, with any particular realisation being determined by the initial conditions, (u_0, v_0). We consider the system $(\alpha, \beta, \gamma, \delta) = (2/3, 4/3, 1, 1)$.

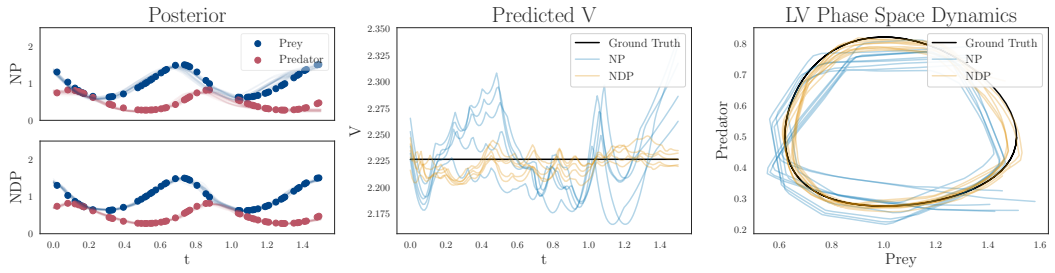

Figure 4: NPs and NDPs on the Lotka-Volterra task. *Left:* NPs are less able to model the dynamics, diverging from the ground truth even in regions with frequent context samples, whereas the variation in the NDP is more appropriately concentrated in the less sampled regions. *Middle:* Plotting the theoretically conserved quantity V better exposes how the models deviate from the ground truth. *Right:* In phase space (u, v) the NDP is more clearly seen to better track the ground truth.

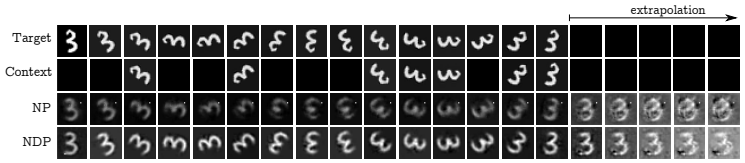

Figure 5: Rotating MNIST dataset. *First row:* The ground truth time-series. *Second row:* Seven context points supplied to the model. *Third row:* NP predictions. *Fourth row:* NDP predictions. NDPs perform better at interpolation, but neither model performs well at extrapolation.

Table 2: MSE on the Rotating MNIST test set.

Model	MSE $\times 10^{-2}$
ODE^{2}VAE-KL	**1.94** ± **0.03**
Conv-NP (1st)	3.18 ± 0.56
Conv-NDP (1st)	**1.94** ± **0.40**
Conv-NP (7)	2.80 ± 0.42
Conv-NDP (7)	**1.63** ± **0.49**

We generate sample time-series from the Lotka Volterra system by considering different starting configurations; $(u_0, v_0) = (2E, E)$, where E is sampled from a uniform distribution in the range (0.25, 1.0). The training set consists of 40 such samples, with a further 10 samples forming the test set. As before, each time series consists of 100 time samples and we evaluate across 5 different random seeds to obtain a standard error.

We find that NDPs are able to train in fewer epochs to a lower loss (Appendix G.2). We record final test MSEs ($\times 10^{-2}$) at 44 ± 4 for the NPs and 15 ± 2 for the NDPs. As in the 1D tasks, NDPs perform better despite having a representation $\boldsymbol{r}$ and context $\boldsymbol{z}$ with lower dimensionality, leading to 10% fewer parameters than NPs. Figure 4 presents these advantages for a single time series.

4.3 Rotating MNIST

We consider the rotating MNIST digits task (Casale et al., 2018; Çağatay Yıldız et al., 2019). Following Çağatay Yıldız et al. (2019), we remove the digit corresponding to the 4th rotation from all time-series in the dataset and use it as a test frame, in order to compare directly with their method. Additionally, we remove four random rotation angles from each sequence in the dataset to simulate irregularly sampled data. At test time, we report the MSE for the 4th rotation angle.

We perform two types of evaluation. We evaluate against NPs in predicting the 4th digit given a random context set C, as in previous tasks. To compare with ODE^{2}VAE (Çağatay Yıldız et al., 2019), we consider the restricted setting where the context set supplied at testing time $\mathrm{C} = \{(\boldsymbol{y}(t_0), t_0)\}$ is formed of the single image from time t_0. The latter setup is required as the ODE^{2}VAE is designed to evolve a latent Bayesian Neural ODE whose distribution over the initial position is given by $\boldsymbol{y}(t_0)$ only. In contrast, NDPs can handle arbitrary contexts. We delay a detailed analysis of the differences between NDPs and ODE^{2}VAE to Section 5.

The top part of Table 2 gives the results for the restricted setting. As expected, NDPs perform better than NPs. At the same time, NDPs have a competitive MSE to ODE^{2}VAE, despite having the disadvantage of not being especially designed or trained for operating in this setting. The second part of Table 2 gives the MSE when a random context of size seven was used. Once again, NDPs show superior performance. We also illustrate this advantage qualitatively in Figure 5. In Appendix G.4 we show that the model is also able to capture diverse stylistic aberrations.

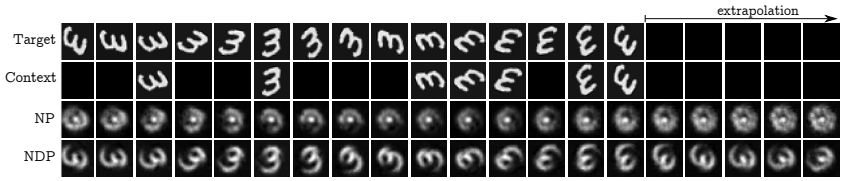

Figure 6: Variable angular velocity and angular shift rotating MNIST extrapolation. NDP is able to extrapolate beyond the training data range whereas NP cannot. The top row shows the targets (ground truth).

Additionally, we introduce a more challenging version of the Rotating MNIST dataset. Samples in the original task start upright and rotate once over 16 frames ($= 360°s^{-1}$) (i.e. constant angular velocity, zero angular shift). In our adaptation, the angular velocity varies between samples in the range $(360° \pm 90°)s^{-1}$ and each sample starts at a random initial rotation. As shown in Figure 6,

NDPs are able to extrapolate on the variable velocity MNIST dataset, although such extrapolation is not evident with the standard rotating MNIST dataset. We hypothesise that this capability is due to the different angular velocities in the dataset inducing the model to unlearn the relatively trivial interpolation between frames in favour of learning the underlying dynamics. In contrast, NPs struggle on this much more challenging domain and are unable to recover the stylistic aberrations of the digits.

5 Discussion and Related Work

We now consider two perspectives on how Neural ODE Processes relate to existing work and discuss the model in these contexts.

NDPs as Neural Processes From the perspective of stochastic processes, NDPs can be seen as a generalisation of NPs defined over time and, as such, existing improvements in this family are likely orthogonal to our own. For instance, following the work of Kim et al. (2019), we would expect adding an attention mechanism to NDPs to reduce uncertainty around context points. Additionally, the intrinsic sequential nature of time could be further exploited to model a dynamically changing sequence of NDPs as in Sequential NPs (Singh et al., 2019). For application domains where the observations evolve on a graph structure, such as traffic networks, relational information could be exploited with message passing operation as in MPNPs (Day et al., 2020).

NDPs as Neural ODEs From a dynamics perspective, NDPs can be thought of as an amortised Bayesian Neural ODE. In this sense, ODE^{2}VAE (Çağatay Yıldız et al., 2019) is the model that is most closely related to our method. While there are many common ideas between the two, significant differences exist. Firstly, NDPs do not use an explicit Bayesian Neural Network but are linked to them through the theoretical connections inherited from NPs (Garnelo et al., 2018). NDPs handle the uncertainty in the velocity (and acceleration) through Z', whereas ODE^{2}VAE uses a distribution over the NODE's weights. Secondly, NDPs stochastically condition the latent NODE's output (the particle's velocity) and initial position on an arbitrary context set of variable size. In contrast, ODE^{2}VAE conditions only the initial position and initial velocity on the first element and the first M elements in the sequence, respectively. Therefore, our model can dynamically adapt the dynamics to *any* observed time points. From that point of view, our model also runs parallel to other attempts of making Neural ODEs capable to dynamically adapt to irregularly sampled data points (Kidger et al., 2020).

6 Conclusion

We introduce Neural ODE Processes (NDPs), a new class of stochastic processes suitable for modelling data-adaptive stochastic dynamics. NDPs tackle the two main problems faced by Neural ODEs applied to dynamics-governed time series: adaptability to incoming data points and uncertainty in the underlying dynamics when the data is sparse and, potentially, irregularly sampled. To do so, NDPs include a probabilistic ODE as an additional encoded structure, thereby incorporating the inductive bias that the time-series is the direct or latent manifestation of an underlying ODE. Furthermore, NDPs maintain the scalability of NPs to very large inputs. We evaluate our model on synthetic 1D and 2D data, as well as higher-dimensional problems such as rotating MNIST digits. Our method exhibits superior training performance when compared with NPs, yielding a lower loss in fewer iterations. Whether or not the underlying ODE of the data is latent, we find that where there is a fundamental ODE governing the dynamics, NDPs perform well.

REFERENCES

Francesco Paolo Casale, Adrian V Dalca, Luca Saglietti, Jennifer Listgarten, and Nicolo Fusi. Gaussian Process Prior Variational Autoencoders. *arXiv e-prints*, art. arXiv:1810.11738, October 2018.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

Ben Day, Cătălina Cangea, Arian R. Jamasb, and Pietro Liò. Message passing neural processes, 2020.

Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural odes. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 32*, pp. 3140–3150. Curran Associates, Inc., 2019. URL `http://papers.nips.cc/paper/8577-augmented-neural-odes.pdf`.

Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J. Rezende, S. M. Ali Eslami, and Yee Whye Teh. Neural processes, 2018.

Jonathan Gordon, Wessel P. Bruinsma, Andrew Y. K. Foong, James Requeima, Yann Dubois, and Richard E. Turner. Convolutional conditional neural processes, 2019.

Junteng Jia and Austin R. Benson. Neural jump stochastic differential equations, 2020.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. *arXiv preprint arXiv:2005.08926*, 2020.

Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes, 2019.

Xuechen Li, Ting-Kam Leonard Wong, Ricky T. Q. Chen, and David Duvenaud. Scalable gradients for stochastic differential equations, 2020.

Xuanqing Liu, Tesi Xiao, Si Si, Qin Cao, Sanjiv Kumar, and Cho-Jui Hsieh. Neural sde: Stabilizing neural ode networks with stochastic noise, 2019.

Stefano Massaroli, Michael Poli, Jinkyoo Park, Atsushi Yamashita, and Hajime Asama. Dissecting neural odes, 2020.

James Morrill, Patrick Kidger, Cristopher Salvi, James Foster, and Terry Lyons. Neural cdes for long time series via the log-ode method, 2020.

Alexander Norcliffe, Cristian Bodnar, Ben Day, Nikola Simidjievski, and Pietro Liò. On second order behaviour in augmented neural odes, 2020.

Bernt Øksendal. Stochastic differential equations. In *Stochastic differential equations*, pp. 65–84. Springer, 2003.

Yulia Rubanova, Ricky T. Q. Chen, and David Duvenaud. Latent odes for irregularly-sampled time series, 2019.

Gautam Singh, Jaesik Yoon, Youngsung Son, and Sungjin Ahn. Sequential neural processes. In *Advances in Neural Information Processing Systems*, pp. 10254–10264, 2019.

T. Tieleman and G. Hinton. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

Belinda Tzen and Maxim Raginsky. Neural stochastic differential equations: Deep latent gaussian models in the diffusion limit, 2019.

Ben H Williams, Marc Toussaint, and Amos J Storkey. Extracting motion primitives from natural handwriting data. In *International Conference on Artificial Neural Networks*, pp. 634–643. Springer, 2006.

Çağatay Yıldız, Markus Heinonen, and Harri Lähdesmäki. Ode^{2}vae: Deep generative second order odes with bayesian neural networks, 2019.