Neural ODE Processes

Alexander Norcliffe[*] [†]
University College London
London, United Kingdom
ucabino@ucl.ac.uk

Cristian Bodnar[*]
University of Cambridge
Cambridge, United Kingdom
cb2015@cam.ac.uk

Ben Day[*]
University of Cambridge
Cambridge, United Kingdom
bjd39@cam.ac.uk

Jacob Moss[*]
University of Cambridge
Cambridge, United Kingdom
jm2311@cam.ac.uk

Pietro Liò
University of Cambridge
Cambridge, United Kingdom
pl219@cam.ac.uk

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) [2], 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 [10, 15, 16, 18, 22].

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.

[*]Equal contribution.
[†]Work done as an AI Resident at the University of Cambridge.

Third Workshop on Machine Learning and the Physical Sciences (NeurIPS 2020), Vancouver, Canada.

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) [6, 7], and runs parallel to other attempts to incorporate application-specific inductive biases in this class of models such as Attentive NPs [11], ConvCNPs [8], and MPNPs [3]. 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 [1].

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. [7], 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 [5] with input layer augmentation [14]. The extra dimensions offer the model additional flexibility as well as the ability to learn higher-order dynamics [16].

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 $i \in I_{\mathrm{C}}$. 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.

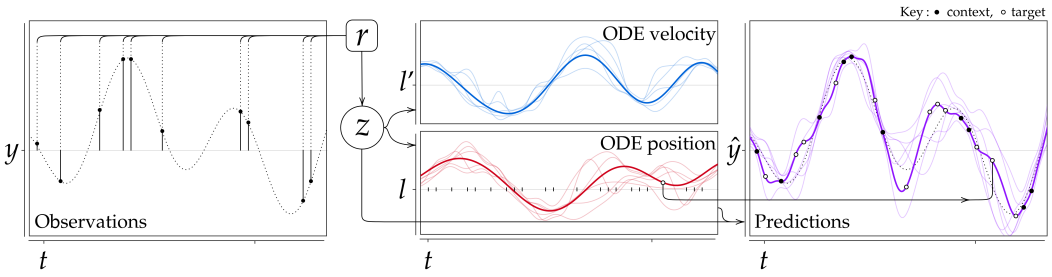

Figure 1: Schematic diagram of Neural ODE Processes. *Left:* Observations from a time series, the context set •, are encoded and aggregated to form $\boldsymbol{r}$ which parametrises the global latent variable $\mathbf{z}$. *Middle:* Samples from the global latent variable, $\boldsymbol{z}$, are used 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 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.

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 $\mathrm{C} = \{(t_i^{\mathrm{C}}, \boldsymbol{y}_i^{\mathrm{C}})\}_{i \in I_{\mathrm{C}}}$ 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.

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 [2] 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').

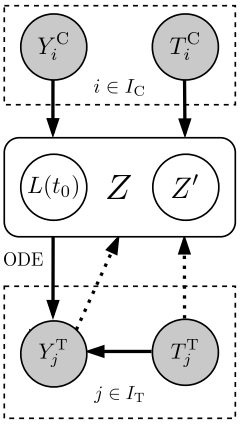

Figure 2: Graphical model of NDPs. The dark nodes denote observed random variables, while the light nodes denote hidden random variables. I_{C} and I_{T} represent the indexing sets for the context and target points, respectively. Full arrows show the generative process. Dotted arrows indicate inference.

Then, using the target times, $t^{\mathrm{T}}_{1:n} = (t^{\mathrm{T}}_1, ..., t^{\mathrm{T}}_N)$, the latent state at a given time is found by evolving a Neural ODE:

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

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

Putting everything together, for a set of observed context points C, the generative process of NDPs is given by $p(\boldsymbol{y}_{1:n}, \boldsymbol{z} | t_{1:n}, \mathrm{C}) = p(\boldsymbol{z}|\mathrm{C}) \prod_{i=1}^{n} p\big(\boldsymbol{y}_i | g(\boldsymbol{l}(t_i), t_i)\big)$, where we highlight once again that $\boldsymbol{l}(t_i)$ also implicitly depends on $\boldsymbol{z}$.

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:

$$\log p(\boldsymbol{y}_{m+1:n} | t_{1:n}, \boldsymbol{y}_{1:m}) \geq \mathbb{E}_{q(z|t_{1:n}, \boldsymbol{y}_{1:n})}\left[\sum_{i=m+1}^{n} \log p(\boldsymbol{y}_i | \boldsymbol{z}, t_i) + \log \frac{q(\boldsymbol{z}|t_{1:m}, \boldsymbol{y}_{1:m})}{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\right], \tag{4}$$

where q gives variational posterior (the encoder described in Section 3.1). The full derivation can be found in Appendix D. Pseudo-code for this training procedure is also given in Appendix E.

4 Experiments

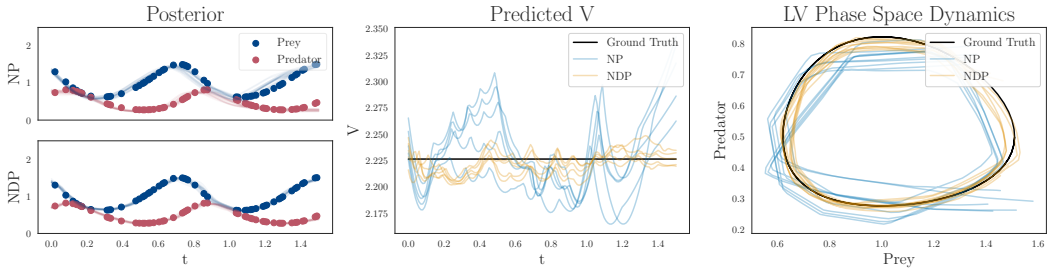

Figure 3: NPs and NDPs on the Lotka-Volterra task. *Left:* NPs are less able to model the dynamics, diverging from the ground truth even in regions with frequent context samples, whereas the variation in the NDP is more appropriately concentrated in the less sampled regions. *Middle:* Plotting the theoretically 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.

4.1 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:

$$\frac{du}{dt} = \alpha u - \beta uv, \qquad \frac{dv}{dt} = \delta uv - \gamma v. \tag{5}$$

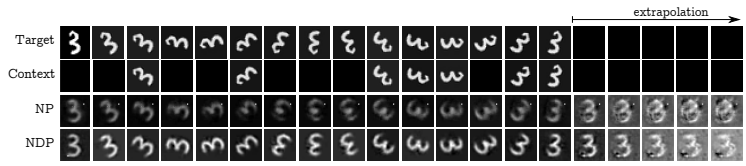

Figure 4: Rotating MNIST dataset. *First row:* The ground truth time-series. *Second row:* Seven context points supplied to the model. *Third row:* NP predictions. *Fourth row:* NDP predictions. NDPs perform better at interpolation, but neither model performs well at extrapolation.

Table 1: MSE on the Rotating MNIST test set.

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

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 $(\alpha, \beta, \gamma, \delta) = (2/3, 4/3, 1, 1)$.

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. NDPs perform better despite having a representation $\boldsymbol{r}$ and context $\boldsymbol{z}$ with lower dimensionality, leading to 10% fewer parameters than NPs. Figure 3 presents these advantages for a single time series.

4.2 Rotating MNIST

We consider the rotating MNIST digits task [1, 22]. Following Çağatay Yıldız et al. [22], 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 [22], 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 Appendix C.

The top part of Table 1 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 1 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 4. In Appendix G.4 we show that the model is also able to capture diverse stylistic aberrations.

5 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 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.

6 Broader Impact

Neural ODEs are a new class of models whose full range of real-world applications are yet to be determined. Our experiments have mostly focused on scientific applications such as learning the the evolution of populations in an ecological network. NDPs could also find applicability in civil engineering by providing assistance with predicting things like the vibrations of a material under stress. At the same time, we cannot neglect the possibility of these civil engineering uses cases to be transferred to a military one. However, our improvements remain incremental in that regard, and their effect should be covered by current institutional norms.

References

[1] 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.

[2] 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.

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

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

[5] 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`.

[6] Marta Garnelo, Dan Rosenbaum, Chris J. Maddison, Tiago Ramalho, David Saxton, Murray Shanahan, Yee Whye Teh, Danilo J. Rezende, and S. M. Ali Eslami. Conditional neural processes, 2018.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[21] 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.

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

A 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 [7, 17]. We define these conditions and show that the NDP model satisfies them.

Before giving the proofs, we state the following important Lemma.

Lemma A.1. *As in NPs, the decoder output $g(\boldsymbol{l}(t), t)$ can be seen as a function $\mathcal{F}(t)$ for a given fixed $\boldsymbol{z}$.*

Proof. This follows directly from the fact that $\boldsymbol{l}(t) = \boldsymbol{l}(t_0) + \int_{t_0}^{\mathrm{T}} f_\theta(\boldsymbol{l}(t), t, \boldsymbol{z}')dt$ can be seen as a function of t. This is because $\boldsymbol{z} = [\boldsymbol{l}(t_0), \boldsymbol{z}']$ and the integration process is deterministic for a given $\boldsymbol{z}$ (i.e. for fixed initial conditions and velocity). □

Definition A.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 A.1. *NDPs satisfy the exchangeability condition.*

Proof. This follows directly from Lemma A.1, since any permutation on $t_{1:n}$ would automatically act on $\mathcal{F}_{1:n}$ and consequently on $p(\boldsymbol{y}_{1:n}, \boldsymbol{z}|t_{1:n})$, for any given $\boldsymbol{z}$. □

Definition A.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 A.2. *NDPs satisfy the consistency condition.*

Proof. Based on Lemma A.1 we can write the joint distribution (similarly to a regular NP) as follows:

$$\rho_{t_{1:n}}(\boldsymbol{y}_{1:n}) = \int p(\mathcal{F}) \prod_{i=1}^{n} \mathcal{N}(\boldsymbol{y}_i|\mathcal{F}(t_i), \sigma^2) d\mathcal{F}. \tag{7}$$

Because the density of any $\boldsymbol{y}_i$ depends only on the corresponding t_i, integrating out any subset of $\boldsymbol{y}_{1:n}$ gives the joint distribution of the remaining random variables in the sequence. Thus, consistency is guaranteed. □

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 [9, 12, 13, 20]. For any given context $\boldsymbol{z}$, both the latent state trajectory and the observation space trajectory are fully determined.

B 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 NDPs are between 1 and 1.5 orders of magnitude slower to train than NPs in terms of wall-clock time as seen below.

B.1 Wall Clock Training Times

To explore the additional term in the runtime given in Section B, we record the wall clock time for each model to train for 30 epochs on 1D synthetic datasets, over 5 seeds. The experiments were run on an *Nvidia Titan XP*. The results can be seen in Table 2.

Table 2: Table of ratios, of Neural ODE Process and Neural Process training times on different 1D synthetic datasets.

Time Ratios	**Sine**	**Exponential**	**Linear**	**Oscillators**
NDP/NP	22.1 ± 0.9	23.6 ± 0.9	10.9 ± 1.4	22.2 ± 2.3
NP Training Time /s	22.4 ± 0.2	45.5 ± 0.3	100.9 ± 0.3	23.2 ± 0.4

C Discussion and Related Work

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. [11], 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 [19]. 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 [3].

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 [22] 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 [7]. 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 [10].

D ELBO Derivation

As noted in Remark A.1, the joint probability $p(\boldsymbol{y}, \boldsymbol{z}|t) = p(\boldsymbol{z})\mathcal{N}(\boldsymbol{y}_i|g(\boldsymbol{l}(t), t, \boldsymbol{z}'), \sigma^2)$ can still be seen as a function that depends only on t, since the ODE integration process is deterministic for a given $\boldsymbol{z}$. Therefore, the ELBO derivation proceeds as usual [7]. First, we derive the ELBO for

$\log p(\boldsymbol{y}_{1:n}|t_{1:n})$.

$$\log p(\boldsymbol{y}_{1:n}|t_{1:n}) = D_{\mathrm{KL}}\big(q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})\|p(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})\big) + \mathcal{L}_{\text{ELBO}} \tag{8}$$

$$\geq \mathcal{L}_{\text{ELBO}} = \mathbb{E}_{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\big[-\log q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n}) + \log p(\boldsymbol{y}_{1:n}, \boldsymbol{z}|t_{1:n})\big] \tag{9}$$

$$= -\mathbb{E}_{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})} \log q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n}) \tag{10}$$

$$+ \mathbb{E}_{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\big[\log p(\boldsymbol{z}) + \log p(\boldsymbol{y}_{1:n}|t_{1:n}, \boldsymbol{z})\big] \tag{11}$$

$$= \mathbb{E}_{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\left[\sum_{i=1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{z}, t_i) + \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\right] \tag{12}$$

Since at training time we are interested in the log evidence given some context, we can split this sequence into the context $t_{1:m}, \boldsymbol{y}_{1:m}$ and the target $t_{m+1:n}, \boldsymbol{y}_{m+1:n}$. Then, we want to maximise $\log p(\boldsymbol{y}_{m+1:n}|t_{1:n}, \boldsymbol{y}_{1:m})$. Using the derivation above, we obtain a similar lower-bound, but with a new prior $p(z|t_{1:m}, \boldsymbol{y}_{1:m})$, updated to reflect the additional information supplied by the context.

$$\log p(\boldsymbol{y}_{m+1:n}|t_{1:n}, \boldsymbol{y}_{1:m}) \geq \mathbb{E}_{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\left[\sum_{i=m+1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{z}, t_i) + \log \frac{p(\boldsymbol{z}|t_{1:m}, \boldsymbol{y}_{1:m})}{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\right] \tag{13}$$

If we approximate the true $p(z|t_{1:m}, \boldsymbol{y}_{1:m})$ with the variational posterior, this takes the final form

$$\log p(\boldsymbol{y}_{m+1:n}|t_{1:n}, \boldsymbol{y}_{1:m}) \geq \mathbb{E}_{q(z|t_{1:n}, \boldsymbol{y}_{1:n})}\left[\sum_{i=m+1}^{n} \log p(\boldsymbol{y}_i|\boldsymbol{z}, t_i) + \log \frac{q(\boldsymbol{z}|t_{1:m}, \boldsymbol{y}_{1:m})}{q(\boldsymbol{z}|t_{1:n}, \boldsymbol{y}_{1:n})}\right] \tag{14}$$

E Learning and Inference Procedure

We include below the pseudocode for training NDPs. For clarity of exposition, we give code for a single time-series. However, in practice, we batch all the operations in lines $6 - 15$.

Algorithm 1: Learning and Inference in Neural ODE Processes

Input : A dataset of time-series $\{\boldsymbol{X}_k\}$, $k \leq K$, where K is the total number of time-series

1 Initialise NDP model with parameters θ
2 Let m be the number of context points and n the number of extra target points
3 **for** $i \leftarrow 0$ **to** *training_steps* **do**
4 Sample m from U[1, $max\_context\_points$]
5 Sample n from U[1, $max\_extra\_target\_points$]
6 Uniformly sample a time-series $\boldsymbol{X}_k$
7 Uniformly sample from $\boldsymbol{X}_k$ the target points $\mathrm{T} = (\mathrm{T}_t, \mathrm{T}_{\boldsymbol{y}})$, where T_t is the time batch with shape $(m + n, 1)$ and $\mathrm{T}_{\boldsymbol{y}}$ is the corresponding outputs batch with shape $(m + n, \dim(\boldsymbol{y}))$
8 Extract (unordered) the context set $\mathrm{C} = (\mathrm{T}_t[0 : m], \mathrm{T}_{\boldsymbol{y}}[0 : m])$
9 Compute $q(\boldsymbol{z}|\mathrm{C})$ using the variational encoder
10 Compute $q(\boldsymbol{z}|\mathrm{T})$ using the variational encoder
  `// During training, we sample from` $q(\boldsymbol{z}|\mathrm{T})$
11 Sample $\boldsymbol{z}$ from $q(\boldsymbol{z}|\mathrm{T})$ and split $\boldsymbol{z} = [\boldsymbol{l}(t_0), \boldsymbol{z}']$
12 Integrate to compute $\boldsymbol{l}(t)$ as in Equation 3 for all times $t \in T_t$
13 **foreach** *time* $t \in T_t$ **do**
14 Use decoder to compute $p(\boldsymbol{y}(t)|g(\boldsymbol{l}(t)), t)$
15 Compute loss $\mathcal{L}_{\text{ELBO}}$ based on Equation 4
16 $\theta \leftarrow \theta - \alpha\nabla_\theta\mathcal{L}_{\text{ELBO}}$

It is worth highlighting that during training we sample $\boldsymbol{z}$ from the target posterior, rather than the context posterior. This is because the expectation in Equation 4 is computed over the target posterior. In contrast, at inference time we sample from the context posterior.

F Architectural Details

For the experiments with low dimensionality (1D, 2D), the architectural details are as follows:

- **Encoder**: $[t_i, y_i] \to \boldsymbol{r}_i$: Multilayer Perceptron, 2 hidden layers, ReLU activations.
- **Aggregator**: $\boldsymbol{r}_{1:n} \to \boldsymbol{r}$: Taking the mean.
- **Representation to Hidden**: $\boldsymbol{r} \to \boldsymbol{h}$: One linear layer followed by ReLU.
- **Hidden to Context Mean**: $\boldsymbol{h} \to \mu_{\boldsymbol{z}}$: One linear layer.
- **Hidden to Context Variance**: $\boldsymbol{h} \to \sigma_{\boldsymbol{z}}$: One linear layer, followed by sigmoid, multiplied by 0.9 add 0.1, i.e. $\sigma_{\boldsymbol{z}} = 0.1 + 0.9 \times \text{sigmoid}(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{b})$.
- **ODE Layers**: $[\boldsymbol{l}, \boldsymbol{z}', t] \to \dot{\boldsymbol{l}}$: Multilayer Perceptron, two hidden layers, $\tanh$ activations.
- **Decoder**: $h(\boldsymbol{l}(t_i^{\mathrm{T}}), \boldsymbol{z}', t_i^{\mathrm{T}}) \to y_i^{\mathrm{T}}$, Multilayer Perceptron with two hidden layers and ReLU activations.

For the high-dimensional experiments (Rotating MNIST).

- **Encoder**: $[t_i, y_i] \to \boldsymbol{r}_i$: Convolutional Neural Network, 4 layers with 16, 32, 64, 128 channels respectively and kernel size of 5, stride 2. ReLU activations. Batch normalisation.
- **Aggregator**: $\boldsymbol{r}_{1:n} \to \boldsymbol{r}$: Taking the mean.
- **Representation to Hidden**: $\boldsymbol{r} \to \boldsymbol{h}$: One linear layer followed by ReLU.
- **Hidden to Context Mean**: $\boldsymbol{h} \to \mu_{\boldsymbol{z}}$: One linear layer.
- **Hidden to Context Variance**: $\boldsymbol{h} \to \sigma_{\boldsymbol{z}}$: One linear layer, followed by sigmoid, multiplied by 0.9 add 0.1, i.e. $\sigma_{\boldsymbol{z}} = 0.1 + 0.9 \times \text{sigmoid}(\boldsymbol{W}\boldsymbol{h} + \boldsymbol{b})$.
- **ODE Layers**: $[\boldsymbol{l}, \boldsymbol{z}', t] \to \dot{\boldsymbol{l}}$: Multilayer Perceptron, two hidden layers, $\tanh$ activations.
- **Decoder**: $h(\boldsymbol{l}(t_i^{\mathrm{T}}), \boldsymbol{z}', t_i^{\mathrm{T}}) \to y_i^{\mathrm{T}}$: 1 linear layer followed by a 4 layer transposed Convolutional Neural Network with 32, 128, 64, 32 channels respectively. ReLU activations. Batch normalisation.

G Task Details and Additional Results

G.1 One Dimensional Regression

We carried out an ablation study over model variations on various 1D synthetic tasks—sines, exponentials, straight lines and harmonic oscillators. Each task is based on some function described by a set of parameters that are sampled over to produce a distribution over functions. In every case, the parameters are sampled from uniform distributions. A trajectory example is formed by sampling from the parameter distributions and then sampling from that function at evenly spaced timestamps, t, over a fixed range to produce 100 data points (t, y). We give the equations for these tasks in terms of their defining parameters and the ranges for these parameters in Table 3.

Task	Form	a	b	t	# train	# test
Sines	$y = a\sin(t - b)$	$(-1, 1)$	$(-1/2, 1/2)$	$(-\pi, \pi)$	490	10
Exponentials	$y = a/60 \times \exp(t - b)$	$(-1, 1)$	$(-1/2, 1/2)$	$(-1, 4)$	490	10
Straight lines	$y = at + b$	$(-1, 1)$	$(-1/2, 1/2)$	$(0, 5)$	490	10
Oscillators	$y = a\sin(t - b)\exp(-t/2)$	$(-1, 1)$	$(-1/2, 1/2)$	$(0, 5)$	490	10

Table 3: Task details for 1D regression. a and b are sampled uniformly at random from the given ranges. t is sampled at 100 regularly spaced intervals over the given range. 490 training examples and 10 test examples were used in every case.

To test after each epoch, 10 random context points are taken, and then the mean-squared error and negative log probability are calculated over all the points (not just a subset of the target points). Each model was trained 5 times on each dataset (with different weight initialisation). We used a batch size of 5, with context size ranging from 1 to 10, and the extra target size ranging from 0 to 5.[3] The results are presented in Figure 5. We see that NDPs reduce the loss in fewer iterations than NPs.

[3] As written in the problem statement in section 2, we make the context set a subset of the target set when training. So we define a context size range and an extra target size range for each task.

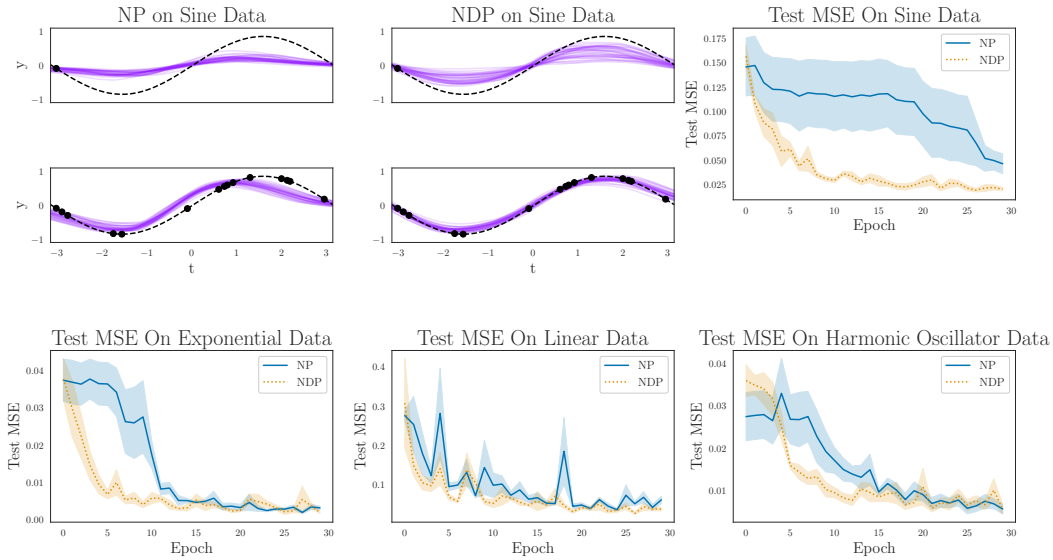

Figure 5: NPs and NDPs on the 1D datasets, we present the loss plots during training and example posteriors on the sine data. On the sine data, we see that NDPs are able to give a larger range of functions when only one point is available, and a smaller, more accurate range as more points in the time 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.

G.2 Lotka-Volterra System

To generate samples from the Lotka Volterra system, we sample different starting configurations, $(u_0, v_0) = (2E, E)$, where E is sampled from a uniform distribution in the range (0.25, 1.0). We then evolve the Lotka Volterra system

$$\frac{du}{dt} = \alpha u - \beta uv, \qquad \frac{dv}{dt} = \delta uv - \gamma v. \tag{15}$$

using $(\alpha, \beta, \gamma, \delta) = (2/3, 4/3, 1, 1)$. This is evolved from $t = 0$ to $t = 15$ and then the times are rescaled by dividing by 10.

The training for the Lotka-Volterra system can be seen in Figure 6. This was taken across 5 seeds, with a training set of 40 trajectories, 10 test trajectories and batch size 5. We use a context size ranging from 1 to 100, and extra target size ranging from 0 to 45. The test context size was fixed at 90 query times. NDP trains slightly faster with lower loss, as expected.

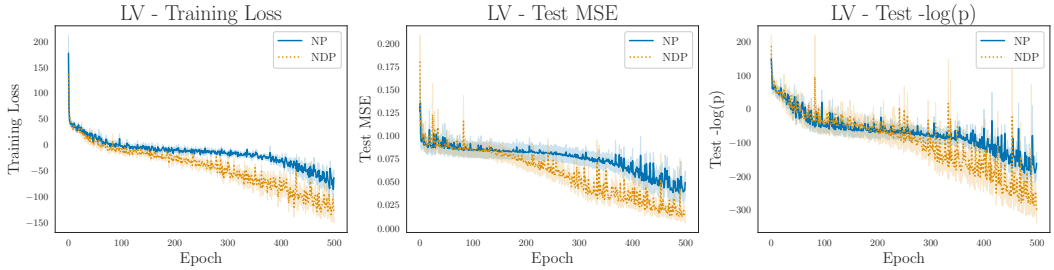

Figure 6: Training NP and NDP on the Lotka-Volterra equations. Due to the additional encoding structure of NDP, it can be seen that NDPs train in fewer iterations, to a lower loss than NPs.

G.3 Handwritten Characters

The `CharacterTrajectories` dataset consists of single-stroke handwritten digits recorded using an electronic tablet [4, 21]. The trajectories of the pen tip in two dimensions, (x, y), are of varying length, with a force cut-off used to determine the start and end of a stroke. We consider a reduced dataset, containing only letters that were written in a single stroke, this disregards letters such as "f", "i" and "t". Whilst it is not obvious that character trajectories should follow an ODE, the related Neural Controlled Differential Equation (NCDEs) model has been applied successfully to this task [10]. We train with a training set with 49600 examples, a test set with 400 examples and a batch size of 200. We use a context size ranging between 1 and 100, an extra target size ranging between 0 and 100 and a fixed test context size of 20. We visualise the training of the models in Figure 7 and the models plotting posteriors in Figure 8.

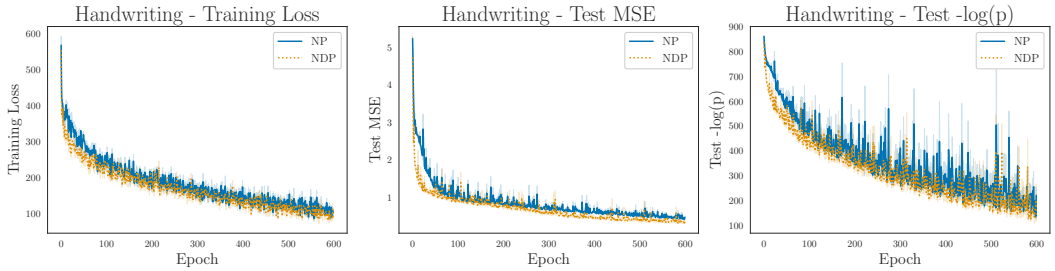

Figure 7: NPs and NDPs training on handwriting. NDPs perform slightly better, achieving a lower loss in fewer iterations. However this is a marginal improvement, and we believe it is down to significant diversity in the dataset, due to there being no fundamental differential equation for handwriting.

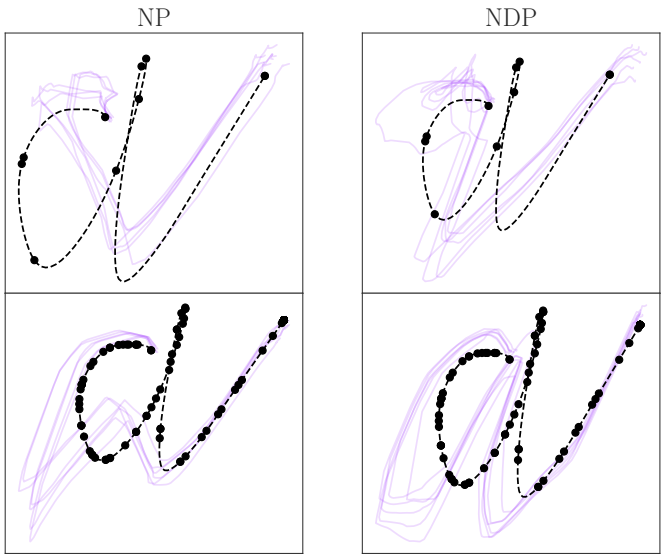

Figure 8: We test the models on drawing the letter "a" with varying numbers of context points. For a few context points, the trajectories are diverse and not entirely recognisable. For example, for NP, qualitatively the drawing looks more like a "v" than "a". And for NDP, the drawing looks more like a "q" than an "a". As more context points are observed, the trajectories become less diverse and start approaching an "a". We expect that with more training, and editing the hyperparameters, such as batch size, or the number of hidden layers this model would improve. Additionally, we observe that NDP qualitatively outperforms NP on a small number and a large number of context points.

We observe that NPs and NDPs are unable to successfully learn the time series as well as NCDEs. We record final test MSEs ($\times 10^{-1}$) at 4.6 ± 0.1 for NPs and a slightly lower 3.4 ± 0.1 for NDPs. We believe the reason is because handwritten digits do not follow an inherent ODE solution, especially given the diversity of handwriting styles for the same letter. We conjecture that Neural Controlled Differential Equations were able to perform well on this dataset due to the control process. Controlled ODEs follow the equation:

$$\boldsymbol{z}(T) = \boldsymbol{z}(t_0) + \int_{t_0}^{\mathrm{T}} f_\theta(\boldsymbol{z}(t), t) \frac{d\boldsymbol{X}(t)}{dt} dt, \qquad \boldsymbol{z}(t_0) = h_1(\boldsymbol{x}(t_0)), \qquad \hat{\boldsymbol{x}}(T) = h_2(\boldsymbol{z}(T)) \quad (16)$$

Where $\boldsymbol{X}(t)$ is the natural cubic spline through the observed points $\boldsymbol{x}(t)$. If the learnt f_θ is an identity operation, then the result returned will be the cubic spline through the observed points. Therefore, a controlled ODE can learn an identity with a small perturbation, which is easier to learn with the aid of a control process, rather than learning the entire ODE trajectory.

G.4 Rotating MNIST

In Figure 9, we plot the prediction of different styles at test-time. Observe that the model is able to detect different styles.

Figure 9: NDP is able to capture different styles in the rotating MNIST dataset.

G.5 Variable Rotating MNIST

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 10, 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.

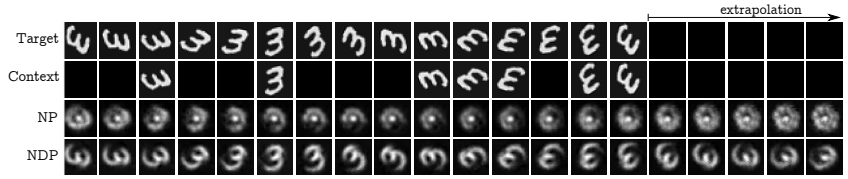

Figure 10: Variable angular velocity and angular shift rotating MNIST extrapolation. NDP is able to extrapolate beyond the training data range whereas NP cannot. The top row shows the targets (ground truth).