Enforcing constraints for time series prediction in supervised, unsupervised and reinforcement learning

Panos Stinis
Advanced Computing, Mathematics and Data Division
Pacific Northwest National Laboratory, Richland WA 99354

Abstract

We assume that we are given a time series of data from a dynamical system and our task is to learn the flow map of the dynamical system. We present a collection of results on how to enforce constraints coming from the dynamical system in order to accelerate the training of deep neural networks to represent the flow map of the system as well as increase their predictive ability. In particular, we provide ways to enforce constraints during training for all three major modes of learning, namely supervised, unsupervised and reinforcement learning. In general, the dynamic constraints need to include terms which are analogous to memory terms in model reduction formalisms. Such memory terms act as a restoring force which corrects the errors committed by the learned flow map during prediction.

For supervised learning, the constraints are added to the objective function. For the case of unsupervised learning, in particular generative adversarial networks, the constraints are introduced by augmenting the input of the discriminator. Finally, for the case of reinforcement learning and in particular actor-critic methods, the constraints are added to the reward function. In addition, for the reinforcement learning case, we present a novel approach based on homotopy of the action-value function in order to stabilize and accelerate training. We use numerical results for the Lorenz system to illustrate the various constructions.

Introduction

Scientific machine learning, which combines the strengths of scientific computing with those of machine learning, is becoming a rather active area of research. Several related priority research directions were stated in the recently published report (Baker et al. 2019). In particular, two priority research directions are: (i) how to leverage scientific domain knowledge in machine learning (e.g. physical principles, symmetries, constraints); and (ii) how can machine learning enhance scientific computing (e.g reduced-order or sub-grid physics models, parameter optimization in multiscale simulations).

Our aim in the current work is to present a collection of results that contribute to both of the aforementioned priority research directions. On the one hand, we provide ways to enforce constraints coming from a dynamical system during the training of a neural network to represent the flow map of the system. Thus, prior domain knowledge is incorporated in the neural network training. On the other hand, as we will show, the accurate representation of the dynamical system flow map through a neural network is equivalent to constructing a temporal integrator for the dynamical system modified to account for unresolved temporal scales. Thus, machine learning can enhance scientific computing.

We assume that we are given data in the form of a time series of the states of a dynamical system (a training trajectory). Our task is to train a neural network to learn the flow map of the dynamical system. This means to optimize the parameters of the neural network so that when it is presented with the state of the system at one instant, it will predict accurately the state of the system at another instant which is a fixed time interval apart. If we want to use the data alone to train a neural network to represent the flow map, then it is easy to construct simple examples where the trained flow map has rather poor predictive ability (Stinis et al. 2019). The reason is that the given data train the flow map to learn how to respond accurately as long as the state of the system is on the trajectory. However, at every timestep, when we invoke the flow map to predict the estimate of the state at the next timestep, we commit an error. After some steps, the predicted trajectory veers into parts of phase space where the neural network has not trained. When this happens, the neural network's predictive ability degrades rapidly.

One way to aid the neural network in its training task is to provide data that account for this inevitable error. In (Stinis et al. 2019), we advanced the idea of using a noisy version of the training data i.e. a noisy version of the training trajectory. In particular, we attach a noise cloud around each point on the training trajectory. During training, the neural network learns how to take as input points from the noise cloud, and map them back to the *noiseless* trajectory at the next time instant. This is an *implicit* way of encoding a restoring force in the parameters of the neural network. We have found that this modification can improve the predictive ability of the trained neural network but up to a point.

We want to aid the neural network further by enforcing constraints that we know the state of the system satisfies. In particular, we assume that we have knowledge of the differential equations that govern the evolution of the system

(our constructions work also if we assume algebraic constraints see e.g. (Stinis et al. 2019)). Enforcing the differential equations directly at the continuum level can be effected for supervised and and reinforcement learning but it is more involved for unsupervised learning. Here we have opted to enforce constraints in discrete time. We want to incorporate the discretized dynamics into the training process of the neural network. The purpose of such an attempt can be explained in two ways: (i) we want to aid the neural network so that it does not have to discover the dynamics (physics) from scratch; and (ii) we want the constraints to act as regularizers for the optimization problem which determines the parameters of the neural network.

Closer inspection of the concept of noisy data and of enforcing the discretized constraints reveals that they can be combined. However, this needs to be done with care. Recall that when we use noisy data we train the neural network to map a point from the noise cloud back to the noiseless point at the next time instant. Thus, we cannot enforce the discretized constraints as they are because the dynamics have been modified. In particular, the use of noisy data requires that the discretized constraints be modified to account *explicitly* for the restoring force. We have called the modification of the discretized constraints the *explicit* error-correction.

The meaning of the restoring force is analogous to that of memory terms in model reduction formalisms (Chorin and Stinis 2006). Note that the memory here is *not* because we are only resolving part of the system's variables (see e.g. (Ma, Wang, and E 2018; Harlim et al. 2019)) but due to the use of a *finite* timestep. The timescales that are smaller than the timestep used are *not* resolved explicitly. However, their effect on the resolved timescales cannot be ignored. In fact, it is what causes the inevitable error at each application of the flow map. The restoring force that we include in the modified constraints is there to remedy this error i.e. to account for the unresolved timescales albeit in a simplified manner. This is precisely the role played by memory terms in model reduction formalisms. In the current work we have restricted attention to *linear* error-correction terms. The linear terms come with coefficients whose magnitude is optimized as part of the training. In this respect, optimizing the error-correction term coefficients becomes akin to *temporal renormalization*. This means that the coefficients depend on the temporal scale at which we probe the system (Goldenfeld 1992; Barenblatt 2003). Finally, we note that the error-correction term can be more complex than linear. In fact, it can be modeled by a separate neural network. It can also involve not just the previous state but also states further back in time. Results for such more elaborate error-correction terms will be presented elsewhere.

We have implemented constraint enforcing in all three major modes of learning. For *supervised* learning, the constraints are added to the objective function. For the case of *unsupervised* learning, in particular generative adversarial networks (GANs) (Goodfellow et al. 2014), the constraints are introduced by augmenting the input of the discriminator (Stinis et al. 2019). Finally, for the case of *reinforcement* learning and in particular actor-critic methods (Sutton et al. 1999), the constraints are added to the reward function. In addition, for the reinforcement learning case, we have developed a novel approach based on homotopy of the action-value function in order to stabilize and accelerate training.

In recent years, there has been considerable interest in the development of methods that utilize data and physical constraints in order to train predictors for dynamical systems and differential equations e.g. see (Berry, Giannakis, and Harlim 2015; Raissi, Perdikaris, and Karniadakis 2018; Chen et al. 2018; Han, Jentzen, and E 2018; Sirignano and Spiliopoulos 2018; Felsberger and Koutsourelakis 2018; Wan et al. 2018; Ma et al. 2018) and references therein. Our approach is different, it introduces the novel concept of training on purpose with modified (noisy) data in order to incorporate (implicitly or explicitly) a restoring force in the dynamics learned by the neural network flow map. We have also provided the connection between the incorporation of such restoring forces and the concept of memory in model reduction.

Due to space limitations, we cannot expand on the details of how to enforce constraints for the 3 major modes of learning (please see Sections 1 and 2 in (Stinis 2019) for a detailed discussion of all the constructions). Instead we focus on the presentation of numerical results for the Lorenz system to showcase the performance of the proposed approach. Also, we note that we have not included results which show how enforcing constraints, implicitly or explicitly, is better than not enforcing constraints at all (please see (Stinis et al. 2019) and (Stinis 2019) for such results).

Numerical results

The Lorenz system is given by

$$\frac{dx_1}{dt} = \sigma(x_2 - x_1) \tag{1}$$

$$\frac{dx_2}{dt} = \rho x_1 - x_2 - x_1 x_3 \tag{2}$$

$$\frac{dx_3}{dt} = x_1 x_2 - \beta x_3 \tag{3}$$

where σ, ρ and β are positive. We have chosen for the numerical experiments the commonly used values $\sigma = 10$, $\rho = 28$ and $\beta = 8/3$. For these values of the parameters the Lorenz system is chaotic and possesses an attractor for almost all initial points. We have chosen the initial condition $x_1(0) = 0$, $x_2(0) = 1$ and $x_3(0) = 0$.

We have used as training data the trajectory that starts from the specified initial condition and is computed by the Euler scheme with timestep $\delta t = 10^{-4}$. In particular, we have used data from a trajectory for $t \in [0, 3]$. For all three modes of learning, we have trained the neural network to represent the flow map with timestep $\Delta t = 1.5 \times 10^{-2}$ i.e. 150 times larger than the timestep used to produce the training data. After we trained the neural network that represents the flow map, we used it to predict the solution for $t \in [0, 9]$. Thus, the trained flow map's task is to predict (through iterative application) the whole training trajectory for $t \in [0, 3]$ starting from the given initial condition and then keep producing predictions for $t \in (3, 9]$.

This is a severe test of the learned flow map's predictive abilities for four reasons. First, due to the chaotic nature of the Lorenz system there is no guarantee that the flow map can correct its errors so that it can follow closely the training trajectory even for the interval $[0, 3]$ used for training. Second, by extending the interval of prediction beyond the one used for training we want to check whether the neural network has actually learned the map of the Lorenz system and not just overfitting the training data. Third, we have chosen an initial condition that is far away from the attractor but our integration interval is long enough so that the system does reach the attractor and then evolves on it. In other words, we want the neural network to learn both the evolution of the transient and the evolution on the attractor. Fourth, we have chosen to train the neural network to represent the flow map corresponding to a much larger timestep than the one used to produce the training trajectory in order to check the ability of the error-correcting term to account for a significant range of unresolved timescales (relative to the training trajectory).

We performed experiments with different values for the various parameters that enter in our constructions. We present here indicative results for the case of $N = 2 \times 10^4$ samples ($N/3$ for training, $N/3$ for validation and $N/3$ for testing). We have chosen $N_{cloud} = 100$ for the cloud of points around each input. Thus, the timestep $\Delta t = 1.5 \times 10^{-2}$. This is because there are $20000/100 = 200$ time instants in the interval $[0, 3]$ at a distance $\Delta t = 3/200 = 1.5 \times 10^{-2}$ apart.

The noise cloud for the neural network at a point t was constructed using the point $x_i(t)$ for $i = 1, 2, 3$, on the training trajectory and adding random disturbances so that it becomes the collection $x_{il}(t)(1 - R_{range} + 2R_{range} \times \xi_{il})$ where $l = 1, \ldots, N_{cloud}$. The random variables $\xi_{il} \sim U[0, 1]$ and $R_{range} = 2 \times 10^{-2}$. As we have explained before, we want to train the neural network to map the input from the noise cloud at a time t to the *noiseless* point $x_i(t+\Delta t)$ (for $i = 1, 2, 3$,) on the training trajectory at time $t + \Delta t$.

We have to also motivate the value of R_{range} for the range of the noise cloud. Recall that the training trajectory was computed with the Euler scheme which is a first-order scheme. For the interval $\Delta t = 1.5 \times 10^{-2}$ we expect the error committed by the flow map to be of similar magnitude and thus we should accommodate this error by considering a cloud of points within this range. We found that taking R_{range} slightly larger and equal to 2×10^{-2} helps the accuracy of the training.

We denote by $(F_1(z_j), F_2(z_j), F_3(z_j))$ the neural network flow map prediction at $t_j + \Delta t$ for the input vector $z_j = (z_{j1}, z_{j2}, z_{j3})$ from the noise cloud at time t_j. Also, $x_j^{data} = (x_1(t_j+\Delta t), x_2(t_j+\Delta t), x_3(t_j+\Delta t))$ is the point on the training trajectory computed by the Euler scheme with $\delta t = 10^{-4}$. For the mini-batch size we have chosen $m = 1000$ for the supervised and unsupervised cases and $m = 33$ for the reinforcement learning case.

We also need to specify the constraints that we want to enforce. Using the notation introduced above, we want to train the neural network flow map so that its output $(F_1(z_j), F_2(z_j), F_3(z_j))$ for an input data point $z_j = (z_{j1}, z_{j2}, z_{j3})$ from the noise cloud makes zero the residuals

$$\epsilon_{j1} = F_1(z_j) - z_{j1} - \Delta t[\sigma(z_{j2} - z_{j1})] + \Delta t a_1 z_{j1} \quad (4)$$

$$\epsilon_{j2} = F_2(z_j) - z_{j2} - \Delta t[\rho z_{j1} - z_{j2} - z_{j1} z_{j3}] + \Delta t a_2 z_{j2} \quad (5)$$

$$\epsilon_{j3} = F_3(z_j) - z_{j3} - \Delta t[z_{j1} z_{j2} - \beta z_{j3}] + \Delta t a_3 z_{j3}, \quad (6)$$

where a_1, a_2 and a_3 are parameters to be optimized during training along with the parameters of the neural network flow map. The first three terms on the RHS of (4)-(6) are the forward Euler scheme, while the third is the *diagonal* linear error-correcting term. More elaborate error-correcting terms will appear elsewhere (see also (Stinis 2019)).

Supervised learning

The loss function used for enforcing constraints in supervised learning was

$$Loss = \frac{1}{m}\Big[\sum_{j=1}^{m}\sum_{l=1}^{3}[(F_l(z_j) - x_{jl}^{data})^2 + \epsilon_{jl}^2]\Big], \quad (7)$$

where ϵ_{jl} are the residuals given by (4)-(6). The unconstrained loss function is given by (7) without the residuals.

We used a deep neural network for the representation of the flow map with 10 hidden layers of width 20. We note that because the solution of the Lorenz system acquires values outside of the region of the activation function we have removed the activation function from the last layer of the generator (alternatively we could have used batch normalization and kept the activation function). Fig. 1 compares the evolution of the prediction for $x_1(t)$ of the neural network flow map starting at $t = 0$ and computed with a timestep $\Delta t = 1.5 \times 10^{-2}$ to the ground truth (training trajectory) computed with the forward Euler scheme with timestep $\delta t = 10^{-4}$. We show plots only for $x_1(t)$ since the results are similar for the $x_2(t)$ and $x_3(t)$.

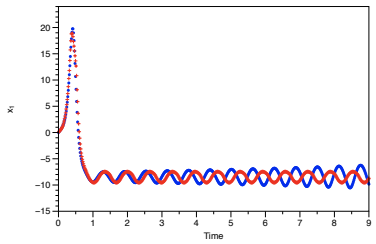
We make two observations. First, the prediction of the neural network flow map is able to follow with adequate accuracy the ground truth not only during the interval $[0, 3]$ that was used for training, but also during the interval $(3, 9]$. Second, the *explicit* enforcing of constraints i.e. the enforcing of the constraints (4)-(6) (see results in Fig. 1(b)) is better than the *implicit* enforcing of constraints.

Unsupervised learning

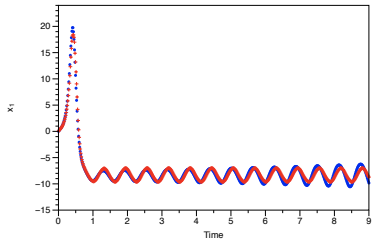
For the case of unsupervised learning we have chosen GANs. To enforce constraints we consider a two-player min-max game with the modified value function $V^{const}(D, G)$:

$$\min_G \max_D V^{const}(D, G) = E_{x\sim p_{data}(x)}[\log D(x, \epsilon_D(x))] + E_{z\sim p_z(z)}[\log(1 - D(G(z), \epsilon_G(z)))], \quad (8)$$

where $\epsilon_D(x) \sim \mathcal{N}(0, (2\delta t)^2)$ is the constraint residual for the true sample (see explanation in Section 2.4 in (Stinis et al. 2019)). Also, $\epsilon_G(z)$ is the constraint residual for the generator-created sample (see (4)-(6) above). The unconstrained value function is given by (8) without the residuals. Note that in our setup, the generator input distribution $p_z(z)$ will be from the noise cloud around the training trajectory.

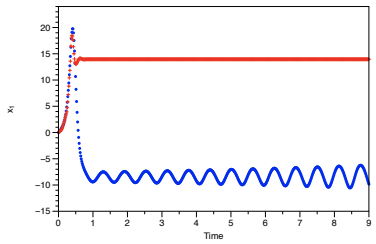


(a)

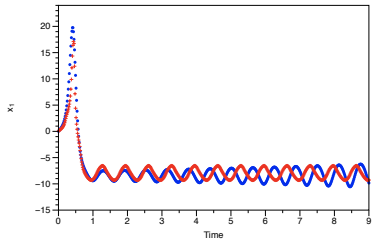


(b)

Figure 1: Supervised learning. Comparison of ground truth for $x_1(t)$ computed with the Euler scheme with timestep $\delta t = 10^{-4}$ (blue dots) and the neural network flow map prediction with timestep $\Delta t = 1.5 \times 10^{-2}$ (red crosses). Note that the timestep of the neural network flow map is 150 times larger than the timestep used to produce the training data. (a) *noisy* data *without enforced constraints* during training; (b) *noisy* data *with enforced constraints* during training (see text for details).



(a)



(b)

Figure 2: Unsupervised learning (GAN). Comparison of ground truth for $x_1(t)$ computed with the Euler scheme with timestep $\delta t = 10^{-4}$ (blue dots) and the neural network flow map (GAN generator) prediction with timestep $\Delta t = 1.5 \times 10^{-2}$ (red crosses). (a) *noisy* data *without enforced constraints* during training; (b) *noisy* data *with enforced constraints* during training (see text for details).

On the other hand, the true data distribution p_{data} is the distribution of values of the (noiseless) training trajectory.

We have used for the GAN generator a deep neural network with 9 hidden layers of width 20 and for the discriminator a neural network with 2 hidden layers of width 20. The numbers of hidden layers both for the generator and the discriminator were chosen as the smallest that allowed the GAN training to reach its game-theoretic optimum without at the same time requiring large scale computations. Fig. 2 compares the evolution of the prediction of the neural network flow map starting at $t = 0$ and computed with a timestep $\Delta t = 1.5 \times 10^{-2}$ to the ground truth (training trajectory) computed with the forward Euler scheme with timestep $\delta t = 10^{-4}$.

Fig. 2(a) shows results for the *implicit* enforcing of constraints. We see that this is not enough to produce a neural network flow map with long-term predictive accuracy. Fig. 2(b) shows the significant improvement in the predictive accuracy when we enforce the constraints *explicitly*. The results for this specific example are not as good as in the case of supervised learning presented earlier. We note that training a GAN with or without constraints is a delicate numerical task as explained in more detail in (Stinis et al. 2019). One needs to find the right balance between the expressive strengths of the generator and the discriminator (game-theoretic optimum) to avoid instabilities but also train the neural network flow map i.e. the GAN generator, so that it has predictive accuracy.

We also note that training with *noiseless* data is even more brittle. For the very few experiments where we avoided instability the predicted solution from the trained GAN generator was not accurate at all.

Reinforcement learning

The last case we examine is that of *reinforcement* learning (see (Stinis 2019) for notation and details about the constructions). In particular, we use a deterministic policy actor-critic method (Lillicrap et al. 2015). In our application we have *identified the neural network flow map with the action policy*. For the representation of the deterministic action policy, we used a deep neural network with 10 hidden layers of width 20. For the representation of the action-value function we used a deep neural network with 15 hidden layers of width 20. The task of learning an accurate representation of the action-value function is more difficult than that of finding the action policy. This justifies the need for a stronger network to represent the action-value function.

The training of actor-critic methods in their original form suffers from stability issues. Researchers have developed various modifications and tricks to stabilize training (see the review in (Pfau and Vinyals 2016)). The one that enabled us to stabilize results in the first place is that of *target networks* (Mnih et al. 2015; Lillicrap et al. 2015). The target network concept uses different networks to represent the action-value function and the action policy that appear in the expression for the target in the Bellman equation. How-

ever, the predictive accuracy of the trained neural network flow map i.e. the action policy, was extremely poor unless we *also* used our homotopy approach for the action-value function. This was true for both cases of enforcing or not constraints explicitly during training. With this in mind we present results with and without the homotopy approach for the action-value function to highlight the accuracy improvement afforded by the use of homotopy.

After each iteration of the optimizer for the action-value function, the homotopy approach uses the quantity

$$\delta \times Q(s_t, \mu(s_t)) + (1-\delta) \times [r_t + \gamma Q(s_{t+1}, \mu(s_{t+1}))] \quad (9)$$

in the optimization for the action policy. Here, $Q(s_t, \mu(s_t))$ is the action-value function, $\mu(s_t)$ is the action policy and r_t the reward function, $\gamma \in [0, 1]$ is the discount factor which expresses the degree of faith in future actions, and δ is the homotopy parameter (see Section 2.3 in (Stinis 2019)). We initialized the homotopy parameter δ at 0, and increased its value (until it reached 1) every 2000 training iterations.

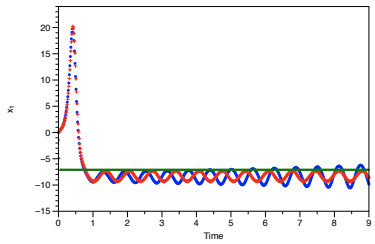
We have set the discount factor to $\gamma = 1$, which is a difficult case. It corresponds to the case of a deterministic environment which means that the same actions always produce the same rewards. This is the situation in our numerical experiments where we are given a training trajectory that does not change. We have conducted more experiments for other values of γ but a detailed presentation of those results will await a future publication.

The reward function (with constraints) for an input point z_j from the noise cloud at time t_j

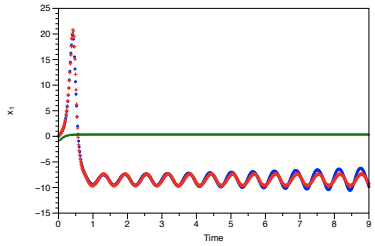
$$r(z_j, x_j^{data}) = -\sum_{l=1}^{3} \left[(\mu_l(z_j) - x_{jl}^{data})^2 + \epsilon_{jl}^2 \right] \quad (10)$$

where x_j^{data} is the *noiseless* point from the training trajectory at time $t_j + \Delta t$. Also, $\mu_l(z_j)$ is the action at z_j i.e. the prediction of the neural network flow map and ϵ_{jl} is the constraint residual for the prediction (see (4)-(6) above).

Fig. 3 presents results of the prediction performance of the neural network flow map when it was trained with and without the use of homotopy for the action value function. In Fig. 3(a) we have results for the *implicit* enforcing of constraints while in Fig. 3(b) for the *explicit* enforcing of constraints. We make two observations. First, both for implicit and explicit enforcing of the constraints, the use of homotopy leads to accurate results for long times. Especially for the case of explicit enforcing which gave us some of the best results from all the numerical experiments we conducted for the different modes of learning. Second, if we do not use homotopy, the predictions are extremely poor both for implicit and explicit forcing. Indeed, the green curve in Fig. 3(a) representing the prediction of $x_1(t)$ for the case of implicit constraint enforcing *without* homotopy is as inaccurate as it looks. It starts at 0 and within a few steps drops to a negative value and does not change much after that. The predictions for $x_2(t)$ and $x_3(t)$ are equally inaccurate.



(a)



(b)

Figure 3: Reinforcement learning (Actor-critic). Comparison of ground truth for $x_1(t)$ computed with the Euler scheme with timestep $\delta t = 10^{-4}$ (blue dots), the neural network flow map prediction with timestep $\Delta t = 1.5 \times 10^{-2}$ *with* homotopy for the action-value function during training (red crosses) and the neural network flow map prediction with timestep $\Delta t = 1.5 \times 10^{-2}$ *without* homotopy for the action-value function during training (green triangles). (a) *noisy* data *without enforced constraints* during training; (b) *noisy* data *with enforced constraints* during training (see text for details).

Discussion and future work

We have presented a collection of results about the enforcing of known constraints for a dynamical system during the training of a neural network to represent the flow map of the system. We have provided ways that the constraints can be enforced in all three major modes of learning, namely supervised, unsupervised and reinforcement learning. In line with the law of scientific computing that one should build in an algorithm as much prior information is known as possible, we observe a striking improvement in performance when known constraints are enforced during training. There is an added benefit of training with noisy data and how these correspond to the incorporation of a restoring force in the dynamics of the system (see (Stinis et al. 2019) and (Stinis 2019) for more details). This restoring force is analogous to memory terms appearing in model reduction formalisms. In our framework, the reduction is in a *temporal* sense i.e. it allows us to construct a flow map that remains accurate though it is defined for *large* timesteps.

The model reduction connection opens an interesting avenue of research that makes contact with complex systems appearing in real-world problems. The use of larger timesteps for the neural network flow map than the ground truth without sacrificing too much accuracy is important. We can imagine an online setting where observations come at *sparsely* placed time instants and are used to update the parameters of the neural network flow map. The use of sparse observations could be dictated by *necessity* e.g. if it is hard

to obtain frequent measurements or *efficiency* e.g. the local processing of data in field-deployed sensors can be costly. Thus, if the trained flow map is capable of accurate estimates using *larger* timesteps then its successful updated training using only sparse observations becomes more probable.

The current approach approximates the flow map using a feed-forward neural network. It will be interesting to compare its performance with other approaches, most notably Recurrent Neural Networks which have been used to model time series data (see e.g. the review (Bianchi et al. 2017)).

The constructions presented in the current work depend on a large number of details that can potentially affect their performance. A thorough study of the relative merits of enforcing constraints for the different modes of learning needs to be undertaken and will be presented in a future publication. We do believe though that the framework provides a promising research direction at the nexus of scientific computing and machine learning.

Acknowledgements

The author would like to thank Court Corley, Tobias Hagge, Nathan Hodas, George Karniadakis, Kevin Lin, Paris Perdikaris, Maziar Raissi, Alexandre Tartakovsky, Ramakrishna Tipireddy, Xiu Yang and Enoch Yeung for helpful discussions and comments. The work presented here was partially supported by the PNNL-funded "Deep Learning for Scientific Discovery Agile Investment" and the DOE-ASCR-funded "Collaboratory on Mathematics and Physics-Informed Learning Machines for Multiscale and Multiphysics Problems (PhILMs)". Pacific Northwest National Laboratory is operated by Battelle Memorial Institute for DOE under Contract DE-AC05-76RL01830.

References

Baker, N.; Alexander, F.; Bremer, T.; Hagberg, A.; Kevrekidis, Y.; Najm, H.; Parashar, M.; Patra, A.; Sethian, J.; Wild, S.; and Willcox, K. 2019. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence.

Barenblatt, G. I. 2003. *Scaling*. Cambridge University Press.

Berry, T.; Giannakis, D.; and Harlim, J. 2015. Nonparametric forecasting of low-dimensional dynamical systems. *Phys. Rev. E* 91:032915.

Bianchi, F. M.; Maiorino, E.; Kampffmeyer, M. C.; Rizzi, A.; and Jenssen, R. 2017. An overview and comparative analysis of recurrent neural networks for short term load forecasting. *arXiv preprint arXiv:1705.04378*.

Chen, R. T. Q.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. 2018. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366v3*.

Chorin, A. J., and Stinis, P. 2006. Problem reduction, renormalization and memory. *Communications in Applied Mathematics and Computational Science* 1:1–27.

Felsberger, L., and Koutsourelakis, P. 2018. Physics-constrained, data-driven discovery of coarse-grained dynamics. *arXiv preprint arXiv:1802.03824v1*.

Goldenfeld, N. 1992. *Lectures on Phase Transitions and the Renormalization Group*. Perseus Books.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. *Advances in neural information processing systems* 2672–2680.

Han, J.; Jentzen, A.; and E, W. 2018. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences* 115(34):8505–8510.

Harlim, J.; Jiang, S. W.; Liang, S.; and Yang, H. 2019. Machine learning for prediction with missing dynamics. *arXiv preprint arXiv:1910.05861*.

Lillicrap, T. P.; Hunt, J. J.; Pritzel, A.; Heess, N.; Erez, T.; Tassa, Y.; Silver, D.; and Wierstra, D. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Ma, H.; Leng, S.; Aihara, K.; Lin, W.; and Chen, L. 2018. Randomly distributed embedding making short-term high-dimensional data predictable. *Proceedings of the National Academy of Sciences* 115(43):E9994–E10002.

Ma, C.; Wang, J.; and E, W. 2018. Model reduction with memory and the machine learning of dynamical systems. *arXiv preprint arXiv:1808.04258v1*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Pfau, D., and Vinyals, O. 2016. Connecting generative adversarial networks and actor-critic methods. *arXiv preprint arXiv:1610.01945*.

Raissi, M.; Perdikaris, P.; and Karniadakis, G. 2018. Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM J. Sci. Comput.* 40:A172–A198.

Sirignano, J., and Spiliopoulos, K. 2018. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics* 375:1339 – 1364.

Stinis, P.; Hagge, T.; Tartakovsky, A. M.; and Young, E. 2019. Enforcing constraints for interpolation and extrapolation in generative adversarial networks. *Journal of Computational Physics* 397.

Stinis, P. 2019. Enforcing constraints for time series prediction in supervised, unsupervised and reinforcement learning. *arXiv preprint arXiv:1905.07501*.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, 1057–1063. Cambridge, MA, USA: MIT Press.

Wan, Z.; Vlachas, P.; Koumoutsakos, P.; and Sapsis, T. 2018. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. *PLoS ONE* 13:e0197704.