Proceedings of Machine Learning Research 255:1–20, 2024 ML-DE Workshop at ECAI 2024

Time and State Dependent Neural Delay Differential Equations

Thibault Monsel THIBAULT.MONSEL@INRIA.FR
LISN and INRIA, CNRS, Université Paris-Saclay, CNRS, 91405, Orsay, France

Onofrio Semeraro ONOFRIO.SEMERARO@UNIVERSITE-PARIS-SACLAY.FR
Lionel Mathelin LIONEL.MATHELIN@CNRS.FR
LISN, CNRS, Université Paris-Saclay, CNRS, 91405, Orsay, France

Guillaume Charpiat GUILLAUME.CHARPIAT@INRIA.FR
LISN, INRIA, Université Paris-Saclay, CNRS, 91405, Orsay, France

Editors: Cecília Coelho, Bernd Zimmering, M. Fernanda P. Costa, Luís L. Ferrás, Oliver Niggemann

Abstract

Discontinuities and delayed terms are encountered in the governing equations of a large class of problems ranging from physics and engineering to medicine and economics. These systems cannot be properly modelled and simulated with standard Ordinary Differential Equations (ODE), or data-driven approximations such as Neural Ordinary Differential Equations (NODE). To circumvent this issue, latent variables are typically introduced to solve the dynamics of the system in a higher dimensional space and obtain the solution as a projection to the original space. However, this solution lacks physical interpretability. In contrast, Delay Differential Equations (DDEs), and their data-driven approximated counterparts, naturally appear as good candidates to characterize such systems. In this work we revisit the recently proposed Neural DDE by introducing Neural State-Dependent DDE (SDDDE), a general and flexible framework that can model multiple and state- and time-dependent delays. We show that our method is competitive and outperforms other continuous-class models on a wide variety of delayed dynamical systems. Code is available at the repository here.

Keywords: Delay, Delay Differential Equations, Neural ODE/DDE, Physical Modelling, Dynamical Systems, Continuous-depth models, DDE solver

1. Introduction

In many applications, one assumes the time-dependent system under consideration satisfies a Markov property; that is, future states of the system are entirely defined from the current state and are independent of the past. In this case, the system is satisfactorily described by an ordinary or a partial differential equation. However, the property of Markovianity is often only a first approximation to the true situation and a more realistic model would include past states of the system. Describing such systems has fueled the extensive development of the theory of delay differential equations (DDE) (Minorsky, 1942; Myshkis, 1949; Hale, 1963). This development has given rise to many practical applications: in the modelling of molecular kinetics (Roussel, 1996) as well as for diffusion processes (Epstein, 1990), in physics for modeling semiconductor lasers (Vladimirov et al., 2004), in climate research for describing the El Ninõ current (Ghil et al., 2008; Keane et al., 2019), infectious diseases (Cooper et al., 2020) and tsunami forecasting (Wu et al., 2022), to list only a few.

At the same time, the blooming of machine learning in recent years boosted the development of new algorithms aimed at modelling and predicting the behaviour of dynamical systems governing phenomena commonly found in a wide variety of fields. Among these novel strategies, the introduction of Neural Ordinary Differential Equations (NODEs) (Chen et al., 2018) has contributed to further deepening the analysis of continuous dynamical systems modelling based on neural networks. NODEs are a family of neural networks that can be seen as the continuous extension of Residual Networks (He et al., 2016), where the dynamics of a vector $\mathbf{y}(t) \in \mathbb{R}^d$ at time t – hereafter often identified with the state of a physical system – is given by the parameterized network $\mathbf{f}_\theta$ and the system's initial condition $\mathbf{y}_0$:

$$\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}_\theta(t, \mathbf{y}(t)), \quad \mathbf{y}(0) = \mathbf{y}_0. \tag{1}$$

NODEs have been successfully applied to various tasks, such as normalizing flows (Kelly et al., 2020; Grathwohl et al., 2018), handling irregularly sampled time data (Rubanova et al., 2019; Kidger et al., 2020), and image segmentation (Pinckaers and Litjens, 2019).

Starting from this groundbreaking work, numerous extensions of the NODE framework enabled to widen the range of applications. Among them, Augmented NODEs (ANODEs) (Dupont et al., 2019) were able to alleviate NODEs' expressivity bottleneck by augmenting the dimension of the space allowing the model to learn more complex functions using simpler flows (Dupont et al., 2019). Let $\mathbf{a}(t) \in \mathbb{R}^p$ denotes a point in the augmented space, the ODE problem is formulated as

$$\frac{d}{dt}\begin{bmatrix}\mathbf{y}(t)\\ \mathbf{a}(t)\end{bmatrix} = \mathbf{f}_\theta\left(t, \begin{bmatrix}\mathbf{y}(t)\\ \mathbf{a}(t)\end{bmatrix}\right), \quad \begin{bmatrix}\mathbf{y}(0)\\ \mathbf{a}(0)\end{bmatrix} = \begin{bmatrix}\mathbf{y}_0\\ \mathbf{0}\end{bmatrix}. \tag{2}$$

By introducing this new variable $\mathbf{a}(t)$, ANODE overcomes the inability of NODE to represent particular classes of systems. However, this comes with the cost of augmenting the data into a higher dimensional space, hence losing interpretability. Among the alternative techniques proposed for circumventing the limitations rising from the modelling of non-Markovian systems, the Neural Laplace model (Holt et al., 2022) proposes a unified framework that solves differential equations (DE): it learns DE solutions in the Laplace domain. The Neural Laplace model cascades 3 steps: first, a network $\mathbf{h}_\gamma$ encodes the trajectory, then the so-called Laplace representation network $\mathbf{g}_\beta$ learns the dynamics in the Laplace domain to finally map it back to the temporal domain with an inverse Laplace transform (ILT). With the state $\mathbf{y}$ sampled T times at arbitrary time instants, $\mathbf{h}_\gamma$ gives a latent initial condition representation vector $\mathbf{p} \in \mathbb{R}^K$:

$$\mathbf{p} = \mathbf{h}_\gamma((\mathbf{y}(t_1), t_1), \ldots, (\mathbf{y}(t_T), t_T)), \tag{3}$$

that is fed to the network $\mathbf{g}_\beta$ to get the Laplace transform

$$\mathbf{F}(\mathbf{s}) = v\left(\mathbf{g}_\beta(\mathbf{p}, u(\mathbf{s}))\right), \tag{4}$$

with u a stereographic projector and v its inverse. Ultimately, an ILT step is applied to reconstruct state estimate $\widehat{\mathbf{y}}$ from the learnt $\mathbf{F}(\mathbf{s})$.

As an alternative to the aforementioned techniques, one may directly address the DDE problem by working within the framework of neural network-based DDEs. Despite the success of the NODEs philosophy, the extension to DDEs has barely been studied yet, possibly owing to the challenges of using general purpose DDE solvers. DDEs extend ODEs by incorporating additional terms into their vector fields, which are states delayed by a certain time τ. Recently, Zhu et al. (2021) introduced a neural network based DDE with one single constant delay:

$$\begin{aligned}&\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}_\theta(t, \mathbf{y}(t), \mathbf{y}(t-\tau)), \quad \tau \in \mathbb{R}^+ \\ &\mathbf{y}(t<0) = \boldsymbol{\phi}(t),\end{aligned} \tag{5}$$

where $\boldsymbol{\phi}(t)$ is the system's history function, τ a constant delay and $\mathbf{f}_\theta$ a parameterized network. This work was next extended in the Neural Piece-Wise Constant Delays Differential Equations (NPCDDEs) model in Zhu et al. (2022). Compared to NODE and its augmented counterpart, neural network-based DDEs do not require an augmentation to a higher dimensional space in order to be a universal approximator, thus preserving physical interpretability of the state vector and allowing the identification of the time delays. Nonetheless, the current variants of Neural DDE models only deals with a single constant delay or several piece-wise constant delays, thus lacking the generalization to arbitrary delays. Moreover, to the best of our knowledge, no machine learning library or open-sourced code exists to model not only these very specific types of DDEs but also any generic DDEs.

In this work, we introduce Neural State-Dependent DDE (SDDDE) model: an open-source, robust python DDE solver compatible with neural networks. Neural SDDDE is based on a general framework that pushes the envelope of Neural DDEs by handling DDEs with several delays in a more generic way. The implementation further encompasses general time- and state-dependent delay systems which extends the reach of Zhu et al. (2022).

In the remainder of the paper, we briefly introduce the framework in Sec. 2. Implementation and methodologies are further discussed in Sec. 3. Experiments and comparisons with the state-of-the-art techniques are detailed in Sec. 4, using as benchmark numerous time-delayed models of incremental complexity. Our model is shown below to compare favorably with the current models on DDEs systems. Conclusions and outlook finalize the article in Sec. 5.

2. Neural State-Dependant DDEs

In this section, we introduce the Neural State-Dependent Delay Differential Equation (SDDDE) model, which is designed to accommodate various types of delays, including constant, time-dependent, and state-dependent delays. However, it is important to emphasize that this approach cannot handle delays that are continuous, meaning those expressed through integrals, as typically found in integro-differential equations. The specific types of delays

Table 1: Existing works that deal with DDEs. Our implementation deals with a wider range of delays (g is a scalar-valued function)

Delay types	τ's Definition	Neural DDE	NPCDDEs	Neural Laplace	Neural SDDDE
References	-	Zhu et al. (2021)	Zhu et al. (2022)	Holt et al. (2022)	This work
Constant	$\tau = a$	$\surd$	$\times$	$\surd$	$\surd$
Piece-wise constant	$\tau = \lfloor \frac{t-a}{a} \rfloor a$	$\times$	$\surd$	$\surd$	$\surd$
Time-dependent	$\tau = g(t)$	$\times$	$\times$	$\surd$	$\surd$
State-dependent	$\tau = g(t, y(t))$	$\times$	$\times$	$\times$	$\surd$
Continuous	$\tau = \int_0^t g(s, y(s))\mathrm{d}s$	$\times$	$\times$	$\times$	$\times$

that our model can accommodate are summarized in Table 1. This table provides a clear overview of the capabilities of the SDDDE model regarding different delay types. A generic Delay Differential Equation is described by:

$$\begin{aligned} &\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}_\theta(t, \mathbf{y}(t), \mathbf{y}(t - \tau_1(t, \mathbf{y}(t))), \ldots, \mathbf{y}(t - \tau_k(t, \mathbf{y}(t)))) \\ &\mathbf{y}(t < 0) = \phi(t), \end{aligned} \tag{6}$$

where $\phi : \mathbb{R}^- \to \mathbb{R}^d$ is the history function, $\tau_i : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}^+$ a delay function and $\mathbf{f}_\theta : [0, T] \times \mathbb{R}^d \times \cdots \times \mathbb{R}^d \to \mathbb{R}^d$ a parameterized network. In Appendix A, we provide more general and detailed information on DDEs, how we integrate them and discuss memory and time complexities for Neural SDDDE.

3. Methods

In the following, we discuss similarities, drawbacks and benefits of Neural SDDDE compared with the following models: NODE, ANODE and Neural Laplace. We recall that Neural SDDDE is a direct method for solving DDEs, specifically designed to handle delayed systems. This is not the case for ANODE where flexibility is obtained by introducing a higher dimensional space. Lastly, Neural Laplace can solve a broader class of differential equations, although with some limitations that we pinpoint in the following.

From the theoretical viewpoint, the Laplace transformation is often a tool used in proofs on DDEs (Bellman and Cooke, 1963) since it allows to transform linear functional equations in $\mathbf{f}(\mathbf{y}(t))$ involving derivative and differences into linear equations involving only $\mathbf{F}(\mathbf{s})$. Thus, time-dependent and constant delay DDEs are transformed into linear equations of $\mathbf{F}(\mathbf{s})$ using the Laplace transform. This transformation enables Neural Laplace to bypass the explicit definition of delays, whereas Neural SDDDE needs the delays to be specified unless the vector flow $\mathbf{f}$ and delays are learnt jointly. These observations tie Neural Laplace to Neural SDDDE as they can be seen as similar models but living in different domains. However, a limitation of Laplace transformation-based approaches is that they are not defined for DDEs with state-dependent delays, thus restricting the class of time-delayed equations that one can solve with this technique.

Neural Laplace is a model that needs memory initialized latent variables, i.e., a long portion of the solution trajectory needs to be fed in order to get a reasonable representation of the latent variable. In contrast, by design, NODE and ANODE require information only at the initial time to predict the system dynamics. From this viewpoint, Neural SDDDE lies between these two frameworks as it requires a history function $\boldsymbol{\phi}(t)$ to be provided for $t \in [-\tau_{\max}, 0]$, where $\tau_{\max}$ is the maximum delay encountered during integration. This is more demanding than solely relying on an initial condition $\mathbf{y}(0)$ but far less than memory-based latent variables methods. Moreover, the training and testing schemes for Neural Laplace is constrained by this very same observation since future events can only be predicted after a certain observation time. This also makes the length of the trajectory to feed to Neural Laplace a hyperparameter to tune. This is not the case with NODE, ANODE and our approach.

4. Experiments

We evaluate and compare the Neural SDDDE on several dynamical systems (time, state-dependent and constant delays) listed below coming from biology and population dynamics. We show that Neural SDDDE outperforms a variety of continuous-depth models and demonstrates its capabilities in simulating delayed systems. For all the systems listed in this section, data generation information is gathered in Appendix C.

4.1. Description of the test cases

Time-dependent delay system Here, we study a time-dependent delayed logistic equation (Arino et al., 2006):

$$\frac{dy(t)}{dt} = y(t)\big[1 - y(t - \tau(t))\big], \tag{7}$$

with $\tau(t) = 2 + \sin(t)$. We integrate in the time range $[0, 20]$ and define the constant history function $\phi(t) = x_0$, where x_0 is sampled uniformly from $[0.1, 2.0]$.

State-dependent time-delay system In this example, we consider the 1-D state-dependent Mackey Glass system from Dads et al. (2022) with a state-dependent delay:

$$\frac{dy(t)}{dt} = -\alpha(t)y(t) + \beta(t)\frac{y^2(t-\tau(y))}{1 + y^2(t-\tau(y))} + \gamma(t) \tag{8}$$

with $\alpha(t) = 4 + \sin(t) + \sin(\sqrt{2}t) + \frac{1}{1+t^2}$, $\beta(t) = \gamma(t) = \sin(t) + \sin(\sqrt{2}t) + \frac{1}{1+t^2}$ and $\tau(y) = \frac{1}{2}\cos(y(t))$. The model is defined on the time range $[0, 10]$ and the constant history function is $\phi(t) = x_0$ with x_0 sampled uniformly from $[0.1, 1]$.

Delayed Diffusion Equation Finally, we choose the delayed PDE taken from Arino et al. (2009). Such dynamics can for example model single species growth in a food-limited environment.

$$\frac{\partial u}{\partial t}(x,t) = D\frac{\partial^2 u}{\partial x^2}(x,t) + ru(x,t)\left(1 - u(x, t-\tau)\right), \tag{9}$$

where $D = 0.01, r = 0.9$ and $\tau = 2$. We integrate in the time range $[0, 4]$, the spatial domain is $\mathcal{D}_x = [0, 1]$ with periodic boundary conditions and define the history function

$\phi(x,t) = a\sin(x)e^{-0.01t}$ where a is uniformly sampled from $[0.1, 4.0]$. The spatial domain is discretized with a uniform grid of resolution $\Delta x = 0.01$.

4.2. Evaluation

We assess the performance of the models with their ability to predict future states of a given system. The metric used is the mean square error (MSE) in all cases. Neural Laplace predicts only after a burn-in time since a part of the observed trajectory is used to learn a latent initial condition vector $\mathbf{p}$. Since NODE, ANODE and Neural SDDDE can be seen as initial value problems (IVPs) we produce trajectories from initial conditions and compute the MSE with respect to the whole trajectory. On each DDE system, to assess the quality of each model, we elaborate additional experiments alongside with the *test set predictions* that can be found in Appendix D.

As a reminder, to produce outputs, NODE and ANODE need an initial condition, Neural SDDDE the history function and Neural Laplace a portion of the trajectory. To ensure a fair comparison in our experiments, we opt to give Neural Laplace the same information as Neural SDDDE, specifically the history function. For one of the models presented above, the Neural Laplace method is given a much larger chunk of the trajectory, in accordance with what its authors were considering.

4.3. Results

Test errors for each dynamical system are reported in Table 2. Complementary information are included in the Appendix B for what concerns the training process; model and training hyperparameters.

	Time Dependent DDE	State-Dependent DDE	Delay Diffusion
NODE	$.72 \pm .086$	$.0355 \pm .00064$	$.0029 \pm .0014$
ANODE	$.00962 \pm .00368$	$.00011 \pm .000071$	$.00087 \pm .00035$
Neural Laplace	$.00191 \pm .0006$	$.00049 \pm .00078$	$\mathbf{.00064 \pm .00016}$
Neural SDDDE	$\mathbf{.000989 \pm .00017}$	$\mathbf{.0000215 \pm .00001}$	$.00075 \pm .00019$

Table 2: Test MSE averaged over 5 runs (random model initialization seed) of each experiment with their standard deviation. Best result bolded.

Testset prediction Neural SDDDE almost consistently outperforms all other models across the DDE systems discussed in Section 4, as demonstrated in Figures 1, 2, 3, and 4. Neural Laplace appears to suffer from the Runge phenomenon (such a phenomenon is a problem of oscillation at the edges of an interval that occurs when using polynomial interpolation with polynomials of high degree over a set of equispaced interpolation points), particularly evident in the State-Dependent DDE (Figure 2). This issue likely stems from the ILT algorithm providing too few query points. As expected, Neural ODE is the most limited model, generally predicting only the mean trajectory of the dynamical systems being considered. ANODE yields satisfactory results, with the exception of the Time Dependent

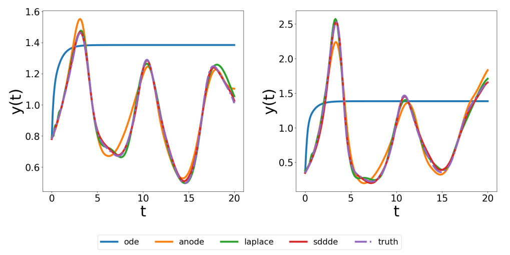

Figure 1: Time Dependent DDE randomly sampled test trajectory plots

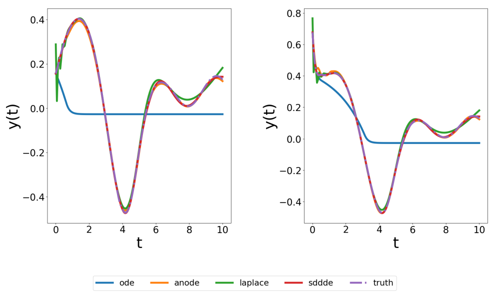

Figure 2: State Dependent DDE randomly sampled test trajectory plots

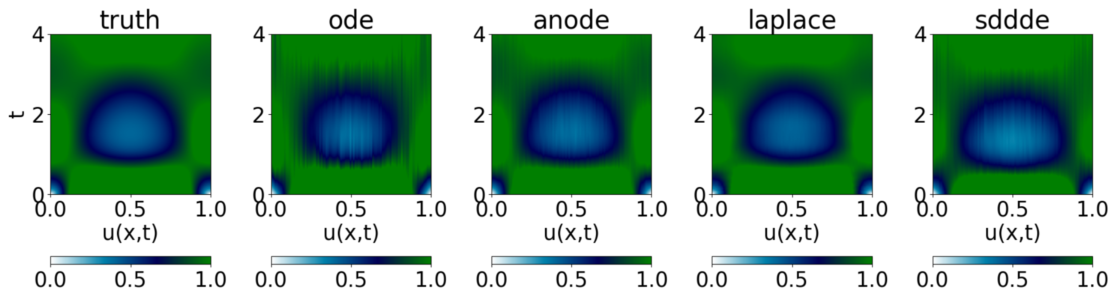

Figure 3: Diffusion Delay PDE randomly sampled from the testset

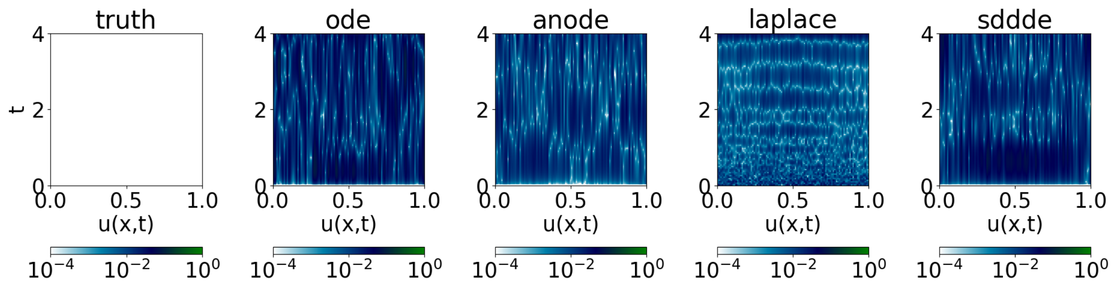

Figure 4: Absolute error of Diffusion Delay PDE randomly sampled from the testset

DDE (Figure 1). For the Diffusion Delay PDE, all models predict the PDE's evolution with an absolute error reaching up to 10^{-2}, as shown in Figure 3. The absolute error, depicted in Figure 4, illustrates the discrepancies between the models. Neural Laplace produces more errors across the entire spatial domain for given time steps, while IVP models have errors localized in specific spatial regions.

Increasing trajectory fed for Neural Laplace Instead of providing the same history as for Neural SDDDE, Neural Laplace is now provided 50% of the trajectory to build its latent initial condition representation vector **p**. To that purpose, the first half of the trajectory is used in Neural Laplace to predict the second half. We choose to train the model on the Time Dependent DDE along with the same training procedure (see Appendix B). Provided the test MSE of Table 2, we choose to only compare the test MSE of Neural SDDDE and Neural Laplace in Table 3. By comparing Figure 5 and 1 one can clearly note that the Runge phenomenon is almost absent and predictions are almost as good as Neural SDDDE. This confirms that, in general, Neural Laplace needs more than the history function in order to correctly simulate DDEs.

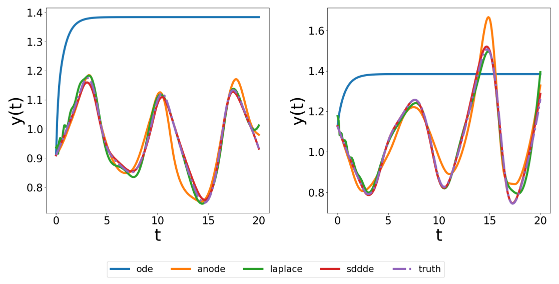

Figure 5: Time-dependent DDE randomly sampled testset trajectories where 50% of data is fed to Neural Laplace

	Test MSE
Neural Laplace	$.00125 \pm .000798$
Neural SDDDE	$\mathbf{.000989 \pm .00017}$

Table 3: Time Dependent test MSE averaged over 5 runs with their standard deviation. Best result bolded.

5. Conclusion and Future Work

In this paper, we introduced Neural State-Dependent Delays Differential Equations (Neural SDDDE) capable of solving DDEs with any type of delays via neural networks. This open-source, robust python DDE solver compatible with neural networks pushes the current envelope of Neural DDEs by handling the delays in a more generic way. To the best of our knowledge, no machine learning library or open-sourced code is available to model such a large class of DDEs.

To validate the effectiveness of Neural SDDDE, we conducted a series of benchmark tests, comparing it against NODEs, the augmented version ANODE, and Neural Laplace. These numerical experiments covered a diverse range of models, including time- and state-dependent scenarios, as well as a delayed Partial Differential Equation (PDE). Our findings revealed that Neural SDDDE accurately reproduced the dynamics across all scenarios tested. Furthermore, Neural SDDDE demonstrated superior performance in terms of accuracy and reliability when compared to the other established methods.

We believe this flexible and versatile tool may provide a valuable contribution to several fields such as control theory where time-delay are often considered. In particular, it may prove useful in learning a model for partially observed systems whose dynamics of observables can be learned, under mild conditions, from their time-history.

Acknowledgments

This research is funded by the grant from Agence Nationale de Recherche : Project number ANR-20-CE23-0025-01.

References

Julien Arino, Lin Wang, and Gail SK Wolkowicz. An alternative formulation for a delayed logistic equation. *Journal of theoretical biology*, 241(1):109–119, 2006.

O. Arino, M.L. Hbid, and E.A. Dads. *Delay Differential Equations and Applications: Proceedings of the NATO Advanced Study Institute held in Marrakech, Morocco, 9-21 September 2002.* Nato Science Series II:. Springer Netherlands, 2009. ISBN 9789048104079.

Krishnan Balachandran. An existence theorem for nonlinear delay differential equations. *Journal of Applied Mathematics and Simulation*, 2(2):85–89, 1989.

Richard Ernest Bellman and Kenneth L. Cooke. *Differential-Difference Equations.* RAND Corporation, Santa Monica, CA, 1963.

Ricky TQ Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. Neural ordinary differential equations. *Advances in neural information processing systems*, 31, 2018.

Ian Cooper, Argha Mondal, and Chris G Antonopoulos. A sir model assumption for the spread of covid-19 in different communities. *Chaos, Solitons & Fractals*, 139:110057, 2020.

E Ait Dads, B Es-sebbar, and L Lhachimi. Almost periodicity in time-dependent and state-dependent delay differential equations. *Mediterranean Journal of Mathematics*, 19(6): 259, 2022.

John R Dormand and Peter J Prince. A family of embedded Runge-Kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Rodney D. Driver. Existence and stability of solutions of a delay-differential system. *Archive for Rational Mechanics and Analysis*, 10(1):401–426, Jan 1962. ISSN 1432-0673.

Emilien Dupont, Arnaud Doucet, and Yee Whye Teh. Augmented neural ODEs. *Advances in Neural Information Processing Systems*, 32, 2019.

Irving R Epstein. Differential delay equations in chemical kinetics: Some simple linear model systems. *The Journal of Chemical Physics*, 92(3):1702–1712, 1990.

Michael Ghil, Ilya Zaliapin, and Sylvester Thompson. A delay differential model of ENSO variability: parametric instability and the distribution of extremes. *Nonlinear Processes in Geophysics*, 15(3):417–433, 2008.

Will Grathwohl, Ricky TQ Chen, Jesse Bettencourt, Ilya Sutskever, and David Duvenaud. Ffjord: Free-form continuous dynamics for scalable reversible generative models, 2018.

Jack K Hale. A stability theorem for functional-differential equations. *Proceedings of the National Academy of Sciences*, 50(5):942–946, 1963.

Jack K Hale. Functional differential equations. In *Analytic Theory of Differential Equations: The Proceedings of the Conference at Western Michigan University, Kalamazoo, from 30 April to 2 May 1970*, pages 9–22. Springer, 2006.

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

Samuel I Holt, Zhaozhi Qian, and Mihaela van der Schaar. Neural Laplace: Learning diverse classes of differential equations in the Laplace domain. In *International Conference on Machine Learning*, pages 8811–8832. PMLR, 2022.

Andrew Keane, Bernd Krauskopf, and Henk A Dijkstra. The effect of state dependence in a delay differential equation model for the El Niño southern oscillation. *Philosophical Transactions of the Royal Society A*, 377(2153):20180121, 2019.

Jacob Kelly, Jesse Bettencourt, Matthew J Johnson, and David K Duvenaud. Learning differential equations that are easy to solve. *Advances in Neural Information Processing Systems*, 33:4370–4380, 2020.

Patrick Kidger, James Morrill, James Foster, and Terry Lyons. Neural controlled differential equations for irregular time series. In *Advances in Neural Information Processing Systems*, volume 33, pages 6696–6707, 2020.

Nicholas Minorsky. Self-excited oscillations in dynamical systems possessing retarded actions. *Journal of Applied Mechanics*, 1942.

Nicolas Morales, Liang Gu, and Yuqing Gao. Adding noise to improve noise robustness in speech recognition. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, volume 2, pages 930–933, 08 2007. doi: 10.21437/Interspeech.2007-335.

Anatolii Dmitrievich Myshkis. General theory of differential equations with retarded arguments. *Uspekhi Matematicheskikh Nauk*, 4(5):99–141, 1949.

Kenneth W. Neves. Automatic integration of functional differential equations: An approach. *ACM Trans. Math. Softw.*, 1(4):357–368, dec 1975. ISSN 0098-3500. doi: 10.1145/355656.355661. URL https://doi.org/10.1145/355656.355661.

H. J. Oberle and H. J. Pesch. Numerical treatment of delay differential equations by hermite interpolation. *Numerische Mathematik*, 37(2):235–255, Jun 1981. ISSN 0945-3245. doi: 10.1007/BF01398255. URL https://doi.org/10.1007/BF01398255.

Baker Paul. Computing stability regions - runge-kutta methods for delay differential equations. *IMA Journal of Numerical Analysis*, 14:347–362, 04 1993. doi: 10.5642/codee.201209.01.10.

Hans Pinckaers and Geert Litjens. Neural ordinary differential equations for semantic segmentation of individual colon glands, 2019.

Marc R Roussel. The use of delay differential equations in chemical kinetics. *The Journal of Physical Chemistry*, 100(20):8323–8330, 1996.

Y Rubanova, RT Chen, and D Duvenaud. Latent ODEs for irregularly-sampled time series (2019), 2019.

Lawrence Shampine and Skip Thompson. Delay-differential equations with constant lags. *CODEE Journal*, 9:1–5, 01 2012. doi: 10.5642/codee.201209.01.10.

Andrei G Vladimirov, Dmitry Turaev, and Gregory Kozyreff. Delay differential equations for mode-locked semiconductor lasers. *Optics letters*, 29(11):1221–1223, 2004.

David Widmann and Chris Rackauckas. Delaydiffeq: Generating delay differential equation solvers via recursive embedding of ordinary differential equation solvers, 2022. URL https://arxiv.org/abs/2208.12879.

Fan Wu, Sanghyun Hong, Donsub Rim, Noseong Park, and Kookjin Lee. Mining causality from continuous-time dynamics models: An application to tsunami forecasting, 2022.

Zhonghui You, Jinmian Ye, Kunming Li, Zenglin Xu, and Ping Wang. Adversarial noise layer: Regularize neural network by adding noise. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 909–913, 2019.

Qunxi Zhu, Yao Guo, and Wei Lin. Neural delay differential equations, 2021.

Qunxi Zhu, Yifei Shen, Dongsheng Li, and Wei Lin. Neural piecewise-constant delay differential equations, 2022.

Juntang Zhuang, Tommy Tang, Yifan Ding, Sekhar Tatikonda, Nicha Dvornek, Xenophon Papademetris, and James S. Duncan. Adabelief optimizer: Adapting stepsizes by the belief in observed gradients, 2020.

Hossein Zivari-Piran and Wayne H Enright. An efficient unified approach for the numerical solution of delay differential equations. *Numerical Algorithms*, 53(2):397–417, 2010.

Appendix A. Overview in DDE integration

This Appendix section is a self-contained introduction on DDE integration.

A.1. Definition

We recall the definition: a delay differential equation (DDE) is defined by

$$\begin{aligned}&\frac{d\mathbf{y}(t)}{dt} = \mathbf{f}_\theta(t, \mathbf{y}(t), \mathbf{y}(t-\tau_1), \dots, \mathbf{y}(t-\tau_k))\\ &\tau_i = \tau_i(t, \mathbf{y}(t)), \quad \forall i \in \{1, 2, \dots, k\}\\ &\mathbf{y}(t<0) = \boldsymbol{\phi}(t),\end{aligned} \tag{10}$$

where $\boldsymbol{\phi} : \mathbb{R}^- \to \mathbb{R}^d$ is the history function, $\tau_i : \mathbb{R} \times \mathbb{R}^d \to \mathbb{R}$ a delay function and $\mathbf{f}_\theta : [0, T]^k \times \mathbb{R}^d \to \mathbb{R}^d$ can be a parameterized network.

Some problems can arise in DDEs that can cause numerical difficulties. First, breaking points may occur in various derivatives of the solution $\mathbf{y}$. Second, a delay may vanish, i.e., $\tau_i \to 0$. The first difficulty is due to the presence of delays terms. In general, DDEs possess a derivative jump (or discontinuity, breaking point) at the initial time point $t = 0$ because

$$\phi'(t = 0^-) \neq \mathbf{y}'(t = 0^+)$$

Moreover, the history function ϕ may also have discontinuities too. Discontinuities can then arise and propagate from the history function and initial point in $\mathbf{y}$ or its higher derivatives (Zivari-Piran and Enright, 2010). The second issue may force the solver to take too many small steps. Zivari-Piran and Enright (2010) transforms the DDE problem into a discontinuous initial value problem (IVP). Alike ODEs, existence and uniqueness theorems for DDEs are based on the continuity of the functions with respect to t and Lispchitz continuity with respect to $\mathbf{y}$ and its delayed counterparts $\mathbf{y}(t-\tau(t, \mathbf{y}))$. For constant, time-dependent and state-dependent delays, these problems have been widely investigated by Bellman and Cooke (1963); Balachandran (1989); Driver (1962) and Hale (2006).

A.2. Example sketch of integrating a simple DDE

Let us consider a first order DDE with a constant time delay τ and a constant history function $\phi(t) = \mathbf{y}_0$. In the most general case, we have our first discontinuity at $t = 0$ since $\phi'(t = 0^-) \neq \mathbf{y}'(t = 0^+)$, so we need to be careful on integration on these derivative jumps.

$$\frac{d\mathbf{y}(t)}{dt} = f(t, \mathbf{y}(t), \mathbf{y}(t-\tau)), \quad \text{with} \quad \mathbf{y}(t<0) = \mathbf{y}_0 \tag{11}$$

On the time interval $t \in]0; \tau[$ there are no discontinuities and the DDE becomes

$$\frac{d\mathbf{y}(t)}{dt} = f(t, \mathbf{y}(t), \mathbf{y}_0), \quad \text{with} \quad \mathbf{y}(0) = \mathbf{y}_0$$

This formulation problem reshapes the DDE problem into an ODE one that we know how to solve with ease. Let us introduce the interpolated function $\phi_1(t)$ to be the solution of the DDE of Equation 11 on the interval $]0;\tau[$.

On the time interval $t \in]\tau; 2\tau[$ there are no discontinuities and the DDE becomes

$$\frac{d\mathbf{y}(t)}{dt} = f(t, \mathbf{y}(t), \phi_1(t)), \quad \text{with} \quad \mathbf{y}(\tau) = \phi_1(\tau)$$

Once again we have an ODE on this interval. Iteratively, we can solve the DDE on successive intervals and this will yield a piecewise continuous solution because of the initial point discontinuity.

One might have noticed that during an integration step, we need the interpolated function of $\mathbf{y}(t-\tau)$. This means that the DDE method is based on the *continuous extensions* of numerical ODE schemes.

A.3. Discontinuity tracking

In the general case, discontinuity that arise from the delay terms are not known a priori unless we are dealing with constant delays (Shampine and Thompson, 2012). During each integration step of our DDE, one must check for discontinuities by checking the roots of the following functions g_{is} (Zivari-Piran and Enright, 2010). Let us stipulate that we integrate from t_n to t_{n+1} and the previous detected discontinuities are $\{\lambda_{-m}, \ldots, \lambda_0, \ldots, \lambda_{r-1}\}$ where the first $m+1$ jumps $\{\lambda_{-m}, \ldots, \lambda_0\}$ are given by the history function and the initial point and the rest were found during previous integration steps. Let

$$\forall (i, s) \in [0, \ldots, r] \times [-m, \ldots, r-1], \quad g_{is}(t) = t - \tau_i - \lambda_s \tag{12}$$

The new discontinuity λ_r is defined as

$$\lambda_r = min\{\lambda > \lambda_{r-1}, \lambda \text{ is a root of odd multiplicity of } g_{is}(t)\} \tag{13}$$

If λ_r is null then the integration step is valid otherwise you redo one from t_n to λ_r. A detailed algorithm procedure is given in Zivari-Piran and Enright (2010).

The first to do such an iterative process to find the discontinuities λ_r and modify the integration step bounds is Paul (1993). An alternative approach relies on stepsize control was proposed by Oberle and Pesch (1981) and Neves (1975). These methods, give up on tracking the discontinuities, which are instead assumed to be automatically included by estimating the error of the integration step. A rejected step will result in a detection of a discontinuity jump and this is the default implementation done in Julia *DelayDiffEq* package (Widmann and Rackauckas, 2022).

For an ODE method of order p, we usually ask the solution to be at least $\mathcal{C}^{p+1}$ continuous. Therefore, to have a successful integration, it is crucial to include in the mesh of points all of the discontinuity of $y^{(k)}$ at least for $k \leq p+1$. Consequently, discontinuity tracking needs to abide by these rules.

A.4. Unconstrained time stepping

Regardless of the method chosen to integrate a DDE; relying on the error estimate of the stepsize method or tracking the breaking points, being able to take arbitrarily large steps that are suggested by the numerical solver is a nice to have. This means that sometimes the formulation of our problem becomes implicit because our approximation solution $\mathbf{y}$ applied to all delays terms in our integration step is simply not yet known. This makes the overall method implicit even if the discrete method we are using is explicit. We call this occurence overlapping, Zivari-Piran and Enright (2010) shows that the issue at hand is well defined and solvable for time and state dependent delays. Let us briefly describe the algorithmic procedure when we are dealing with overlapping (ie $t_{n+1} - t_n > \tau$). Given equation 11 and an integration step from t_n to t_{n+1}. $\mathbf{y}(t-\tau)$ is at the very best partially known and need to be extrapolated to get an good approximation of $\mathbf{y}(t_{n+1})$. The following actions are taken :

- Choose an initial guess for the interpolant Π_n of $\mathbf{y}(t-\tau)$ in $[t_n; t_{n+1}]$.
- Compute the solution $\mathbf{y}(t_{n+1})$ using the interpolant Π_n and by stepping the solver
- Update the interpolant Π_n using the computed solution
- End if the interpolant has converged

The initial guess is usually the extrapolation of the interpolant of the previous step and the end criterion of convergence can vary across cases.

A.5. Pseudo code for DDE solver

Following the detailed explanation of the challenges posed by DDE, we present the pseudo code of the DDE solver implemented by Zivari-Piran and Enright (2010) that is detailed in Algorithm 1, where the general outline of one integration step of a DDE is shown; the DDE solver is illustrated in Algorithm 2. For sake of simplicity, we suppose for the pseudo code a single time delay DDE since the general case does not differ from it.

Algorithm 1 Pseudo code for one DDE numerical integration step

```
1: **Input:**
   Vector field $\mathbf{f}(t, \mathbf{y}, \mathbf{y}(t-\tau))$
   Integration bound $t_n$, $t_{n+1}$
   Interpolated estimated solution $\hat{\mathbf{y}}(t)$ in $[t_0; t_n]$
   Set of detected discontinuities $\Lambda = \{\lambda_{-m}, \dots, \lambda_0, \dots, \lambda_{r-1}\}$.
2: **if** $t_{n+1} - t_n > \min(\Lambda)$ **then**
3:    Declare the interpolant $\Pi_n = \hat{\mathbf{y}}$ of $\mathbf{y}(t-\tau)$ in $[t_n; t_{n+1}]$
4:    **while** the interpolant $\Pi_n$ has not converged **do**
5:       Define $\mathbf{f}_{\text{ODE}}(t, \mathbf{y}) = \mathbf{f}(t, \mathbf{y}, \Pi_n(t))$
6:       Step the solver $\mathbf{y}(t_{n+1}) = ODESolve(\mathbf{f}_{\text{ODE}}, t_n, t_{n+1}, \hat{\mathbf{y}}(t_n))$
7:       Update $\Pi_n$ using the computed solution $\mathbf{y}(t_{n+1})$.
8:    **end while**
9: **else**
10:   Define $\mathbf{f}_{\text{ODE}}(t, \mathbf{y}) = \mathbf{f}(t, \mathbf{y}, \hat{\mathbf{y}}(t-\tau))$
11:   Step the solver $\mathbf{y}(t_{n+1}) = ODESolve(\mathbf{f}_{\text{ODE}}, t_n, t_{n+1}, \hat{\mathbf{y}}(t_n))$
12: **end if**
13: Determine next time step $t_{\text{next}}$ from solver
14: **if** step is accepted **then**
15:   Return updated $\hat{\mathbf{y}}(t)$, next integration bounds $t_{n+1}, t_{\text{next}}$ and $\Lambda$.
16: **else**
17:   Check for discontinuities in $[t_n; t_{n+1}]$ i.e
18:   $\lambda_r = \min\{\lambda > \lambda_{r-1} : \lambda \text{ is a root of odd multiplicity of } g_i(t, y(t)), i \leq r-1\}$
19:   where $g_i(t, \mathbf{y}(t)) = t - \tau(t, \mathbf{y}(t)) - \lambda_i$
20:   **if** a discontinuity is found, $\lambda_{r+1}$ **then**
21:      Return same $\hat{\mathbf{y}}(t)$, next integration bounds $t_n, \lambda_{r+1}$ and $\Lambda \cup \{\lambda_r\}$ .
22:   **else**
23:      Return same $\hat{\mathbf{y}}(t)$, next integration bounds $t_n$, $t_{next}$ and $\Lambda$.
24:   **end if**
25: **end if**
26: **Output:**
   Interpolated estimated solution
   Next integration bounds
   Updated set of discontinuities
```

Algorithm 2 Pseudo code for DDE solver

```
1: Input:
   Vector field $\mathbf{f}(t, \mathbf{y}, \mathbf{y}(t - \tau))$
   Integration bound $t_0$, $t_F$
   History function $\phi(t)$
   Set of history function's discontinuities $\Lambda = \{\lambda_{-m}, \ldots, \lambda_0\}$.
2: Choose an initial $dt$
3: Declare $t_n = t_0, t_{n+1} = t_0 + dt$
4: Declare interpolated estimated solution $\hat{\mathbf{y}}(t) = \phi(t)$ for $t < t_0$
5: repeat
6:    Algorithm 1
7: until $t_{n+1} = t_F$
```

A.6. Memory and computational complexity

Neural SDDDEs rely on an ODE solver, thus function evaluations are associated with the same computational cost as NODEs. However, Neural SDDDE has some extra constraints making the method more computationally involved. Hereafter, we compare the complexity of these two schemes; we define S as the number of stages in the Runge-Kutta (RK) scheme used for the time-integration, N the total number of integration steps, and d the state's dimension.

Memory complexity DDE integration necessitates keeping a record of all previous states in memory due to the presence of delayed terms. This is because at any given time t, the DDE solver must be able to accurately determine $\mathbf{y}(t-\tau)$ through interpolation of the stored state history. This extra amount of extra memory needed depends on the solver used. For example, the additional memory required when using a RK solver for one trajectory is $O(SNd)$. The model's memory footprint is not affected by the number of delays.

Time complexity In comparison to NODE, the solution estimate $\widehat{\mathbf{y}}$ needs to be evaluated for each delayed state argument, i.e., $\mathbf{y}(t - \tau_i)$. The cost of evaluating the interpolant is small compared to the cost of computing its coefficients. Similarly, the time complexity is conditioned by the solver used. For example, for a RK scheme, the coefficient computation scales linearly with the number of stages. Hence, Neural SDDDE adds a time cost $O(SD\,d)$ compared to NODE for each vector field function evaluation.

Appendix B. Training information

Table 4 sums up the MLP architecture of each IVP model (i.e., NODE, ANODE and Neural SDDDE) for each dynamical system. ANODE has an arbitrary augmented state of dimension 10 except for the PDE that has 100. Neural Laplace's architecture is the default one taken from the official implementation for all systems. The learning rate and the number of epochs are the same for all models. The optimizer used is `AdaBelief` (Zhuang et al., 2020). Table 5 gives the number of parameters for each model. In all of our experiments, we used the Dopri5 solver across all of our models.

	Width	Depth	Activation	Epochs	lr
Time Dependent DDE	64	3	relu	2000	.001
State Dependent DDE	64	3	relu	1000	.001
Diffusion PDE DDE	128	3	relu	500	.0001

Table 4: Model and training hyperparameters

	NODE	ANODE	Neural DDE	Neural Laplace
Time Dependent DDE	8513	9815	8578	17194
State Dependent DDE	8513	9815	8642	17194
Diffusion PDE DDE	58852	84552	71653	17194

Table 5: Number of parameters for each DDE system

Appendix C. Data generation parameters

We expose in Table 6 the parameters used for each dataset generation. The start integration time is always $T_0 = 0$. T_F refers to the end time integration. `NUM_STEPS` equally spaced points are sampled in $[T_0, T_F]$. The specific delays `DELAYS` and the constant history function $\phi(t)$ function domain are given. Each training dataset is comprised of 256 datapoints and the testset of 32 datapoints. We used our own DDE solver to generate the data (Dopri5 solver (Dormand and Prince, 1980) was used.). We then double-checked and compared its validity with Julia's DDE solver. $\mathcal{U}$ refers to the uniform distribution. For example, Time Dependent DDE's constant history function value is uniformly sampled between 0.1 and 2.0. For the Diffusion Delay PDE the history function value in the column $\phi(t)$ refers to the constant a defined in Section 4.

	T_F	num_steps	delays	$\phi(t)$
Time Dependent DDE	20.0	200	$2\sin(t)$	$\mathcal{U}(0.1, 2.0)$
State Dependent DDE	10.0	150	$0.5\cos(\mathbf{y}(t))$	$\mathcal{U}(0.1, 1.0)$
Diffusion PDE DDE	4.0	100	1.0	$\mathcal{U}(0.1, 4.0)$

Table 6: Dataset generation information

Appendix D. Additional Experimentation

Hereafter, we discuss about the supplementary experiments undertaken to assess the models' quality. In the initial supplementary experiment, we alter the history function $\phi(t)$ which results in modified system behavior, thereby creating new trajectories for section 4's dynamical systems. This first experiment places each model into a *pure extrapolation regime*; the constant value of the history function $\phi(t) = \mathbf{y}_0$ is sampled outside the range of the training and testing data. This allows to see the models' extrapolation capabilities. The second experiment is more a hybrid approach where the *history function is a step function*:

$$\phi(t) = \begin{cases} \mathbf{y}_0 & t \leq t_{\text{jump}} \\ \mathbf{y}_1 & \text{otherwise} \end{cases}$$

$$t_{\text{jump}} \sim \mathcal{U}(-\tau_{\max}, 0), \quad \mathbf{y}_0, \mathbf{y}_1 \sim \mathcal{U}(c_0, c_1)$$

where t_{jump} is the largest delay in the system and c_0, c_1 are system specific randomly sampled values (see Appendix E for more details). Not only can the nature of the history step function change but can also have its domain function outside of the training and test data (extrapolation regime).

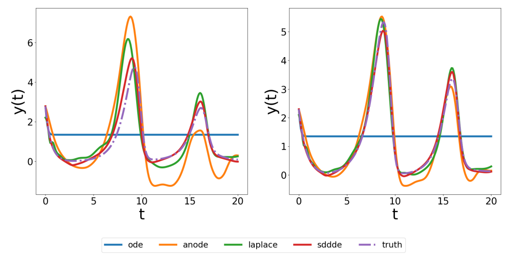

Figure 6: Time Dependent DDE randomly sampled extrapolated trajectory plots

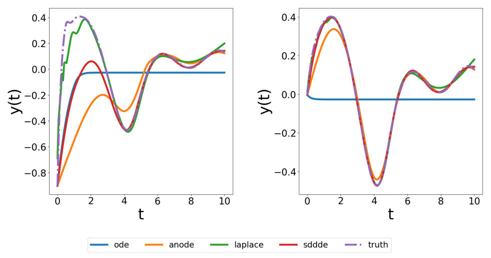

Figure 7: State Dependent DDE randomly sampled extrapolated trajectory plots

Extrapolation regime prediction This experiment really challenges model generalization capabilities. Overall, on certain datasets, some models can extrapolate with new constant history functions that are not too far out from the function domain of history functions used during training; more in details, trajectories were generated with $\phi(t) = x_0 \in [a, b]$: some models are able to exhibit adequate predictions for history functions that have a value near the bounds of $[a, b]$. For the Time Dependent system (Figure 6), Neural SDDDE yields better results compared to the other models. NODE produces the trajectory's mean field while ANODE captures the dynamics main trend but with amplitude discrepancies. For the State-Dependent DDE (Figure 7), Neural SDDDE once again efficiently captures the dynamics while the other models fail. For the Diffusion Delay PDE displayed in Figure 8 & 9, overfitting is observed for Neural Laplace. Out of all the IVP models, Neural SDDDE predicts the best possible outcome compared to NODE and ANODE.

Step history function prediction This third appendix experiment also demonstrates how the modification of the history function leads to changes in the transient regime and

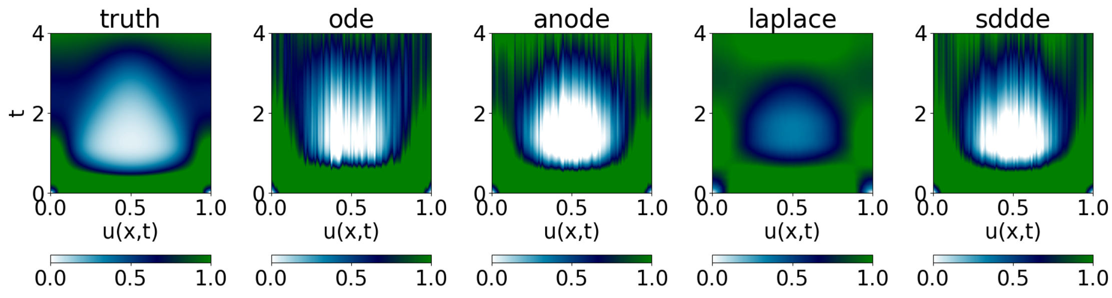

Figure 8: Diffusion Delay PDE randomly sampled from the extrapolated testset

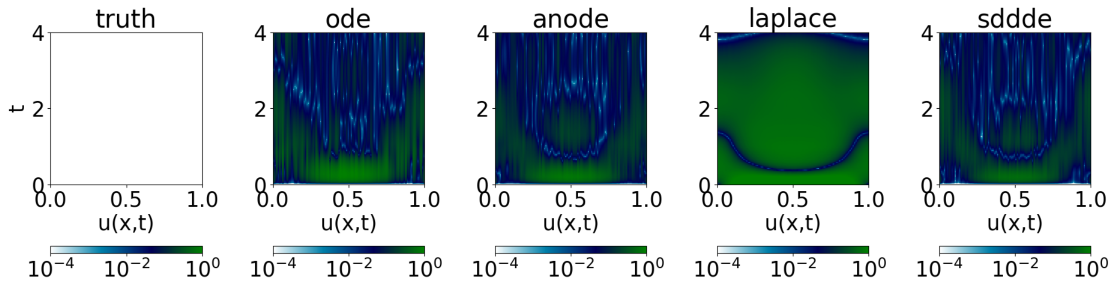

Figure 9: Absolute error of Diffusion Delay PDE randomly sampled from the extrapolated testset

impacts later dynamics. Neural Laplace fails to generate adequate trajectories for the Time-dependent DDE system (Figure 10) and the State-Dependent DDE (Figure 11). NODE and ANODE do not generalize well compared to Neural SDDDE that accurately predicts the dynamics. By studying the effect of such a new history function on the Diffusion Delay PDE, we saw that the system's dynamics is not changed substantially, therefore, we decided to omit this system's comparison.

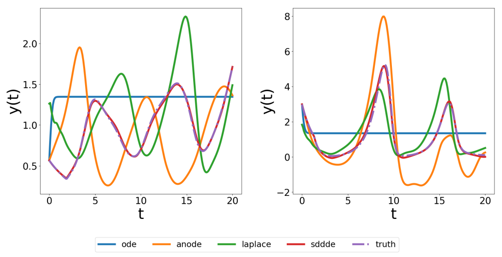

Figure 10: Time Dependent DDE randomly sampled from history step function

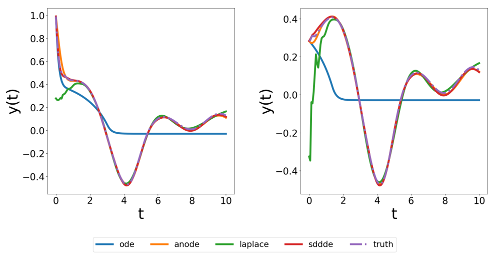

Figure 11: State Dependent DDE randomly sampled from history step function

Noise analysis Finally, we also conduct a noise study on one of the datasets, the Time Dependent DDE system. Each data point is added Gaussian noise that is scaled with a certain factor α of the trajectory's variance. The model is then trained with this noisy data and evaluated on the noiseless testset. In our experiment, we selected 4 scaling factors α: $0.02, 0.05, 0.1$ and 0.2. Results in Table 7 show that our model is robust to noisy data and almost consistently outperforms other models. Additionally, results from Table 7 show that adding a small amount of noise (here $\alpha = 0.02$) makes the learning process more robust, a common result in Machine Learning (Morales et al., 2007; You et al., 2019).

	NODE	ANODE	Neural Laplace	Neural SDDDE
$\alpha = 0$	$1.01 \pm .435$	$.00729 \pm .00235$	$.0014 \pm .00046$	$\mathbf{.00148 \pm .000872}$
$\alpha = 0.02$	$.720 \pm .00254$	$.0128 \pm .002377$	$.00881 \pm .00254$	$\mathbf{.000906 \pm .000441}$
$\alpha = 0.05$	4.032 ± 4.225	$.03655 \pm .0349$	$.00977 \pm .00146$	$\mathbf{.00250 \pm .000951}$
$\alpha = 0.1$	1.597 ± 1.100	$.0223 \pm .00634$	$.0154 \pm .00501$	$\mathbf{.0121 \pm .00534}$
$\alpha = 0.2$	$1.02 \pm .282$	$.0321 \pm .00319$	$.0273 \pm .00704$	$\mathbf{.0186 \pm .00524}$

Table 7: Test MSE with the noiseless data averaged over 5 runs of each Time Dependent DDE noise experiments with their standard deviation. Best result bolded.

Appendix E. Additional Experiment hyperparameters

In table 8 we give the parameters used for each experiment. Extrapolated $\phi(t)$ indicates the possible value of the constant history function (first appendix experiment). $\tau_{\max}$ and c_0, c_1 are described in appendix D (second appendix experiment). For the Diffusion Delay PDE, as stated in appendix D, the other history step function is omitted.

	Extrapolated $\phi(t)$	$\tau_{\max}$	c_0	c_1
Time Dependent DDE	$\mathcal{U}(2.0, 3.0)$	3.0	0.1	3.0
State Dependent DDE	$\mathcal{U}(-1.0, 0.1)$	1/2	-1.0	1.0

Table 8: System specific values for each testing experiment