Application of neural ordinary differential equations to the prediction of multi-agent systems

Sebastian Herzog¹ · Florentin Wörgötter¹

Received: 30 July 2021 / Accepted: 11 November 2021 / Published online: 26 November 2021
© The Author(s) 2021

Abstract

Dynamic systems are usually described by differential equations, but formulating these equations requires a high level of expertise and a detailed understanding of the observed system to be modelled. In this work, we present a data-driven approach, which tries to find a parameterization of neural differential equations system to describe the underlying dynamic of the observed data. The presented method is applied to a multi-agent system with thousand agents.

Keywords Artificial neural networks · Neural ordinary differential equations · Data-driven modelling · Hamiltonian

1 Introduction

We present an approach to learn the dynamics of hundreds to thousands of agents in a system, where the underlying assumption is that the agents follow a collective dynamic, which is not obviously recognisable, but present. In this way, we hope to show scientists a method that allows them to verify model assumptions based on ordinary differential equations on their data. For the purposes of this report, we shall limit ourselves to the Hamiltonian of canonical classical mechanic equations with which movements of objects can be described by their position and velocity. Examples for such dynamics systems are flocks of birds or the movement of cell components. The underlying dynamics allow many conclusions, e.g. about the energy distribution in the system or whether there are symmetries or not.

To visualise the motion of many particles or of a swarm with many agents several approaches exists, like the group of optical flow methods [8, 9, 12], cross-correlation [1] and

Kalman filters [19], these methods can be translated into more complex approaches like particle tracking velocimetry (PTV) [5, 14, 15, 20]. If a sufficient amount of data is available and agent/particle density is high enough, a vector field can be obtained by mapping the trajectories on a Cartesian grid. Thus, it is possible to visualise the dynamics of the underlying system. However, these approaches often result in an algorithmic frameworks of high complexity. The approach presented here is based on neural ordinary differential equations (nODEs) [3]. An artificial neural network (ANN) is used to approximate the Hamiltonian of canonical classical mechanic equations. This allows describing the underlying particle- or agent-swarm directly as a dynamic system without detours. To train the here-presented approach only a list of positions with their momentum is necessary. In this study, we derive the method for this and apply it to simulated data based on the boids [18] algorithm.

2 Methods

Considering a set of N coordinates $(\mathbf{q}, \mathbf{p})$, where $\mathbf{q} \in \mathbb{R}^N$ represents the positions and $\mathbf{p} \in \mathbb{R}^N$ their momentum. A scalar function $\mathcal{H}(\mathbf{q}, \mathbf{p})$ is called the Hamiltonian if:

$$\dot{\mathbf{q}} = \frac{d\mathbf{q}}{dt} = \frac{\partial \mathcal{H}}{\partial \mathbf{p}} \text{ and } \dot{\mathbf{p}} = \frac{d\mathbf{p}}{dt} = -\frac{\partial \mathcal{H}}{\partial \mathbf{q}}. \quad (1)$$

This system can be rephrased without restricting generality to

This work was presented in part at the joint symposium with the 15th International Symposium on Distributed Autonomous Robotic Systems 2021 and the 4th International Symposium on Swarm Behavior and Bio-Inspired Robotics 2021(Online, June1–4, 2021).

✉ Sebastian Herzog
sherzog3@gwdg.de

¹ Department for Computational Neuroscience, Third Institute of Physics-Biophysics, University of Göttingen, Göttingen, Germany

$$\dot{\mathbf{q}} = \nabla_{\mathbf{p}} \mathcal{H}(\mathbf{q}, \mathbf{p}) \text{ and } \dot{\mathbf{p}} = -\nabla_{\mathbf{q}} \mathcal{H}(\mathbf{q}, \mathbf{p}), \quad (2)$$

where ∇

is the gradient operator. For nODEs it is common to approximate (Eq. 2) by ANNs, denoted by $\mathcal{H}_{\theta}(\mathbf{q}, \mathbf{p})$, where θ is a vector with all parameters of the ANN. A (feedforward) artificial neural network, also known as a multi-layer perceptron (MLP), is a series of logistic regression models stacked on top of each other. MLPs can be used for classification problems as well as for regression problems, depending on the final layer. In this work, we are solving regression problems defined by:

$$p(y|\mathbf{x}, \theta) = \mathcal{N}(y|\mathbf{w}^T \mathbf{z}(\mathbf{x}), \sigma^2) \quad (3)$$

$$\mathbf{z}(\mathbf{x}) = q(\mathbf{V}\mathbf{x}) = [g(\mathbf{v}_1^T \mathbf{x}), \dots, g(\mathbf{v}_H^T \mathbf{x})],$$

where y is the desired regression output, $\mathbf{x}$ the input vector, $\mathbf{w}$ the weight matrix, $\mathbf{v}_j = \mathbf{V}_{j,:}$ is the j th row of $\mathbf{V}$, hidden layers $\mathbf{z}(\mathbf{x}) = \phi(\mathbf{x}, \mathbf{V})$ of size H , where g is the so-called activation function. $\mathcal{N}(\cdot, \sigma^2)$ is the symbol for the normal distribution with variance σ^2 . The hyperbolic tangent (tanh) is used as the activation function. To use (Eq. 3) for $\mathcal{H}_{\theta}(\mathbf{q}, \mathbf{p})$ the framework of nODEs is used, which are a family of ANNs where the hidden layers parameterize the derivative of the hidden state using a neural network and the output of the network is computed using a black-box differential equation solver, where backpropagation [2, 4, 10] through the ODE solver is applied [3]. A special feature of nODEs is that the gradient is calculated using the adjointed sensitivity method [17], allowing a linear relation between problem size and computational complexity and a low memory consumption. After obtaining the gradient, the parameters of $\mathcal{H}_{\theta}(\mathbf{q}, \mathbf{p})$ are trained by a gradient method with adaptive moment estimation, called ADAM [11]. The input data are all pairs of $(\mathbf{q}, \mathbf{p})$ in the training-set. And the loss function which is supposed to be optimised is

$$l_{\theta} = \|\dot{\mathbf{q}} - \nabla_{\mathbf{p}} \mathcal{H}_{\theta}(\mathbf{q}, \mathbf{p})\|_2^2 + \|\dot{\mathbf{p}} - \nabla_{\mathbf{q}} \mathcal{H}_{\theta}(\mathbf{q}, \mathbf{p})\|_2^2 + \alpha \|\theta\|_1, \quad (4)$$

where the first two terms simply describe the quadratic deviation, $\|\cdot\|_2^2$ denotes the l_2 -norm and the third term $\alpha \|\theta\|_1$, with $\|\cdot\|_1$ denotes the l_1 -norm and forces sparsity on θ for with $\alpha \in \mathbb{R}$ a weighting parameter (in this work $\alpha = 0.3$). While the details mentioned above may only differ in technical details from the nODE approach as described in [3], the addition of the sparsity term is a novelty to [3] which has proven to be necessary for the convergence to achieve the results discussed later. The implementation of $\mathcal{H}_{\theta}(\mathbf{q}, \mathbf{p})$ was done with PyTorch [16] and is based on [3]. The network architecture consists of five dense layers, each layer has an

activation layer with tanh. The first layer expects 2 inputs and has 128 nodes, followed by four more layers with 128 nodes and finally output layer with one output node.

In this work, we have deliberately avoided dividing the data into a training, validation and test set, because we are interested in the vector field in the end, which is not directly evident from the data anyhow. A generalization beyond the spatio-temporal domain of the training data is not the goal of this study. In the following results two points are examined. First, the learned vector field is validated if it can reproduce the ground truth trajectories from the boid algorithm, when using the initial conditions from the boid algorithm. Second, the trained system is used on new data, but with the same parameters for the random sources and compared to the nearest neighbours of the training data.

3 Results

To investigate the presented approach empirically, a simulation of the flocking behaviour of birds is used. For this simulation 1000 agents with separation, alignment and cohesion rules were modelled by the boids algorithm [18]. The boids algorithm with reflecting boundaries was implemented using python [21] and the numpy package [7]. The 1000 agents were placed randomly centred around (0, 0) (Fig. 1) but without overlap and with random initial velocities from $\mathcal{U}[0.2, 5]$ and random accelerations of $\mathcal{U}[0, 0.03]$, where $\mathcal{U}[a, b]$ is the uniform distribution between a and b . The system was simulated $t_{max} = 5000$ time steps. These data were then used to train the presented approach with early stopping [6] and a patience of 15. In (Fig. 2) the values of the loss function over the epochs are plotted. Early stopping ends the training if the loss function does not improve over the last 15 epochs. The system was trained on a Nvidia GTX 1080, an epoch calculates on average 61 ± 3 seconds. In total 54 min 54 s, but probably half of the time would also be sufficient. After training the vector field of the nODE can be visualised, by generating an equidistant grid with $n = 50$ points and applying the learned $\mathcal{H}_{\theta}(\cdot, \mathbf{0})$ to each point. The resulting vector field is illustrated in (Fig. 3). Based on the learned vector field it's possible to predict trajectories, using some values for $\mathbf{q}$ and $\mathbf{p}$ as initial conditions and applying this to the field. To check if the presented approach learned the underlying dynamic from the boid algorithm, all $\mathbf{q}$ and $\mathbf{p}$ from the training data were selected. A few exemplary trajectories are shown in (Fig. 4). The corresponding statistics are available in blue in (Fig. 5). It can be seen (Fig. 5) that the trajectories deviate about 7 units of length in the median, but there are also cases where the deviate exceeds 80 units, with a domain size of $10^3 \times 10^3$ this is a comparably small error. To validate our approach further, we took the predicted nODE and randomly initialised 1000 agents with the

Fig. 1 Snapshots of 1000 agent system simulated with the boids algorithm [18] at different times (from left to right: $t = 0$, $t = 500$, $t = 1000$). Each dot represents an agent and the borders are reflective

Fig. 2 Development of the train loss. Early stopping, with a patience of 15 stopped the training after 57 epochs. It can be seen that the training started to oscillate from around epoch 30

same scheme as above and calculated the trajectories based on nODE over 5000 time steps. For each trajectory, the nearest neighbour of the original trajectories was searched and the l_2 -deviation was calculated. This deviations can be seen in Fig. 5 in light orange. The expectation of the distribution is about 80 length units, which in relation to the total domain size again is not much.

4 Discussion

The approach presented here is an extension of the classic nODE approach. By adding a sparse regularisation term to (Eq. 4), it was possible to train the nODE network to learn the underlying dynamic of the agent motions from the boids algorithm. Surprisingly, even the resulting trajectories from the learned system only have a maximum deviation of 7 length units (divided by the domain width this equals 0.7%) compared to the ground truth data, which means that the underlying dynamic of the training data can be reconstructed. Here, it should be taken into account that the ANN

Fig. 3 Visualisation of the learned vector field. The colours (in colour: blue to red) are an indication of the energy in the system (colour figure online)

used to learned the vector field has a very high degree of freedom which allows it to overfit the training data and describe even complex trajectories. Considering randomly initialised agents, from the same source distribution, the deviation increases in the median to up to 80 length units (or 8%) which is an increase of a factor of 10, but given the large domain this is still quite a good result. It also should be noted that no hyperparameter optimisation was considered for the approach presented at this stage. It is, therefore, to be expected that, if the loss function or the architecture of the network is further adapted, even better results could be achieved. It is also worth mentioning that we have chosen ADAM [11] because of its fast convergence, other algorithms such as SGD [13] may achieve better results, too. Methods like these need further investigation to understand

Fig. 4 Selection of trajectories for visualization purposes. The ground truth trajectories are coloured in blue, while the trajectories generate from the nODE are coloured in light orange (colour figure online)

Fig. 5 In blue on the left side: Statistical error distribution of the trained nODE, based on the l_2 -deviation to the original trajectories from the boid algorithm. In light orange on the right: Statistical error distribution of the trained nODE with 1000 newly initiated agents. The deviation was calculated to the nearest neighbour to the original trajectories from the training data (colour figure online)

in detail in which cases they generalise and in which not. However, we hope that approaches like the presented one can be helpful to gain new insights into systems where models based on first principles are hard to obtain. Especially, applications where only a part of the dynamics are described by first-principles can benefit from such “hybrid” modelling approaches where the unknown part of the dynamic is approximated by an nODE.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Bracewell RN, Bracewell RN (1986) The Fourier transform and its applications, vol 31999. McGraw-Hill New York
2. Bryson AE (1961) A gradient method for optimizing multi-stage allocation processes. In: Proc. Harvard Univ, Symposium on digital computers and their applications
3. Chen Ricky TQ, Rubanova Y, Bettencourt J, Duvenaud D (2018) Neural ordinary differential equations. In: Proceedings of the 32nd International Conference on Neural Information Processing Systems, NIPS'18, page 6572–6583, Red Hook, NY, USA, Curran Associates Inc
4. Dreyfus SE (1962) The numerical solution of variational problems. *J Math Anal Appl* 5:30–45
5. Fuchs T, Hain R, Kähler CJ (2017) Non-iterative double-frame 2D/3D particle tracking velocimetry. *Exp. Fluids* 58:119
6. Goodfellow Ian J, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge. <http://www.deeplearningbook.org>
7. Harris CR, Millman KJ, van der Gommers Walt SJ, Virtanen PR, Cournapeau D, Wieser E, Taylor J, Berg S, Smith NJ, Kern R, Picus M, Hoyer S, Kerkwijk MH, van Brett M, Haldane A, del R'io JF, Wiebe M, Peterson P, G'erard-Marchant P, Sheppard K, Reddy T, Weckesser W, Abbasi H, Gohlke C, Oliphant TE, (2020) Array programming with NumPy. *Nature* 585(7825):357–362
8. Berthold KPH, Schunck BG (1981) Determining optical flow. *Artif Intell* 17(1):185–203
9. Jason JY, Harley AW, Derpanis KG (2016) Back to basics: unsupervised learning of optical flow via brightness constancy and motion smoothness. In: European Conference on Computer Vision, pages 3–10. Springer
10. Kelley HJ (1960) Gradient theory of optimal flight paths. *ARS J* 30(10):947–954
11. Diederik PK, Jimmy B (2017) A method for stochastic optimization, Adam
12. Kroeger T, Timofte R, Dai D, Gool LV (2016) Fast optical flow using dense inverse search. In: *European Conference on Computer Vision*, pages 471–488. Springer
13. LeCun YA, Bottou L, Orr Genevieve B, Müller K-R (2012) Efficient BackProp. Springer, Berlin, Heidelberg, pp 9–48
14. Gruen A, Maas HG, Papantoniou D (1993) Particle tracking velocimetry in three-dimensional flows. *Exp. Fluids* 15(2):133–146
15. Nishino K, Kasagi N, Hirata M (1989) Three-dimensional particle tracking velocimetry based on automated digital image processing
16. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) Pytorch: an imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d Alché-Buc F, Fox E, Garnett R (eds) *Advances in Neural Information Processing Systems* 32, pages 8024–8035. Curran Associates, Inc
17. Pontryagin LS (2018) Mathematical theory of optimal processes. Routledge
18. Reynolds CW (1987) Flocks herds and schools: A distributed behavioral model. In: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '87, page 25–34. Association for Computing Machinery, New York
19. Jinya S, Li B, Chen W-H (2015) On existence, optimality and asymptotic stability of the kalman filter with partially observed inputs. *Automatica* 53:149–154
20. Tanida Y, Miyashiro H (1992) Flow visualization VI, chapter: 3D particle tracking velocimetry-its possibilities and limitations. Springer, Berlin
21. Van Rossum Guido, Drake Fred L (2009) Python 3 reference manual. CreateSpace, Scotts Valley

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.