Neural Galerkin Schemes with Active Learning for High-Dimensional Evolution Equations

Joan Bruna* Benjamin Peherstorfer* Eric Vanden-Eijnden*

Deep neural networks have been shown to provide accurate function approximations in high dimensions. However, fitting network parameters requires informative training data that are often challenging to collect in science and engineering applications. This work proposes Neural Galerkin schemes based on deep learning that generate training data with active learning for numerically solving high-dimensional partial differential equations. Neural Galerkin schemes build on the Dirac-Frenkel variational principle to train networks by minimizing the residual sequentially over time, which enables adaptively collecting new training data in a self-informed manner that is guided by the dynamics described by the partial differential equations. This is in contrast to other machine learning methods that aim to fit network parameters globally in time without taking into account training data acquisition. Our finding is that the active form of gathering training data of the proposed Neural Galerkin schemes is key for numerically realizing the expressive power of networks in high dimensions. Numerical experiments demonstrate that Neural Galerkin schemes have the potential to enable simulating phenomena and processes with many variables for which traditional and other deep-learning-based solvers fail, especially when features of the solutions evolve locally such as in high-dimensional wave propagation problems and interacting particle systems described by Fokker-Planck and kinetic equations.

1. Introduction

Partial differential equations (PDEs) are used to describe the dynamics of systems in a wide variety of science and engineering applications. Many of these equations, however, are challenging to solve, analytically and even numerically. Classical techniques from scientific computing such as the finite element method and other grid-based methods can work well if the solution domain is low dimensional; however, in higher dimensions, grid-based methods fail because their computational costs increase at an exponential rate with the dimension of the domain. This curse of dimensionality leaves simulating many systems of interest out of scope of classical PDE solvers. For example, interacting particles described by Schrödinger's and other kinetic equations such as Fokker-Planck and Boltzmann equations lead to high-dimensional systems when the number of particles is more than a few. Even multiscale techniques as sophisticated as multigrid, fast multipole, and adaptive-mesh refinement methods fail in such instances. Thus, high-dimensional approximation problems require a fundamentally different notion of discretization than via grids to circumvent the curse of dimensionality. In supervised machine learning applications, such adaptivity is achieved with deep neural networks (DNNs) via feature or representation learning.

The last few years have seen many developments in the direction of using DNNs for numerically solving PDEs, with the introduction of techniques such as physics-informed neural networks and the Deep Ritz method [9, 44, 36, 4]. In these schemes, the solution of a time-dependent PDE is represented with a DNN

*Courant Institute of Mathematical Sciences, New York University, New York, NY 10012

and the parameters of the network are optimized by minimizing the residual of the PDE on collocation points from the temporal and spatial domains. These collocation points can lie on a grid, or be sampled randomly in time-space. Either way, their positions are usually not tailored to the (unknown) solution of the PDE. In high dimensions, the lack of adaptivity in the training procedure ultimately suffers from the curse of dimensionality because all features of the solution have to be discovered without guidance. In addition, the training treats space and time equivalently, which is inefficient for initial value problems if the goal is predicting the PDE solution over long time ranges, and it can lead to a loss of causality in the solution.

In this work we take a different approach. We develop time-integrators for PDEs that use DNNs to represent the solution but update the parameters sequentially from one time slice to the next rather than globally over the whole time-space domain. To achieve this, we follow the well-established Dirac-Frenkel variational principle [8, 15] that is widely used in computational chemistry [28] and scientific computing [23, 43, 19] and that has been considered in the context of PDEs and DNNs in the works [11, 1]. It updates the network parameters so that the PDE residual is minimized as the numerical solution is evolved in time. The key novelty of our Neural Galerkin schemes with active learning is that we combine the sequential updating of the network parameters given via the Dirac-Frenkel variational principle with adaptive sampling ("data collection") over time to determine at which sampling points to evaluate the residual to guide the parameter updates. This means that the measure—from which samples are drawn for estimating the expected residual—is adapted over time so that the samples follow where the important features of the solutions are, which is critical in high-dimensional spatial domains and when the solution has local features that evolve in the spatial domain. Importantly, our scheme uses the PDEs that govern the solution to inform the adaptive sampling as well, rather than relying on a black-box global optimization strategy. No *a priori* data about the PDE solutions is used, however. Overall, we summarize that our approach leverages adaptivity in *both* function approximation and sampling.

Summary of main contributions Our main results can be summarized as follows: 1. Given a nonlinear parametric representation (e.g. via a DNN) of the solution of an initial value problem for a time-dependent PDE, we leverage the Dirac-Frenkel variational principle as in [8, 15, 28, 23, 43, 19, 11, 1] to derive systems of nonlinear evolution equations for the parameters of the nonlinear parametric representation. The system of equations that govern the parameters can then be integrated using standard solvers with different level of sophistication, for example by dynamically choosing the time-step size based on properties of the numerical solution. In particular, with suitable time-integration schemes, the proposed approach takes larger time steps when possible and corrects to smaller time-step sizes if the dynamics of the solution require it. This is in contrast to typical collocation methods such as the Deep Ritz method that samples over the whole time-space domain without having the opportunity to exploit the smoothness of solutions over time and thus has to draw samples even when the solution hardly varies.

2. The evolution equations that we derive for the DNN parameters involve operators that require estimation via sampling in space. We show how to perform this sampling adaptatively, by using the property of the current solution itself, which is again in contrast to collocation methods. We propose a dynamical estimation of the loss, which is shown to be essential in two aspects: First it achieves accuracy in situations where static, non-adaptive sampling methods fail to capture the details of the solution that determine its evolution. Second, it is key to the interpretation of the PDE solution itself, since the solution can be quite complicated, and sampling its feature to compute statistics typically is the only way to exploit and interpret it.

3. We illustrate the viability and usefulness of our approach on a series of test cases. First we consider two equations in low dimensions, the Korteweg-de Vries (KdV) and the Allen-Cahn (AC) equations to show that our method can be used to efficiently capture solutions with localized features that move in the spatial domain. The features are solitons in the context of the KdV equation and domain walls in the context of the AC equation, which can both be modeled with few degrees of freedom using neural networks with specific nonlinear units. Second, we show that our method can be used to simulate PDEs in unbounded, high-dimensional domains, that are out of reach of standard methods. Specifically, we study advection

equations of hyperbolic type with wave speeds that vary in space and time, and Fokker-Planck equations of parabolic type for the evolution of several interacting particles. In both cases, our results show that our method is able to efficiently capture the evolution of the solution and overcome the sampling limitations of collocation-based methods.

Related works The need for adaptive data acquisition in the context of machine learning for problems in science and engineering has been emphasized in previous works [40]. There also is a large body of work on numerically solving PDEs with DNN parametrizations based on collocation over the spatiotemporal domain (space-time discretizations), which makes adaptive sampling challenging; see, e.g., Refs. [9, 44, 36, 4]. Methods that do not rely on collocation typically exploit specific formulations of elliptic and semilinear parabolic PDEs [12, 17], and focus on specific approximation tasks such as learning committor functions [20, 27, 40], coarse-graining [2, 24, 48], or de-noising [42]. Thus, these approaches are limited in their scope and, in particular, are not applicable to pure transport dynamics given by kinetic and hyperbolic equations. There also is a range of surrogate-modeling methods [41, 3] based on nonlinear parametrizations such as [47, 25, 5, 21, 26, 45, 30, 13, 37, 38, 34, 19]; however, these methods require access to training sets of ground truth PDE solutions, which is precisely what is unavailable in our setting because classical PDE solvers are cursed by dimension. The surrogate-modeling method introduced in [26, 22] embeds large state vectors stemming from traditional discretizations of PDEs in low-dimensional manifolds and then propagates forward in time the embedding based on the dynamics given by the traditional discretizations; however, this means that in each time step a lifting from low to high dimensions has to be performed, which is inherently computationally expensive. In contrast, the proposed Neural Galerkin schemes directly integrate the parameters of the PDE solutions, rather than embeddings of high-dimensional states. This approach circumvents the requirement of needing traditional discretizations of the underlying PDEs, which avoids the expensive lifting to higher dimensions that is intractable for problems formulated over high-dimensional spatial domains.

There is a wide range of methods that builds on the Dirac-Frenkel variational principle to update the parameters of nonlinear parametrizations, see, e.g., the works on dynamic low-rank approximations and related methods [23, 6, 19, 50, 18, 43, 33, 31, 29]. Closest to our approach are the works [11, 1], which derive the same dynamics for parameters of nonlinear parametrizations based on the Dirac-Frenkel variational principle, where the work [1] additionally derives schemes that conserve quantities. However, both works [11, 1] focus exclusively either on specific nonlinear parametrizations and PDEs where the objective value can be computed analytically or on low-dimensional problems where data acquisition with static, uniform sampling is tractable. In contrast, our Neural Galerkin schemes with adaptive sampling applies to high-dimensional problems with generic nonlinear parametrizations where adaptive data collection becomes crucial, as will be illustrated by our case studies.

2. Setup and problem formulation

In this section, we formulation the problem of numerically solving time-dependent PDEs, which are sometimes referred to as evolution equations. We are particularly interested in PDEs formulated over potentially high-dimensional spatial domains and with solutions that exhibit local features that are transported over time.

2.1. Evolution equations

Given a spatial domain $\mathcal{X} \subseteq \mathbb{R}^d$, we will consider the evolution of a time-dependent field $u : [0, \infty) \times \mathcal{X} \to \mathbb{R}$ which at all times belongs to some function space $\mathcal{U}$ and whose dynamics is governed by the PDE

$$\begin{aligned} \partial_t u(t, \boldsymbol{x}) &= f(t, \boldsymbol{x}, u) && \text{for } \; (t, \boldsymbol{x}) \in [0, \infty) \times \mathcal{X}\,, \\ u(0, \boldsymbol{x}) &= u_0(\boldsymbol{x}) && \text{for } \; \boldsymbol{x} \in \mathcal{X} \end{aligned} \tag{1}$$

where $u_0 \in \mathcal{U}$ is the initial condition. By appropriately choosing $f : [0,\infty) \times \mathcal{X} \times \mathcal{U} \to \mathbb{R}$, Eq. (1) can represent different PDEs of interest. For example, we get an advection-diffusion-reaction equation by taking

$$f(t, \boldsymbol{x}, u) = b(t, \boldsymbol{x}) \cdot \nabla u + a(t, \boldsymbol{x}) : \nabla\nabla u + G(t, \boldsymbol{x}, u)$$

for a $b : [0,\infty) \times \mathcal{X} \to \mathbb{R}^d$, $a : [0,\infty) \times \mathcal{X} \to \mathbb{R}^d \times \mathbb{R}^d$ and $G : [0,\infty) \times \mathcal{X} \times \mathbb{R} \to \mathbb{R}$. In the following, we only consider problems with appropriate boundary conditions for Eq. (1) that make Eq. (1) well-posed for all $t \in [0,\infty)$, which means that its solution exists, is unique, and depends continuously on $u_0 \in \mathcal{U}$.

2.2. Problem formulation

Widely used approaches for numerically solving PDEs of the form (1) with deep neural networks rely on globally optimizing network parameters over the time-space domain. Let $\tilde{u} : [0,\infty) \times \mathcal{X} \times \Theta \to \mathbb{R}$ be a neural-network parametric approximation of u so that $\tilde{u}(t, \boldsymbol{x}; \theta)$ approximates $u(t, \boldsymbol{x})$ for a parameter $\theta \in \Theta$. Notice that time t is an input of the network function $\tilde{u}$. Widely used collocation-based methods train the network parameters θ of $\tilde{u}$ to globally minimize the expected residual

$$r(t, \boldsymbol{x}; \theta) = \partial_t \tilde{u}(t, \boldsymbol{x}; \theta) - f(t, \boldsymbol{x}, \tilde{u}) \tag{2}$$

of the PDE,

$$\min_{\theta \in \Theta} \mathbb{E}_{(t,\boldsymbol{x})\sim\nu} \left[|r(t, \boldsymbol{x}; \theta)|^2 \right] . \tag{3}$$

The objective of the optimization problem (3) depends on the expected value of the squared residual over the time-space domain with respect to the distribution ν. To estimate the expected value in (3), the typical approach is to draw $i = 1, \ldots, n$ samples $(t_i, \boldsymbol{x}_i)$ from the distribution ν, to form a Monte Carlo estimator based on the drawn samples, and to minimize the Monte Carlo estimate of the expected value as in

$$\min_{\theta \in \Theta} \frac{1}{n} \sum_{i=1}^{n} |r(t_i, \boldsymbol{x}_i; \theta)|^2 , \qquad (t_i, \boldsymbol{x}_i) \sim \nu. \tag{4}$$

However, it can be challenging to ensure that the Monte Carlo estimator in (4) is a good estimate of the expected value in (3). In particular, in high-dimensional spatial domains $\mathcal{X}$ and if the solutions of the PDEs (1) develop spatially localized structures, then uniform (or other un-informed, un-guided, solution-independent) sampling schemes quickly fail to provide accurate Monte Carlo estimators; see also [40]. In fact, even though the deep neural network parametrization might be expressive enough and exploit properties of the solution to circumvent the curse of dimensionality from an approximation-theoretic point of view, minimizing the objective in (4) can be well affected by the curse of dimensionality through the variance of the estimator, especially with local solution features advecting in high-dimensional spatial domains.

3. Neural Galerkin schemes via the Dirac-Frenkel variational principle

In this section, we break with the traditional time-space approach of training DNNs in the context of PDEs and instead formulate the training of deep neural network parametrizations via the Dirac-Frenkel variational principle [8, 15, 28, 11], which prescribes a system of ordinary differential equations (ODEs) for the network parameters. This means that the network parameters are ultimately trained by solving a system of ODEs over time, rather than solving a global optimization problem over the time-space domain as in traditional collocation-based methods. Carrying over the dynamics described by the PDEs of interest to the network parameters is of critical importance to our Neural Galerkin approach because it will allow us to use information from previous time steps to inform finding the solution at the current time step, including the measure against which to estimate the residual.

We emphasize that using the Dirac-Frenkel variational principle to train nonlinear parametrizations has a long history in molecular dynamics and computational chemistry; see [28, 19] for the history and early works. More recent works [11, 1] in the context of deep neural networks derive the same systems of ODEs that we obtain in the following.

3.1. Nonlinear parametrizations that depend on time

We parametrically represent the solution $u(t)$ at time t as $U(\theta(t)) \in \mathcal{U}$ with parameters $\theta(t) \in \Theta$ and $U : \Theta \times \mathcal{X} \to \mathbb{R}$, i.e. we use the ansatz

$$u(t, \boldsymbol{x}) = U(\theta(t), \boldsymbol{x}) \qquad \text{for} \quad (t, \boldsymbol{x}) \in [0, \infty) \times \mathcal{X} \,. \tag{5}$$

Two remarks are in order. First, time t enters the parametrization (5) via the parameter $\theta(t)$, instead of being an input to the network function as in the global, time-space approaches discussed in Section 2.2. This is of critical importance to our approach because it will allow us to use information from previous times to inform approximations of the PDE solution at future times. Second, the parametrization U may depend nonlinearly on $\theta(t)$, which is in stark contrast to the majority of classical approximations in scientific computing that have a linear dependence on the parameter [14]. And it is in stark contrast to traditional, linear model reduction methods that seek linear approximations in subspaces; see [32].

For the representation in Eq. (5) to be complete, i.e. such that for any $u(t) \in \mathcal{U}$ there exists at least one $\theta(t) \in \Theta$ such that $U(\theta(t)) \equiv u(t)$, Θ should itself be an infinite-dimensional function space in general—for more discussion on this point, see the Appendix. In practice, however, we will be interested in situations where the parametric representation U depends on a finite-dimensional parameter $\theta(t)$ such as in a DNN, where $\theta(t)$ denotes the vector of adjustable parameters/weights in this network whose number can be large but is finite—specific DNN representations are discussed below. In this case, using Eq. (5) as ansatz for the solution of Eq. (1) introduces approximation errors, the magnitude of which we aim to keep low at all times.

3.2. Training nonlinear parametrization via the Dirac-Frenkel variational principle

We emphasize that we consider the situation where we do not have access to training data to learn the parameter $\theta(t)$ so that $U(\theta(t))$ approximates well the PDE solution. Instead, we use the structure of the governing equation (1) to find $\theta(t)$ so that the error is low. To this end, note that inserting the ansatz solution $U(\theta(t))$ in Eq. (1), assuming differentiability of $\theta(t)$ and using $\partial_t U(\theta(t)) = \nabla_\theta U(\theta) \cdot \dot{\theta}(t)$, leads to the residual function $r : \Theta \times \dot{\Theta} \times \mathcal{X} \to \mathbb{R}$ defined as

$$r_t(\theta, \eta, \boldsymbol{x}) \equiv \nabla_\theta U(\theta, \boldsymbol{x}) \cdot \eta - f(t, \boldsymbol{x}, U(\theta)) \,, \tag{6}$$

where $\dot{\Theta}$ is the set of time derivatives of $\theta(t)$. Notice that the residual r_t defined in (6) depends on time t, whereas the residual r defined in (2) is global over the time-space domain.

The goal is now to keep the error made by the parametrization described in Eq. (5) low by minimizing the magnitude of the residual r_t with respect to η, given an appropriate metric. One possibility is to do so globally over some time interval $t \in [0, T]$ with $T > 0$; this option, which is discussed in C, has the disadvantage that it leads to a complicated boundary value problem that requires the storage of the entire path $\theta(t)$ for $t \in [0, T]$. Instead, we follow the Dirac-Frenkel variational principle [8, 15, 28], see also [11] for the principle applied in the context of deep neural network, which leads to an initial value problem that can be solved over arbitrary long times. Specifically, we seek $\theta(t)$ such that for all $t > 0$ it holds

$$\dot{\theta}(t) \in \operatorname*{argmin}_{\eta \in \dot{\Theta}} J_t(\theta(t), \eta) \tag{7}$$

where we define the objective function $J_t : \Theta \times \dot{\Theta} \to \mathbb{R}$ as

$$J_t(\theta, \eta) = \frac{1}{2} \int_{\mathcal{X}} |r_t(\theta, \eta, \boldsymbol{x})|^2 d\nu(\boldsymbol{x}). \tag{8}$$

In Eq. (8), ν is a positive measure with full support on $\mathcal{X}$, which we will discuss in detail later in Section 4.

The initial condition θ_0 can be obtained via e.g. minimization of the least-squares loss between u_0 and $U(\theta_0)$:

$$\theta_0 \in \operatorname*{argmin}_{\theta \in \Theta} \int_{\mathcal{X}} |u_0(\boldsymbol{x}) - U(\theta, \boldsymbol{x})|^2 d\nu(\boldsymbol{x}). \tag{9}$$

Similarly, boundary conditions can be enforced either naturally by choosing $U(\theta(t))$ such that these conditions are satisfied for all $\theta(t) \in \Theta$, or by adding appropriate penalty terms to the residual (6) and the objective (8). For more justification of Eq. (11) note that this equation also arises as a consequence of the proposition introduced in B.

3.3. System of ODEs for network parameters

The minimizers of $J_t(\theta(t), \eta)$ solve the Euler-Lagrange equation

$$\nabla_\eta J_t(\theta(t), \eta) = 0\,. \tag{10}$$

Written explicitly, Eq. (10) is a system of ODEs for $\theta(t)$:

$$M(\theta)\dot{\theta} = F(t, \theta)\,, \qquad \theta(0) = \theta_0\,, \tag{11}$$

where we defined

$$\begin{aligned} M(\theta) &= \int_{\mathcal{X}} \nabla_\theta U(\theta, \boldsymbol{x}) \otimes \nabla_\theta U(\theta, \boldsymbol{x}) d\nu(\boldsymbol{x})\,, \\ F(t, \theta) &= \int_{\mathcal{X}} \nabla_\theta U(\theta, \boldsymbol{x}) f(t, \boldsymbol{x}, U(\theta)) d\nu(\boldsymbol{x})\,, \end{aligned} \tag{12}$$

in which $\otimes$ denotes the outer product. Following the same principle, analogous equations as given by (12) have been derived in other settings of nonlinear parametrizations as well; see, e.g., the works on dynamic low-rank approximations and related methods [23, 43] and the works [7, 11, 1].

The system of ODEs defined in (11) for $\theta(t)$ is the basis for the Neural Galerkin schemes we introduce below. We refer to $U(\theta(t))$ as Neural Galerkin solution because Eq. (11) can be derived via testing the residual $r_t(\theta, \dot{\theta}, \boldsymbol{x})$ in Eq. (6) using $\nabla_\theta U(\theta, x)$ as test function and Galerkin projection with respect to the inner product of $L^2(\mathcal{X}; \nu)$. The objective function (8) also arises if we take the inner product between $r_t(\theta, \dot{\theta}, \boldsymbol{x})$ and a test function $g(\boldsymbol{x})$ and maximize it over all $g(\boldsymbol{x})$ with norm one in $L^2(\mathcal{X}; \nu_\theta)$.

3.4. Discretization in time

Eq. (11) is a system ODEs that can be numerically integrated using standard methods; thus training the network parameters becomes solving a system of ODEs in time. Let us denote by θ^k the numerical approximation to $\theta(t_k)$ where the times t_k with $k = 0, 1, 2, \ldots$ are defined recursively via $t_0 = 0$, $t_{k+1} = t_k + \delta t_k$ using the time steps $\delta t_k > 0$, possibly nonuniform and chosen adaptively. To update θ^k, we can either use explicit or implicit integrators.

3.4.1. Explicit integrators.

The forward Euler scheme is an explicit scheme that leads to

$$M(\theta^k)\theta^{k+1} = M(\theta^k)\theta^k + \delta t_k F(t_k, \theta^k)\,. \tag{13}$$

This is an explicit equation for θ^{k+1} which is solved at every time step—to this end it may be useful to add a regularizing term $\lambda\,\mathrm{Id}$ to M, where Id is the identity. Other integrators can be used as well, such as the adaptive Runge–Kutta–Fehlberg (RK45) method [10], which leads to linear systems similar to (13) at every stage—RK45 will be used in the examples below.

3.4.2. Implicit integrators.

We consider the backward Euler scheme as an example for implicit integrators, which leads to the system of equations

$$M(\theta^{k+1})\theta^{k+1} = M(\theta^{k+1})\theta^k + \delta t_k F(t_{k+1}, \theta^{k+1})\,. \tag{14}$$

Implicit integrators are more costly per time step since they lead to nonlinear equations such as (14) for θ^{k+1} that have to be solved at each time step.

We stress that the adaptive nature of our Neural Galerkin approach is different from time-space collocation methods based on neural networks [9, 44, 36] and other DNN-based PDE solution methods that are formulated over equidistant time steps [17].

Instead of deriving the variational formulation (7) in continuous time and then discretizing the Euler-Lagrange equations (10), one could also first discretize in time and then derive the Euler-Lagrange equation of the discrete-in-time optimization problem; this is similar to the distinction between discretize-then-optimize versus optimize-then-discretize approaches used e.g. in inverse problems and PDE-constrained optimization: see e.g. Ref. [16].

4. Active learning with Neural Galerkin schemes

We now introduce active learning for Neural Galerkin schemes to efficiently estimate $M(\theta)$ and $F(t, \theta)$. To achieve this, we make the measure in the objective (8) and the definition of M and F depend on the parameter $\theta(t)$ so that we can adapt the measure $\nu_{\theta(t)}$ over time; see Section 4.1. We then formulate adaptive Monte Carlo estimators of M and F with the aim of achieving lower mean-squared errors than regular Monte Carlo estimations that are based on un-informed, e.g., uniform, sampling. In Section 4.2, we propose to directly use a time-adaptive measure and in Section 4.3 we show that Neural Galerkin schemes can be combined with importance sampling. Section 4.4 specifically considers shallow networks with Gaussian units that gives rise to an adaptive sampling scheme that leverages the network architecture.

4.1. Active learning via time-dependent measures

For generic choices of parametrization $U(\theta)$, the integrals in Eq. (12) do not admit a closed-form solution and so will need to be numerically estimated. In low dimensions, this calculation can be performed by quadrature on a grid. When $\mathcal{X}$ is high dimensional, however, quadrature becomes computationally intractable quickly and we must proceed differently. If the measure ν against which is integrated in Eq. (12) is a probability measure, we can consider using a vanilla Monte Carlo estimator for each term, by drawing n samples $\{\boldsymbol{x}_i\}_{i=1}^n$ from ν and replacing the expectations by empirical averages over these samples. This estimator is efficient to approximate certain kernels uniformly over high-dimensional spaces [35], but not necessarily if the solution to the PDE (1) develops spatially localized structures, as is the case in many physical systems of interest.

Instead, we consider the objective J_t defined in (8) and reformulate it with a parameter-dependent measure ν_θ as

$$J_\theta(\theta, \eta) = \frac{1}{2} \int_{\mathcal{X}} |r_t(\theta, \eta, \boldsymbol{x})|^2 \mathrm{d}\nu_\theta(\boldsymbol{x}) \,. \tag{15}$$

The key difference to J_t used in Section 3.2, is that now the objective defined in (15) integrates the squared residual against a measure ν_θ that depends on the parameter $\theta(t)$ and thus it depends on time t. Following an analogous derivation as in Section 3.3, we obtain the operators

$$\begin{aligned} M(\theta) &= \int_{\mathcal{X}} \nabla_\theta U(\theta, \boldsymbol{x}) \otimes \nabla_\theta U(\theta, \boldsymbol{x}) d\nu_\theta(\boldsymbol{x}) \\ F(t, \theta) &= \int_{\mathcal{X}} \nabla_\theta U(\theta, \boldsymbol{x}) f(t, \boldsymbol{x}, U(\theta)) d\nu_\theta(\boldsymbol{x}) \,, \end{aligned} \tag{16}$$

with measure ν_θ depending on θ. This new formulation with measure ν_θ allows us to adapt the measure so that Monte Carlo estimators of M and F have a low mean-squared error.

4.2. Monte Carlo estimators with adaptive samples

Let $\{\boldsymbol{x}_i(t)\}_{i=1}^n$ be n samples of $\nu_{\theta(t)}$ and notice that the samples $\boldsymbol{x}_1(t), \ldots, \boldsymbol{x}_n(t)$ depend on time t. Monte Carlo estimators $\tilde{M}$ and $\tilde{F}$ of M and F, respectively, are then given by

$$\begin{aligned} \tilde{M}(\theta(t)) =& \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta U(\theta(t), \boldsymbol{x}_i(t)) \otimes \nabla_\theta U(\theta(t), \boldsymbol{x}_i(t)), \\ \tilde{F}(t, \theta(t)) =& \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta U(\theta(t), \boldsymbol{x}_i(t)) f(t, \boldsymbol{x}_i(t), U(\theta(t)))\,. \end{aligned} \tag{17}$$

For these estimators to be effective, their variances must be small, which requires choosing the measure $\nu_{\theta(t)}$ in a way that is informed by the solution $U(\theta(t))$. This choice must be made on a case-by-case basis. For example, below we consider Fokker-Planck equations whose solutions are normalized probability density functions: in this case, it is reasonable to take $\nu_\theta = U(\theta)$—this choice leads to low variance estimators if the density has mass localized in a few relatively small regions that move in $\mathcal{X}$. If in addition the parametrization $U(\theta)$ involves Gaussian units, as in the Fokker-Planck examples below, sampling from it is straightforward—with this specific choice we can can in fact evaluate analytically the integrals of the components of $M(\theta)$ and sampling is only required to estimate $F(t, \theta)$. Other choices for ν_θ are possible too, e.g. by taking $\nu_{\theta(t)}$ proportional to $|\nabla_{\boldsymbol{x}} U(\theta(t), \boldsymbol{x})|$ in situations where the solution has localized fronts that propagate in $\mathcal{X}$.

4.3. Importance sampling with respect to a nominal measure

Consider a nominal measure ν. Then, an adaptive measure ν_θ can be realized via importance sampling by defining positive weights $\omega : \mathcal{X} \times \Theta \to (0, \infty)$ such that $Z_\theta = \int_{\mathcal{X}} \omega(\boldsymbol{x}, \theta) \mathrm{d}\nu(\boldsymbol{x}) < \infty$ for all $\theta \in \Theta$. Notice that the weights given by ω depend on the parameter $\theta(t)$ and thus on time t. One then draws a set of samples $\{\boldsymbol{x}_i(t)\}_{i=1}^n$ from $d\nu_{\theta(t)}(\boldsymbol{x}) = Z_\theta^{-1} \omega(\boldsymbol{x}, \theta(t)) d\nu(\boldsymbol{x})$ and uses the reweighted estimators $\tilde{M}$ and $\tilde{F}$ given by

$$\begin{aligned} \tilde{M}(\theta) =& \frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_\theta U(\theta, \boldsymbol{x}_i) \otimes \nabla_\theta U(\theta, \boldsymbol{x}_i)}{\omega(\boldsymbol{x}_i, \theta)}, \\ \tilde{F}(t, \theta) =& \frac{1}{n}\sum_{i=1}^{n} \frac{\nabla_\theta U(\theta, \boldsymbol{x}_i) f(t, \boldsymbol{x}_i, U(\theta))}{\omega(\boldsymbol{x}_i, \theta)}\,. \end{aligned} \tag{18}$$

4.4. Neural architectures and sampling

Our method offers the possibility to use DNNs of arbitrary complexity and sophistication for the parametrization $U(\theta, \boldsymbol{x})$, in a way that can be tailored to known properties or symmetries of the solution of Eq. (1), as well as its boundary conditions. However, the network architecture can also be chosen such that it aligns well with the sampling from the adaptive measure $\nu_{\theta(t)}$. In later numerical experiments, we will consider shallow (one-hidden-layer) network with m nodes given by

$$U(\theta, \boldsymbol{x}) = \sum\nolimits_{i=1}^{m} c_i \varphi(\boldsymbol{x}, w_i, \boldsymbol{b}_i)\,, \tag{19}$$

where θ is defined as $\theta = (c_i, w_i, \boldsymbol{b}_i)_{i=1}^m$ and $c_i, w_i \in \mathbb{R}$ and $\boldsymbol{b}_i \in \mathbb{R}^d$ are the parameters of the DNN, and $\varphi : \mathcal{X} \times \mathbb{R}^{d+1} \to \mathbb{R}$ is the following nonlinear unit (activation function) given by the Gaussian kernel

$$\varphi_{\mathrm{G}}(\boldsymbol{x}, w, \boldsymbol{b}) = \exp\left(-w^2 |\boldsymbol{x} - \boldsymbol{b}|^2\right)\,. \tag{20}$$

Here and below the exponential function acts componentwise, and we refer to the c_i as the coefficients of the network and to the $w_i, \boldsymbol{b}_i$ as its features.

The network architecture (19) is especially useful for approximating solutions that are probability density functions, such as for the Fokker-Planck equation considered below. In such a case, as discussed in Section 4.2, it is appropriate to set $\nu_{\theta(t)} = U(\theta(t))$. Then, the set of samples $\{(\boldsymbol{x}_i(t)\}_{i=1}^n$ at time t is obtained via sampling from $U(\theta)$ directly, which can be done efficiently by exploiting that the units (20) are Gaussians.

5. Numerical experiments

We demonstrate Neural Galerkin schemes with active learning on several numerical examples. We first discuss network architectures in Section 5.1 and then consider numerical examples reaching from low-dimensional benchmark problems in Section 5.2 and Section 5.3 to PDEs whose solutions develop local features in moderately high-dimensional spatial domains in Section 5.4 and Section 5.5.

5.1. Network architectures

We use two specific neural architectures. The first is a shallow (one-hidden-layer) network with m nodes given by (19). The unit φ is one of the following two nonlinear units (activation function): The first is the Gaussian kernel defined in (20) and the second one is

$$\varphi_{\mathrm{G}}^L(\boldsymbol{x}, w, \boldsymbol{b}) = \exp\left(-w^2 |\sin\left(\pi(\boldsymbol{x} - \boldsymbol{b})/L\right)|^2\right) , \tag{21}$$

which we use when $\mathcal{X} = L\mathbb{T}^d$ with $L > 0$ and we need to enforce periodicity. The sine function acts componentwise on its argument. We refer to the c_i as the coefficients of the network and to the $w_i, \boldsymbol{b}_i$ as its features.

The other neural architecture that we use is a feedforward neural network with $\ell \in \mathbb{N}$ hidden layers and m nodes per layer:

$$U(\theta, \boldsymbol{x}) = \boldsymbol{c}(t)^T \tanh(\boldsymbol{W}_\ell \tanh(\boldsymbol{W}_{\ell-1}(\cdots \varphi_{\tanh}^L(\boldsymbol{x}, \boldsymbol{W}_1, \boldsymbol{b}_1)\cdots) + \boldsymbol{b}_{\ell-1}) + \boldsymbol{b}_\ell), \tag{22}$$

where $\theta = (\boldsymbol{c}, \boldsymbol{W}_1, \ldots, \boldsymbol{W}_\ell, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_\ell)$ with $\boldsymbol{c} \in \mathbb{R}^m$, $\boldsymbol{W}_1 \in \mathbb{R}^{m\times d}, \boldsymbol{W}_i \in \mathbb{R}^{m\times m}, i > 1$, and $\boldsymbol{b}_i \in \mathbb{R}^d$ are the parameters, and $\varphi_{\tanh}^L$ is the nonlinear unit

$$\varphi_{\tanh}^L(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = \tanh\left(\boldsymbol{W} \sin\left(2\pi\left(\boldsymbol{x} - \boldsymbol{b}\right)/L\right)\right) . \tag{23}$$

The tanh function acts compontwise on its argument.

5.2. Korteweg-de Vries equation

Consider the KdV equation $\partial_t u + \partial_x^3 u + 6u\partial_x u = 0$ on the one-dimensional spatial domain $\mathcal{X} = [-20, 40) \subset \mathbb{R}$ with periodic boundary condition. We follow the setup described in Ref. [46], which for the initial conditions described in their Section I-a-(ii) leads to a solution of the KdV equation that consists of two interacting solitons that approach each other, collide, and then separate—this solution, shown in Figure 1(a), is available analytically, thereby providing us with a benchmark of the numerical results.

For the parametric solution, we use the shallow neural network defined in Eq. (19) with the Gaussian units defined in Eq. (21) and $m = 10$ nodes. Because the dimension of the spatial domain is one in this example, we simply take $d\nu_\theta(x) = dx$, and sample $n = 1000$ points uniformily in $[0, 1]$ to estimate M and F. The initial parameter θ_0 is obtained via a least-squares fit of the initial condition. The batch size is 10^5 and number of iterations is 10^5. Samples are drawn uniformly in the spatial domain and the learning rate is 10^{-1}. We take five replicates with randomized initialization of parameters and then use the fit with lowest error on test samples. As integrator, we use RK45 so that the time-step size is adaptively chosen based on the dynamics of the solution.

The time-space plot of the Neural Galerkin approximation shown in Figure 1(b) is in close agreement with the analytic solution shown in Figure 1(a). The relative ℓ_2 error remains low over time, as shown in

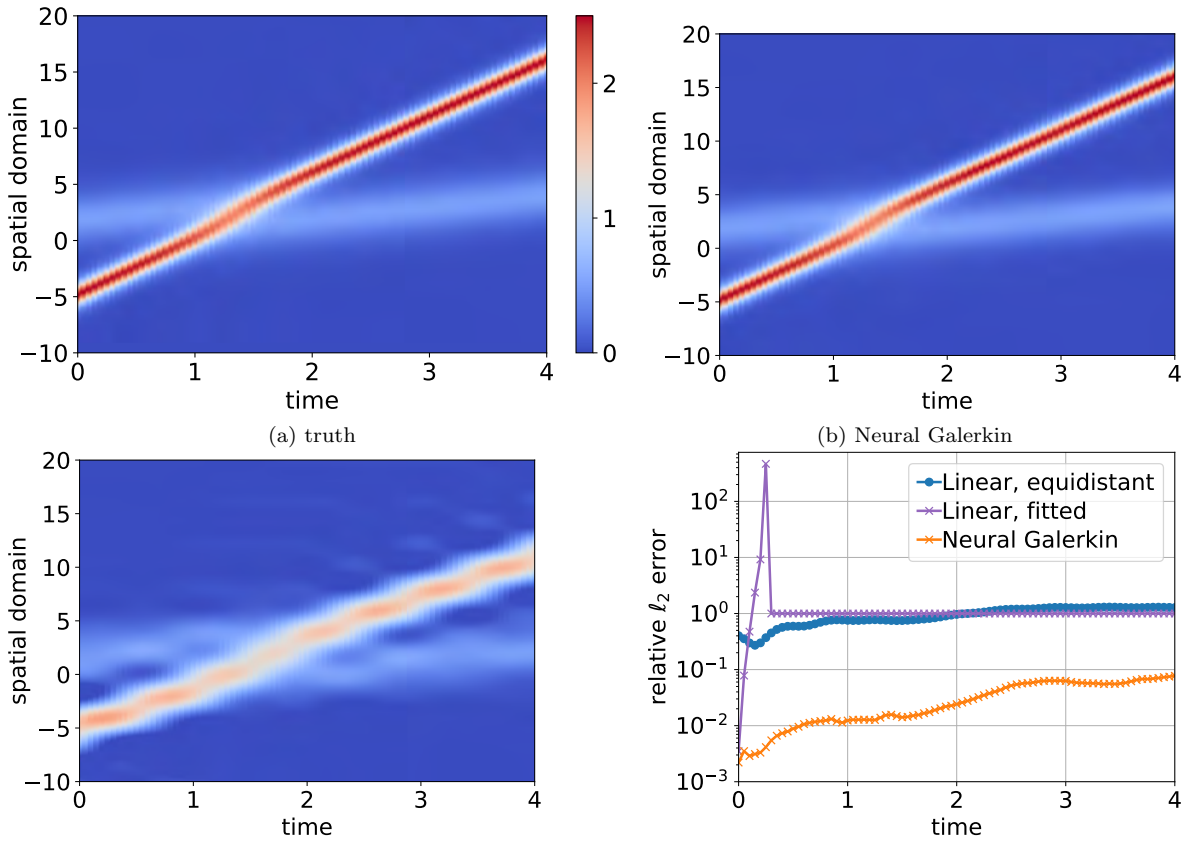


Figure 1: Korteweg-De Vries: Neural Galerkin schemes integrate in time a nonlinear parametrization of the solution and so obtain an accurate approximation with few degrees of freedom in this example. In contrast, linear (standard) Galerkin that derives approximations in fixed spaces with fixed bases without feature adaptation leads to poor approximations when there are local dynamics in the spatial domain.

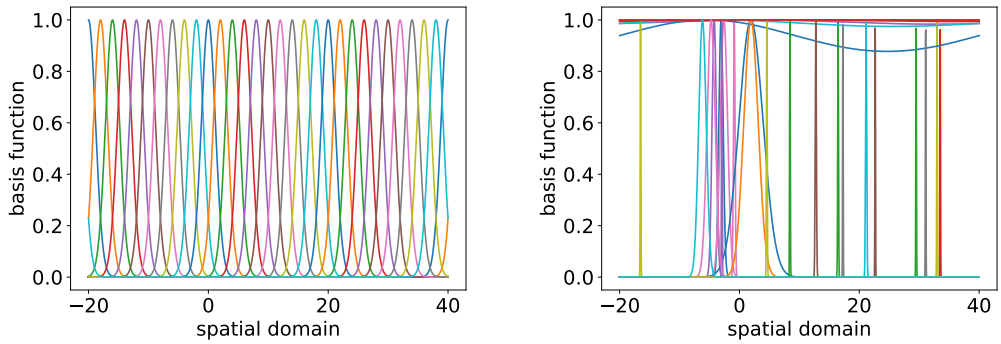
Figure 1(d). The relative ℓ_2 error is computed as follows: Consider the $w = 2048$ equidistant grid points $x_1, \ldots, x_w$ in $\mathcal{X}$ and define

$$\boldsymbol{u}(t) = [u(t, x_1), \ldots, u(t, x_w)]^T \in \mathbb{R}^w \,,$$

to be the vector of the analytic solution u at time t and at the grid points $x_1, \ldots, x_w$. Similarly, define $\tilde{\boldsymbol{u}}(t) = [\tilde{u}(t, x_1), \ldots, \tilde{u}(t, x_w)]^T$ as the solution vector corresponding to an approximation $\tilde{u}$. Then, the reported relative ℓ_2 error is

$$e_{\ell_2} = \frac{\sum_{k=0}^{K} \|\boldsymbol{u}(t_k) - \tilde{\boldsymbol{u}}(t_k)\|_2^2}{\sum_{k=0}^{K} \|\boldsymbol{u}(t_k)\|_2^2} \,.$$

To emphasize the importance of the nonlinear feature adaptation, we also consider linear approximations with network Eq. (19) with $m = 30$ nodes but with fixed features w_i, and $\boldsymbol{b}_i$ that are either located equidistantly in $\mathcal{X}$ or fitted to the initial condition and then kept fixed. For linear Galerkin with equidistant basis functions, the basis functions are Gaussians units φ_G^L with a fixed bandwidth and equidistantly located in the domain Ω, see Figure 2a. For the comparison with linear Galerkin with basis functions fitted to the initial condition, we use the units fitted to the initial condition as described above and keep the features fixed, see Figure 2b. In both cases, the linear approximation has the same number of degrees of freedom as the nonlinear Neural Galerkin approximation.



(a) KdV, equidistant basis functions

(b) KdV, fitted basis functions

Figure 2: KdV: Basis functions used for linear Galerkin approximations.

The linear approximation leads to large errors that develop after only a few time steps, as apparent from the solution shown in Figure 1(c). This is also in agreement with Figure 1(d) which shows that the linear Galerkin approximations based on fixed features lead to errors that are orders of magnitude larger than the Neural Galerkin approximation with the same number of degrees of freedom.

5.3. Allen-Cahn (AC) equation

Consider the prototypical reaction diffusion equation known as the AC equation

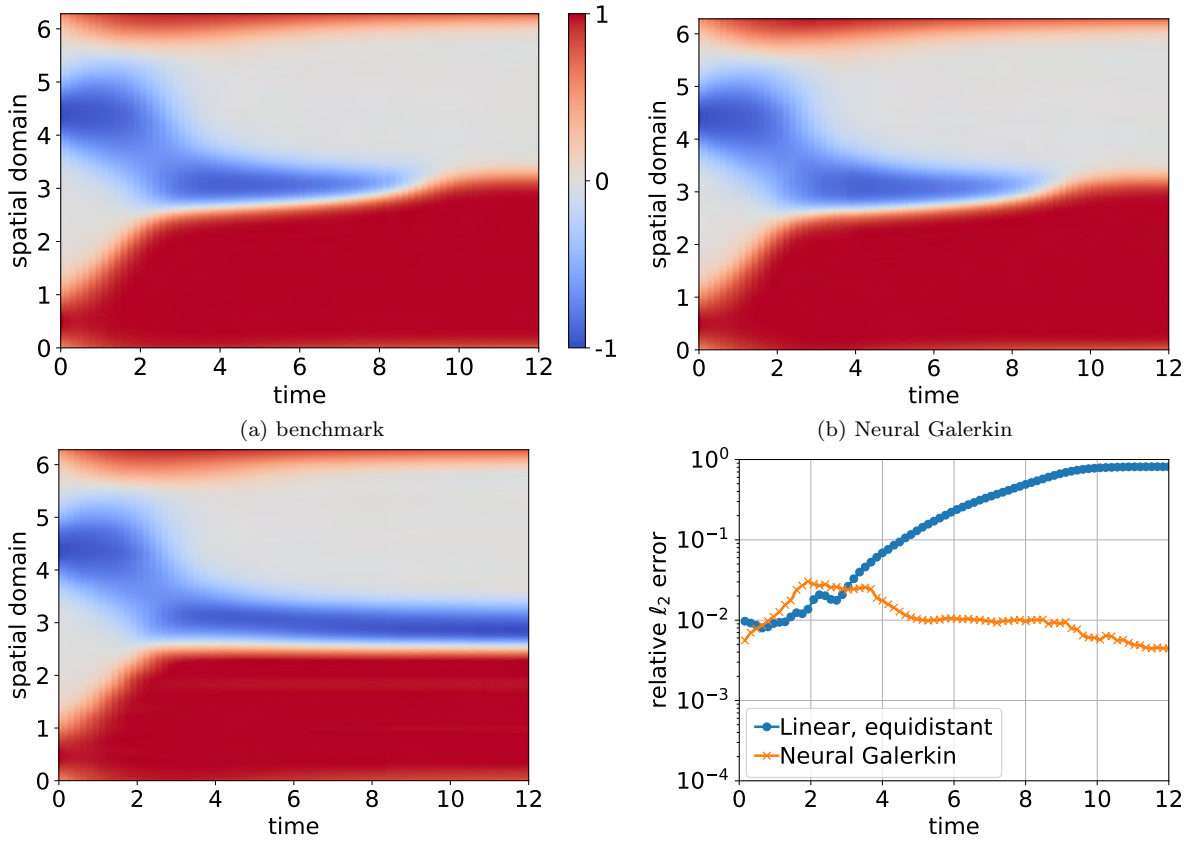
$$\partial_t u = \epsilon \partial_x^2 u - a(t,x)(u - u^3)$$

in the one-dimensional domain $\mathcal{X} = [0, 2\pi)$ with periodic boundary conditions. We set $\epsilon = 5 \times 10^{-2}$ in the diffusion term and let the coefficient in the reaction term vary in time and space using $a(t,x) = 1.05 + t\sin(x)$. This coefficient drives u towards $u = \pm 1$ for $t < 1.05$, wheres for $t > 1.05$ the solution u is driven towards ± 1 for x in the region where $\sin(x) < 1.05/t$ and 0 in the region where $\sin(x) > 1.05/t$; as $t \to \infty$ these two regions converge to $(0, \pi)$ and $(\pi, 2\pi)$, respectively. The initial condition is

$$u(0,x) = \varphi_{\mathrm{G}}^L(x, \sqrt{10}, 1/2) - \varphi_{\mathrm{G}}^L(x, \sqrt{10}, 4.4)\,,$$

with φ_{G}^L defined in Eq. (21) and $L = 1/2$.

Figure 3(a) shows a benchmark solution computed with a grid-based method. The benchmark solution is computed with a finite-difference discretization on 2048 equidistant grid points in the spatial domain. Time is discretized with semi-implicit Euler where the linear operators are treated implicitly and the nonlinear operators are treated explicitly. The time-step size is 10^{-5}. The solution consists of relatively flat pieces separated by sharp walls, which evolve in a way that is consistent with the evolving sign structure of the coefficient $a(t,x)$ described above. Figure 3(b) shows the Neural Galerkin approximation obtained with the three-hidden-layer network defined in (22), with $m = 2$ tanh units (16 degrees of freedom) and fixed uniform measure $d\nu_\theta(x) = dx$. The initial condition for the Neural Galerkin scheme is fitted analogously to fitting of the initial condition in the experiment with the KdV equation. Neural Galerkin uses a backward Euler discretization in time and then takes the gradient with automatic differentiation implemented in JAX (`https://github.com/google/jax`). The time-step size is $\delta t = 10^{-2}$. The nonlinear system is solved with ADAM and 10,000 iterations. The corresponding optimization problem in each time step is approximately solved with SGD and $n = 1000$ samples to estimate M and F. As can be seen, the Neural Galerkin solution agrees well with the benchmark solution. In contrast, the approximation obtained with linear Galerkin with 16 equidistantly located Gaussian basis functions defined in (21) and shown in Figure 3(c) leads to



(a) benchmark

(b) Neural Galerkin

(c) Linear Galerkin, equidistant

(d) error over time

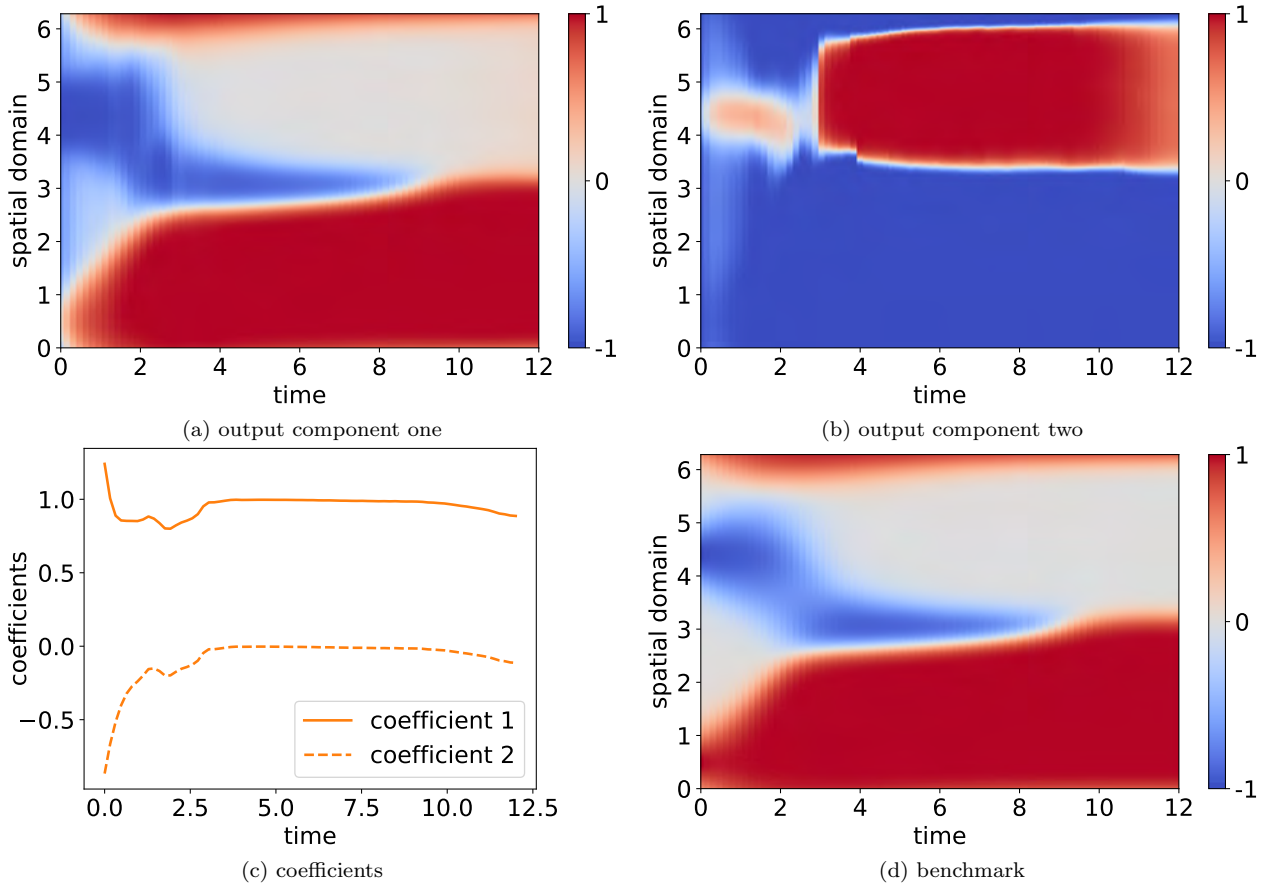
Figure 3: Allen-Cahn: The proposed Neural Galerkin approach with a three-hidden-layer network correctly predicts the dynamics of the sharp walls that eventually separate the solution into two flat pieces at 0 and +1 in this experiment. In contrast, linear Galerkin with the same number of degrees of freedom wrongly predicts a solution with three flat pieces at $-1, 0$, and $+1$ because of the lack of expressiveness of the linear approximation.

a poorer approximation. This is further quantified in Figure 3(d) which shows that the Neural Galerkin approximation with a three-hidden-layer network achieves an ℓ_2 error that is two orders of magnitude lower than that of the linear Galerkin approximation.

Figure 4 visualizes the adaptation of the coefficients and features of the Neural Galerkin approximation. Consider the network (22) with $\boldsymbol{c}(t) = (c_1(t), c_2(t))^T \in \mathbb{R}^2$, $\ell = 3$, and set

$$\psi_i(t, \boldsymbol{x}) \quad = \quad \boldsymbol{e}_i^T \tanh(\boldsymbol{W}_3(t) \tanh(\boldsymbol{W}_{2-1}(t)(\varphi^L_{\tanh}(\boldsymbol{x}, \boldsymbol{W}_1(t), \boldsymbol{b}_1(t))) \quad + \quad \boldsymbol{b}_2(t)) \quad + \quad \boldsymbol{b}_3(t)), \quad (24)$$

where $\boldsymbol{e}_i \in \mathbb{R}^2$ with $i = 1, 2$ are the two canonical unit vectors. Thus the output components $\psi_1(t, \boldsymbol{x})$ and $\psi_2(t, \boldsymbol{x})$ are weighted by the coefficients $c_1(t)$ and $c_2(t)$ to obtain the Neural Galerkin approximation; these two components are shown in Figure 4(a) and (b). The corresponding coefficients are visualized in panel (c) and the benchmark solution is plotted again in panel (d). Even though the Neural Galerkin approximation uses only two $\psi_i(t, \boldsymbol{x})$, because these functions depends nonlinearly on parameters that evolve in time, they adapt their shape to accurately approximate the solution.



(a) output component one

(b) output component two

(c) coefficients

(d) benchmark

Figure 4: Allen-Cahn: The plots visualize how the used Neural Galerkin scheme propagates forward in time the coefficients and features of a DNN parametrization by following the dynamics prescribed by the PDE. Notice that the output components, defined in Eq. (24) and shown in (a) and (b), reflect the transition of the sharp walls between the flat pieces of the solution and that the dynamics rapidly change from time $t = 1.05$ to about $t = 2.5$ when the potential changes sign in parts of the spatial domain.

5.4. Advection in unbounded, high-dimensional domains

Our next example is an advection equation in high dimension,

$$\partial_t u + \boldsymbol{a}(t, \boldsymbol{x}) \cdot \nabla_{\boldsymbol{x}} u = 0, \qquad u(0, \boldsymbol{x}) = u_0(\boldsymbol{x}), \tag{25}$$

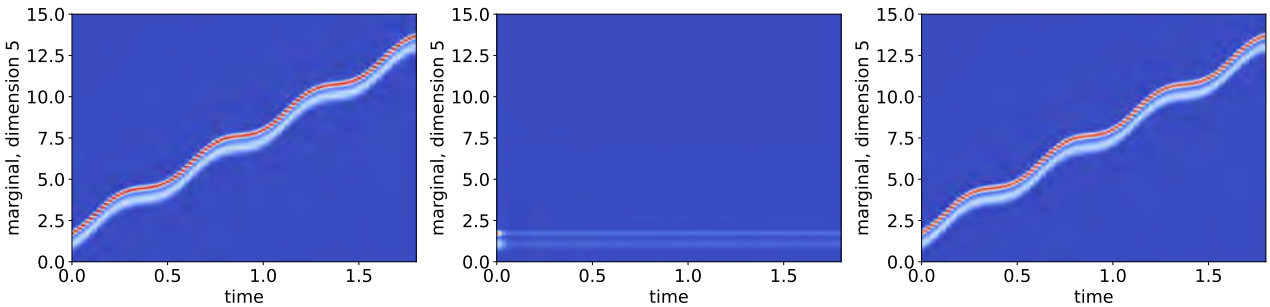
where $\boldsymbol{x} \in \mathcal{X} \equiv \mathbb{R}^d$, $\boldsymbol{a} : [0, \infty) \times \mathbb{R}^d \to \mathbb{R}$ is some bounded velocity field that is assumed to be differentiable in both its arguments, and $u_0 : \mathbb{R}^d \to \mathbb{R}$ is some initial condition satisfying $\lim_{|\boldsymbol{x}| \to \infty} u_0(\boldsymbol{x}) = 0$. This equation can in principle be solved by the method of characteristics, i.e. by considering

$$\dot{\boldsymbol{X}}(t, \boldsymbol{x}) = \boldsymbol{a}(t, \boldsymbol{X}(t, \boldsymbol{x})), \qquad \boldsymbol{X}(0, \boldsymbol{x}) = \boldsymbol{x}, \tag{26}$$

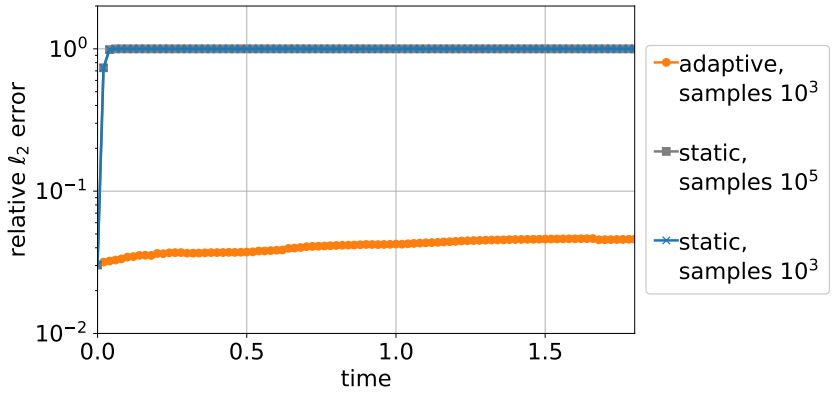
and using

$$u(t, \boldsymbol{x}) = u_0(\boldsymbol{X}(-t, \boldsymbol{x})). \tag{27}$$

In practice, however, Eq. (27) is hard to use to obtain global information about the solution, except in situations where Eq. (26) can be solved explicitly (an instance of which we will also consider below as benchmark). Next we show how to numerically solve (25) directly using a Neural Galerkin scheme.



(a) truth (b) static (c) Neural Galerkin



(d) error over time

Figure 5: High-dimensional advection with time-only varying advection speed: (a) marginal in dimension 5 of the analytic solution; (b) approximation obtained with static sampling; (c) result of the Neural Galerkin method that adaptively samples data over time to estimate the operators M and F depending on the dynamics of the problem at hand. This is in stark contrast to classical time-space collocation approaches that uniformly sample over space and time when optimizing for a solution. In this example, the adaptive sampling is key for Neural Galerkin to accurately predict the local-in-space dynamics of the solution. In contrast, an approximate solution obtained with uniform sampling that is static over time fails to lead to meaningful predictions, as shown by plot (b) and the error shown in plot (d).

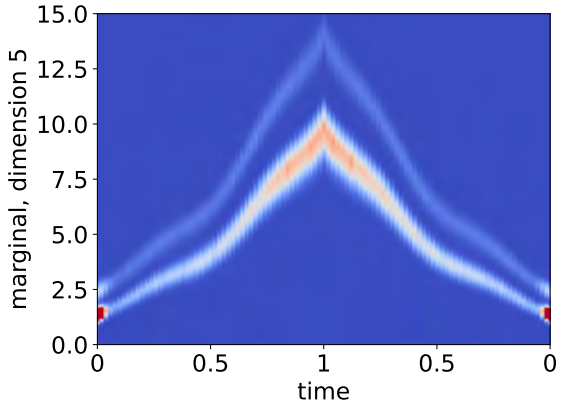
5.4.1. Advection with a time-dependent coefficient.

In the first experiment with the advection equation, we make the transport coefficient depend only on time using

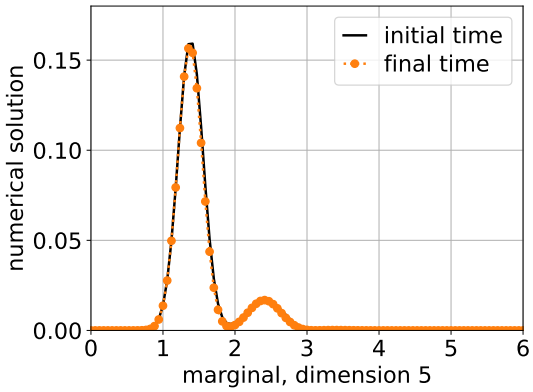
$$\boldsymbol{a}_t(t) = \boldsymbol{a}_s \odot (\sin(\boldsymbol{a}_v \pi t) + 5/4) \tag{28}$$

with $\boldsymbol{a}_s = [1, 2, \ldots, d]^T$, $\boldsymbol{a}_v = 2 + \frac{2}{d}[0, 1, \ldots, d-1]^T$ and the element-wise vector multiplication $\odot$. We set $d = 5$ and take as initial condition u_0 a mixture of two non-isotropic Gaussian packets with means

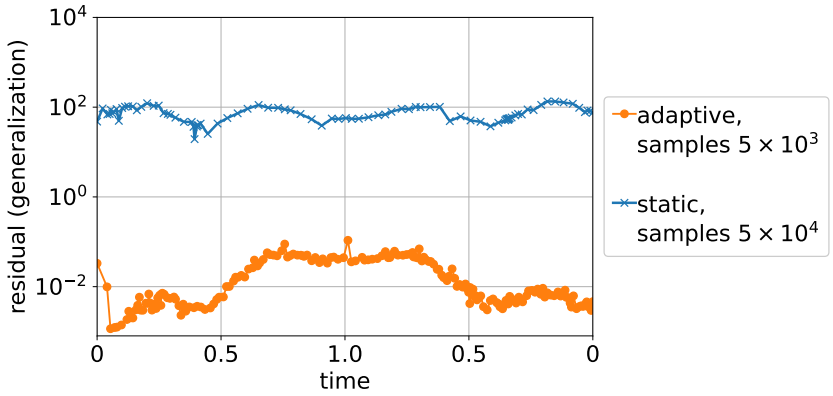
$$\boldsymbol{\mu}_1 = \frac{11}{10}\begin{bmatrix}1\\1\\\vdots\\1\end{bmatrix}, \qquad \boldsymbol{\mu}_2 = \frac{3}{4}\begin{bmatrix}1.5 - (-1)^1 1/(d+1)\\1.5 - (-1)^2 2/(d+1)\\\vdots\\1.5 - (-1)^d d/(d+1)\end{bmatrix}$$



(a) Neural Galerkin with adaptive measure



(b) initial vs. final solution



(c) residual

Figure 6: High-dimensional advection with time-space varying advection speed: The equation is integrated forward in time until $t = 1$ and then integrated backwards until time $t = 0$, where the initial condition is reached again. Plot (a) shows the prediction of the Neural Galerkin method. Plot (b) shows that the Neural Galerkin solution at final time closely matches the initial condition, which means that little error is accumulated by integrating forward and then backward in time. Adaptive sampling is key in this example because the solution is local in the high-dimensional spatial domain. Neural Galerkin with adaptive sampling tracks the local dynamics well, whereas approximations based on static sampling fail after only a few time steps as shown by a large residual in (c).

and covariance matrices

$$\boldsymbol{\Sigma}_1 = \frac{1}{200}\begin{bmatrix} 2 & & \\ & 4 & \\ & & \ddots \\ & & & 2d \end{bmatrix}, \qquad \boldsymbol{\Sigma}_2 = \frac{1}{200}\begin{bmatrix} d & & \\ & d-1 & \\ & & \ddots \\ & & & 1 \end{bmatrix}. \tag{29}$$

In this case, the solution (27) can be derived explicitly,

$$u(t, \boldsymbol{x}) = u_0\left(\boldsymbol{x} - \textstyle\int_0^t \boldsymbol{a}_t(s)ds\right), \tag{30}$$

where $\boldsymbol{a}_t$ is the time-varying coefficient defined in the main text. We will use the exact solution as benchmark. Marginals of the solutions are computed as follows: Let $\mathcal{X} = \mathcal{X}_1 \times \cdots \times \mathcal{X}_d$ be the domain of interest. We

then plot for each dimension $i = 1, \dots, d$ the function

$$(t, x_i) \mapsto \int_{\Omega_1} \cdots \int_{\Omega_{i-1}} \int_{\Omega_{i+1}} \cdots \int_{\Omega_d} u(t, x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_d)\,,$$

where u is the function of which marginals are to be plotted and the integrals are numerically approximated via Monte Carlo and 8192 samples. The samples are drawn from the analytic solution, which is available in this experiment. The marginal in dimension five is shown in Figure 5(b); see also Figure 11 in the Appendix.

For Neural Galerkin, see Figure 5(b), we use a shallow network as in Eq. (19) with the exponential unit (20)—notice that since these units are isotropic, several of them are required to accurately approximate both the initial condition and the solution of Eq. (25) at time $t > 0$. We take $m = 50$ nodes and use the adaptive RK45 method as time-integrator. We also use a measure ν_θ adapted to the solution to sample $n = 1000$ data points $\{\boldsymbol{x}_i\}_{i=1}^n$ at each integration step to estimate M and F via Eq. (17). Specifically, we take ν_θ to be the Gaussian mixture with 50 nodes obtained from the current neural approximation of the solution by equating the weights c_i, dividing w_i by a factor $\kappa > 0$, and keeping the same $\boldsymbol{b}_i$, i.e. if the current Neural Galerkin approximation of the solution is

$$U(\theta, \boldsymbol{x}) = \sum\nolimits_{i=1}^{50} c_i(t) \exp\left(-w_i^2(t)|\boldsymbol{x} - \boldsymbol{b}_i(t)|^2\right)\,, \tag{31}$$

we sample the data points from

$$d\nu_{\theta(t)}(\boldsymbol{x}) = C^{-1} \sum\nolimits_{i=1}^{50} \exp\left(-\tfrac{1}{\kappa^2} w_i^2(t)|\boldsymbol{x} - \boldsymbol{b}_i(t)|^2\right) d\boldsymbol{x}\,, \tag{32}$$

where C is a normalization constant and $\kappa = 1$ in this experiment; see Section 4.4. Using this adapted measure is key for accuracy, and leads to the small ℓ_2 error plotted in Figure 5(d). If instead we estimate M and F by drawing the data points uniformly in $[0, 15]^d$ (to cover the domain over which the solution propagates for $t \in [0, 2]$), the relative error is 100% after only a few time steps even if we use as many as $n = 10^5$ data points, see Figure 5(b) and Figure 5(d). This emphasizes the importance of using adaptivity in both the function approximation via neural networks, and the data acquisition to estimate M and F. This double-adaptivity is a key distinguishing feature of the proposed Neural Galerkin schemes compared to existing DNN approaches.

5.4.2. Advection with time- and space-dependent coefficients.

Next we consider Eq. (25) with the time-space varying advection speed,

$$\boldsymbol{a}_{st}(t, \boldsymbol{x}) = \boldsymbol{a}_s \odot (\sin(\boldsymbol{a}_v \pi t) + 3) \odot (\boldsymbol{x} + 1)/10\,, \tag{33}$$

where $\boldsymbol{a}_s, \boldsymbol{a}_v$ are the vectors defined in Eq. (28). For the initial condition we take a sum of two Gaussian packets, $u_0(x) = p_1(x)/10 + p_2(x)/10$. The two Gaussian probability density functions p_1, p_2 are scaled by a factor $1/10$ to avoid scaling issues. The means are

$$\boldsymbol{\mu}_1 = 2 - \frac{1}{12}\begin{bmatrix}-1 & 2 & -3 & 4 & -5\end{bmatrix}^T\,, \qquad \boldsymbol{\mu}_2 = 1.8 - \frac{1}{12}\begin{bmatrix}1 & -2 & 3 & -4 & 5\end{bmatrix}^T$$

and the covariances are

$$\boldsymbol{\Sigma}_1 = \begin{bmatrix}\frac{3}{50} & & \\ & \ddots & \\ & & \frac{3}{50}\end{bmatrix}\,, \qquad \boldsymbol{\Sigma}_2 = \begin{bmatrix}\frac{3}{100} & & \\ & \ddots & \\ & & \frac{3}{100}\end{bmatrix}\,.$$

The marginals for all five dimensions of the Neural Galerkin solutions are shown in the appendix in Figure 12.

Since the analytic solution for this problem is not available in closed form, to assess the performance of our Neural Galerkin approach, we numerically integrate the problem forward in time until time $T = 1$, then invert the advection coefficient in time, and integrate backwards until time 0: we can then estimate the error by comparing the final numerical solution with the initial condition. The numerical setup is the same as in the previous experiment with the advection speed (28), except that we draw $n = 5000$ samples from the adapted measure in (32) at each integration step to estimate M and F. Figure 6(a) shows the time-space marginal of the Neural Galerkin solution: as can be seen the spatially varying coefficient leads to a significantly changing solution over time. Figure 6(b) compares the initial condition with the final solution obtained by integrating forward and then backward in time, which are in good agreement. In Figure 6(c) we show an estimate of the PDE residual defined in Eq. (8) using the adapted measure in (32) and estimated using 10^5 samples drawn independently from this measure. The mean residual is computed as follows: We draw $n = 10^5$ samples from the equally-weighted mixture of Gaussians given by the $m = 50$ nodes of the network at time t. The standard deviation of the Gaussians is double the standard deviation of the nodes of the network. With these samples, we compute a Monte Carlo estimate of the objective J_t at time t. The residuals indicate that the proposed Neural Galerkin approach with adaptive sampling approximates well the local dynamics in this high-dimensional transport problem, whereas static sampling with a uniform distribution fails again to provide meaningful predictions.

5.5. Interacting particle systems

Let $\mathcal{X} = \mathbb{R}^d$ and consider the evolution of d interacting particles with positions $X_1(t), \ldots, X_d(t) \in \mathbb{R}$ governed by

$$\mathrm{d}X_i = g(t, X_i)\mathrm{d}t + \sum\nolimits_{j=1}^{d} K(X_i, X_j)\mathrm{d}t + \sqrt{2D}\,\mathrm{d}W_i\,, \tag{34}$$

with $i = 1, \ldots, d$ and where $g : [0, \infty) \times \mathbb{R} \to \mathbb{R}$ is a time-dependent one-body force, $K : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ a pairwise interaction term, $D > 0$ the diffusion coefficient, and W_i are independent Wiener processes. The evolution of the joint probability density of these particles, $u(t, \boldsymbol{x})$ with $\boldsymbol{x} = [x_1, \ldots, x_d]^T$, is governed by the Fokker-Planck equation

$$\partial_t u = \sum\nolimits_{i=1}^{d} \left(-\partial_{x_i}\left(u h_i(t, x_1, \ldots, x_d)\right) + D\partial_{x_i}^2 u\right), \tag{35}$$

where $h_i(t, x_1, \ldots, x_d) = g(t, x_i) + \sum_{j=1}^{d} K(x_i, x_j)$. In high dimension, one typically resorts to Monte Carlo methods based on integrating Eq. (34) to obtain samples from the density u. In contrast, we solve the Fokker-Planck equation (35) directly with our Neural Galerkin approach to derive an approximation of $u(t, \boldsymbol{x})$ via $U(\theta(t), \boldsymbol{x})$ everywhere in the domain $\mathcal{X}$. Having the density at hand allows us to efficiently compute quantities of interest that are not expressible as expectation over u, such as the entropy, and therefore not directly accessible to Monte Carlo methods. Note that Neural Galerkin schemes also involve sampling to estimate M and F, but it is done adaptively based on the current approximation to keep low the number of samples required to obtain accurate estimates.

5.5.1. Particles in harmonic trap.

In the following experiment, we set

$$g(t, x) = a(t) - x, \qquad K(x, y) = \frac{\alpha}{d}(y - x) \tag{36}$$

with $a : [0, \infty) \to \mathbb{R}$ and $\alpha > 0$. This corresponds to following the evolution of d particles put in a harmonic trap centered around the moving position $[a(t), \ldots, a(t)]^T \in \mathcal{X}$ and also attracting each other harmonically. This choice has the advantage that we can derive closed ODEs for the mean and covariance of the particle positions that can be solved numerically and used as benchmark, see E. If the initial condition $u(0, \boldsymbol{x})$ is a Gaussian probability density, the density $u(t, \boldsymbol{x})$ remains Gaussian for all times t, with the mean and covariance given by the ODEs. Accordingly, we take $u(0, \boldsymbol{x})$ to be an isotropic Gaussian density

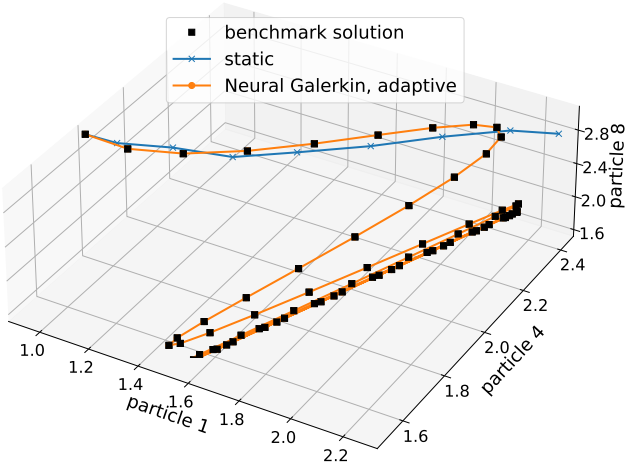


Figure 7: Particles in harmonic trap: As the particles become attracted by the trap, the particle density becomes increasingly concentrated in the high-dimensional domain. The Neural Galerkin scheme adaptively samples from the neural approximation of the density over time and thereby accurately tracks the particles mean position (orange) in contrast to methods with static sampling done uniformly in the spatial domain (blue).

with mean $9/10 + 21/(10(d-1))[0, 1, \ldots, d-1]^T$ and variance $\sigma^2 = 0.1$. We take $d = 8$ particles and set $a(t) = 5/4(\sin(\pi t) + 3/2)$, $\alpha = 1/4$ and $D = 10^{-2}$.

We apply our Neural Galerkin approach to solve the Fokker-Planck equation (35) using the neural network defined in (19) with $m = 30$ nodes and non-negative weights to guarantee that the Neural Galerkin approximation is non-negative, i.e.,

$$U(\theta, \boldsymbol{x}) = \sum_{i=1}^{m} c_i^2 \varphi(\boldsymbol{x}, w_i, \boldsymbol{b}_i)\,. \tag{37}$$

The normalized Neural Galerkin approximation is given by

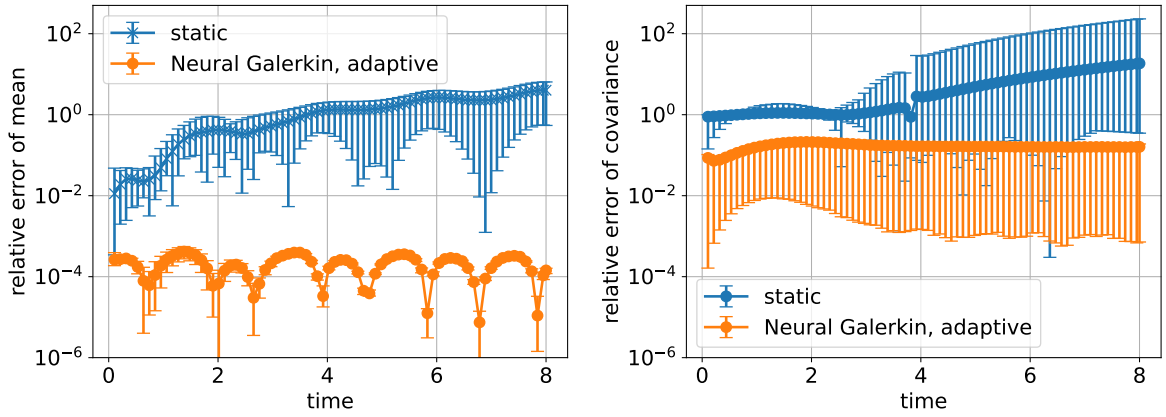
$$\bar{U}(\theta, \boldsymbol{x}) = \frac{1}{\sum_i c_i^2} \sum_{i=1}^{m} c_i^2 \varphi(\boldsymbol{x}, w_i, \boldsymbol{b}_i)\,. \tag{38}$$

The scaling factor κ in the adaptive sampling is set to $\kappa = 2$ in this experiment.

We use implicit Euler with time-step size $\delta t = 10^{-3}$. The number of SGD iterations to solve Eq. (14) in θ^{k+1} is 5×10^3. To estimate M and F, we draw $n = 1000$ samples at every time step, using the adaptive sampling procedure used in the previous experiments with the advection equation. Figure 7 shows the mean position of particles 1, 4, and 8 in the spatial domain over time. The approximation obtained with Neural Galerkin with adaptive sampling closely follows the benchmark solution. In contrast, estimation of M and F via uniform sampling in $[0, 5]^d$ leads again to a poor approximation. The relative error of the predicted mean with adaptive Neural Galerkin is roughly 10^{-4} as shown in Figure 8(a) and the entries of the covariance matrix are approximated with relative error of about 10^{-1} to 10^{-2} as shown in Figure 8(b). Because Neural Galerkin schemes provide an approximation of the density of the particles, rather than just samples of their positions as Monte Carlo methods, we can compute the entropy of the particle distribution over time; see Figure 9(a). The entropy of the distribution described by the Neural Galerkin approximation is computed by drawing $n = 5000$ samples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ from the mixture given by (37) and then estimating the entropy via Monte Carlo

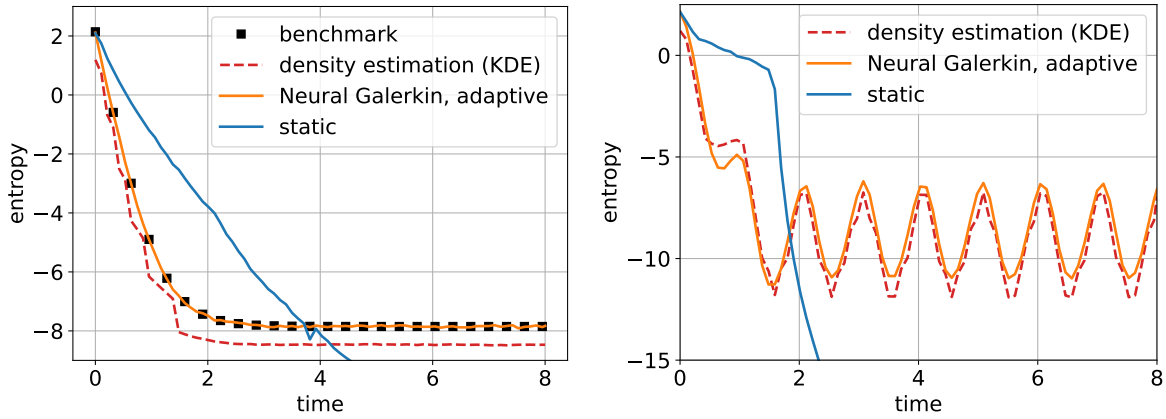
$$E_n = -\frac{1}{n} \sum_{i=1}^{n} \log \bar{U}(\theta(t), \boldsymbol{x}_i)\,.$$

The adaptive Neural Galerkin approximation of the entropy is in close agreement with the benchmark, whereas static sampling and Monte Carlo combined with density estimation from $5,000$ samples leads to poor approximations.



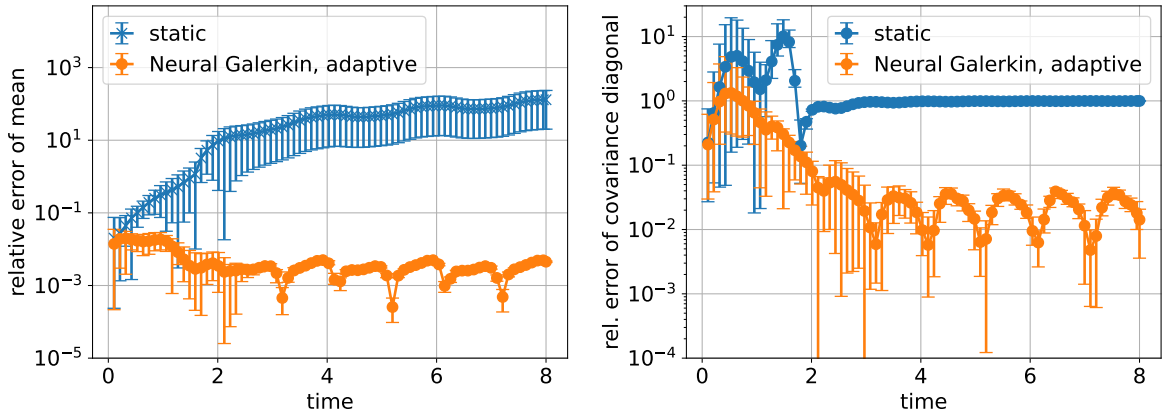
(a) mean prediction (b) covariance prediction

Figure 8: Particles in harmonic trap: Our Neural Galerkin schemes with adaptive sampling achieve orders of magnitude lower errors in predicting the mean (orange, panel a) and the covariance (orange, panel b) of the particle density than methods that rely on static sampling uniform over time and space (blue, panels a and b). Additionally, the Neural Galerkin scheme provides an approximation of the particle density that allows efficiently computing quantities of interest such as the entropy (Figure 9a), in contrast to Monte Carlo that only draws samples of the particle distribution.



(a) entropy, harmonic trap (b) entropy, aharmonic trap

Figure 9: Particles in trap: The proposed Neural Galerkin schemes provide an approximation of the particle density that allows efficiently computing quantities of interest such as the entropy. In contrast, Monte Carlo estimation only draws samples of the particle distribution and requires subsequent density estimation with e.g., kernel density estimation (KDE) to compute quantities such as the entropy.



(a) mean positions

(b) covariance of particles

Figure 10: Particles in aharmonic trap: We use the same plotting convention as in Figure 8

5.5.2. Particles in aharmonic trap.

Next we consider an aharmonic trap with one-body force $g(t, X_i) = (a(t) - x)^3$ and the same pairwise interaction term K as defined in Eq. (36) except that α is set to $\alpha = -0.5$, so that the particles now repel each other withn the attracting trap. Because of the nonlinear force, the density u is not necessarily Gaussian anymore. A Monte Carlo estimate of the particle positions from $100,000$ samples serves as benchmark in the following.

To apply a Neural Galerkin scheme in this experiment, we take the neural network defined in Eq. (19) with $m = 40$ nodes and non-negative weights. Time is discretized with implicit Euler and $\delta t = 10^{-3}$. We take $n = 5000$ sample and 1000 SGD iterations in each time step. The rest of the setup is the same as in the example with the harmonic trap. Figure 10(a) and (b) show the relative error of the mean and the relative error of the covariance diagonal, respectively. With adaptive sampling, the used Neural Galerkin scheme achieves an error between 10^{-3} and 10^{-2}. In contrast, static sampling is insufficient and leads to large relative errors. Note that the off-diagonal elements of the covariance of the particle density converge to zero over time in this experiment and thus the relative error is not informative and not plotted here. Because Neural Galerkin schemes provide an approximation of the density of the particles, we can efficiently compute the entropy: As shown in Figure 9(b), it is in agreement with the prediction obtained with Monte Carlo and density estimation from $100,000$ samples.

6. Concluding remarks

As machine-learning techniques via deep learning play an ever more important role in enabling realistic simulations of phenomena and processes in science and engineering, there is an increasing need for training on data that are informative about the underlying physics. This problem of data generation becomes especially important in dynamical systems that evolve over time where *a priori* data is sparse or non-existent, and static data collection in a pre-processing step is insufficient. The proposed Neural Galerkin schemes intertwine data acquisition via sampling and numerical simulations in an adaptive way. The schemes learn PDE solutions by evolving a non-linear function representation, which is adjusted from data points obtained via importance sampling that is informed by the solution itself. Our numerical experiments demonstrate that the adaptivity in function approximation as well as data collection is key to enable accurate prediction of local features in high-dimensional domains.

These results suggest that the proposed Neural Galerkin schemes can be applied to simulate many other high-dimensional equations relevant in science and engineering, such as kinetic equations, non-linear Fokker-

Planck equations, Boltzmann equations, etc. This would be especially useful in situations where Monte-Carlo methods are not directly applicable because the solution does not admit a representation via the Feynman-Kac formula. The usage of Neural Galerkin schemes in such applications also opens the door to several interesting questions that we leave as future research avenues: First, we should understand better which structural properties of the neural architectures will guarantee that the function approximation from (5) $u(t, \boldsymbol{x}) \approx U(\theta(t), \boldsymbol{x})$ will hold uniformly in time. For some initial efforts in the context of shallow ReLU architectures we refer the reader to [49]. Second, we should quantify when the integral defining the instantaneous loss (8) admits an efficient estimation, beyond what non-adaptive Monte-Carlo schemes can provide, by exploiting sparsity of the residual in a certain domain, not necessarily the spatial domain. For example, if the solution is represented in the frequency domain, then locality (often referred to as sparsity) means that only few frequencies are relevant, which in principle can be exploited numerically with Neural Galerkin schemes by using suitable nonlinear parametrizations. Third, it would be interesting to explore and systematize importance sampling strategies that adapt to the transient regularity structures appearing in the solution and thereby will allow us to maintain an adaptive neuron budget. Such strategies could for example use birth-death processes [39], which would allow us to vary the width of the neural network in time, or methods that couple the PDE to some evolution equation for the input data points used to sample the residual.

References

[1] W. Anderson and M. Farazmand. Evolution of nonlinear reduced-order solutions for PDEs with conserved quantities. *SIAM Journal on Scientific Computing*, 44(1):A176–A197, 2022.

[2] Y. Bar-Sinai, S. Hoyer, J. Hickey, and M. P. Brenner. Learning data-driven discretizations for partial differential equations. *Proceedings of the National Academy of Sciences*, 116(31):15344–15349, 2019.

[3] P. Benner, S. Gugercin, and K. Willcox. A survey of projection-based model reduction methods for parametric dynamical systems. *SIAM Rev.*, 57(4):483–531, 2015.

[4] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28 – 41, 2018.

[5] K. Bhattacharya, B. Hosseini, N. B. Kovachki, and A. M. Stuart. Model reduction and neural networks for parametric PDEs. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[6] M. Billaud-Friess and A. Nouy. Dynamical model reduction method for solving parameter-dependent dynamical systems. *SIAM Journal on Scientific Computing*, 39(4):A1766–A1792, 2017.

[7] Black, Felix, Schulze, Philipp, and Unger, Benjamin. Projection-based model reduction with dynamically transformed modes. *ESAIM: M2AN*, 54(6):2011–2043, 2020.

[8] P. A. M. Dirac. Note on exchange phenomena in the thomas atom. *Mathematical Proceedings of the Cambridge Philosophical Society*, 26(3):376–385, 1930.

[9] M. W. M. G. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[10] J. Dormand and P. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6(1):19–26, 1980.

[11] Y. Du and T. A. Zaki. Evolutional deep neural network. *Phys. Rev. E*, 104:045303, Oct 2021.

[12] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Dec 2017.

[13] V. Ehrlacher, D. Lombardi, O. Mula, and F.-X. Vialard. Nonlinear model reduction on metric spaces. Application to one-dimensional conservative PDEs in Wasserstein spaces. *ESAIM Math. Model. Numer. Anal.*, 54(6):2159–2197, 2020.

[14] A. Ern and J.-L. Guermond. *Theory and practice of finite elements*. Springer, 2004.

[15] J. Frenkel. *Wave Mechanics, Advanced General Theor*. Clarendon Press, Oxford, 1934.

[16] M. D. Gunzburger. *Perspectives in Flow Control and Optimization*. Society for Industrial and Applied Mathematics, 2002.

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

[18] J. S. Hesthaven, C. Pagliantini, and N. Ripamonti. Rank-adaptive structure-preserving model order reduction of Hamiltonian systems. *ESAIM: M2AN*, 56(2):617–650, 2022.

[19] J. S. Hesthaven, C. Pagliantini, and G. Rozza. Reduced basis methods for time-dependent problems. *Acta Numerica*, 31:265–345, 2022.

[20] Y. Khoo, J. Lu, and L. Ying. Solving for high-dimensional committor functions using artificial neural networks. *Research in the Mathematical Sciences*, 6(1):1, Oct 2018.

[21] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *European Journal of Applied Mathematics*, 23:421–435, 2020.

[22] Y. Kim, Y. Choi, D. Widemann, and T. Zohdi. A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. *Journal of Computational Physics*, 451:110841, 2022.

[23] O. Koch and C. Lubich. Dynamical low-rank approximation. *SIAM Journal on Matrix Analysis and Applications*, 29(2):434–454, 2007.

[24] D. Kochkov, J. A. Smith, A. Alieva, Q. Wang, M. P. Brenner, and S. Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21), 2021.

[25] N. B. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. M. Stuart, and A. Anandkumar. Neural operator: Learning maps between function spaces. *arXiv*, 2108.08481, 2021.

[26] K. Lee and K. T. Carlberg. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. *Journal of Computational Physics*, 404:108973, 2020.

[27] Q. Li, B. Lin, and W. Ren. Computing committor functions for the study of rare events using deep learning. *The Journal of Chemical Physics*, 151(5):054112, 2019.

[28] C. Lubich. *From Quantum to Classical Molecular Dynamics: Reduced Models and Numerical Analysis*. EMS Press, 2008.

[29] E. Musharbash, F. Nobile, and T. Zhou. Error analysis of the dynamically orthogonal approximation of time dependent random pdes. *SIAM Journal on Scientific Computing*, 37(2):A776–A810, 2015.

[30] T. O'Leary-Roseberry, U. Villa, P. Chen, and O. Ghattas. Derivative-informed projected neural networks for high-dimensional parametric maps governed by PDEs. *Computer Methods in Applied Mechanics and Engineering*, 388:114199, 2022.

[31] B. Peherstorfer. Model reduction for transport-dominated problems via online adaptive bases and adaptive sampling. *SIAM Journal on Scientific Computing*, 42(5):A2803–A2836, 2020.

[32] B. Peherstorfer. Breaking the Kolmogorov barrier with nonlinear model reduction. *Notices of the American Mathematical Society*, 69:725–733, 2022.

[33] B. Peherstorfer and K. Willcox. Online adaptive model reduction for nonlinear systems via low-rank updates. *SIAM Journal on Scientific Computing*, 37(4):A2123–A2150, 2015.

[34] E. Qian, B. Kramer, B. Peherstorfer, and K. Willcox. Lift & learn: Physics-informed machine learning for large-scale nonlinear dynamical systems. *Physica D: Nonlinear Phenomena*, 406:132401, 2020.

[35] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems*, volume 20, 2008.

[36] M. Raissi, P. Perdikaris, and G. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

[37] J. Reiss, P. Schulze, J. Sesterhenn, and V. Mehrmann. The shifted proper orthogonal decomposition: a mode decomposition for multiple transport phenomena. *SIAM J. Sci. Comput.*, 40(3):A1322–A1344, 2018.

[38] F. Romor, G. Stabile, and G. Rozza. Non-linear manifold ROM with convolutional autoencoders and reduced over-collocation method. *arXiv*, 2203.00360, 2022.

[39] G. Rotskoff, S. Jelassi, J. Bruna, and E. Vanden-Eijnden. Global convergence of neuron birth-death dynamics. *arXiv*, 1902.01843, 2019.

[40] G. M. Rotskoff, A. R. Mitchell, and E. Vanden-Eijnden. Active importance sampling for variational objectives dominated by rare events: Consequences for optimization and generalization. *arXiv*, 2008.06334, 2021.

[41] G. Rozza, D. Huynh, and A. Patera. Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations. *Archives of Computational Methods in Engineering*, 15(3):1–47, 2007.

[42] S. H. Rudy, J. Nathan Kutz, and S. L. Brunton. Deep learning of dynamics and signal-noise decomposition with time-stepping constraints. *Journal of Computational Physics*, 396:483–506, 2019.

[43] T. P. Sapsis and P. F. Lermusiaux. Dynamically orthogonal field equations for continuous stochastic dynamical systems. *Physica D: Nonlinear Phenomena*, 238(23):2347–2360, 2009.

[44] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2018.

[45] T. Taddei, S. Perotto, and A. Quarteroni. Reduced basis techniques for nonlinear conservation laws. *ESAIM Math. Model. Numer. Anal.*, 49(3):787–814, 2015.

[46] T. R. Taha and M. I. Ablowitz. Analytical and numerical aspects of certain nonlinear evolution equations. III. Numerical, Korteweg-de Vries equation. *Journal of Computational Physics*, 55(2):231–253, 1984.

[47] Q. Wang, J. S. Hesthaven, and D. Ray. Non-intrusive reduced order modeling of unsteady flows using artificial neural networks with application to a combustion problem. *J. Comput. Phys.*, 384:289–307, 2019.

[48] Q. Wang, N. Ripamonti, and J. S. Hesthaven. Recurrent neural network closure of parametric POD-Galerkin reduced-order models based on the Mori-Zwanzig formalism. *J. Comput. Phys.*, 410:109402, 2020.

[49] S. Wojtowytsch and W. E. Some observations on high-dimensional partial differential equations with Barron data. In *Mathematical and Scientific Machine Learning*, 2021.

[50] R. Zimmermann, B. Peherstorfer, and K. Willcox. Geometric subspace updates with applications to online adaptive nonlinear model reduction. *SIAM Journal on Matrix Analysis and Applications*, 39(1):234–261, 2018.

A. Learning in the Banach space $\mathcal{F}_1$

Let us consider the Neural Galerkin representation $u(t, \boldsymbol{x}) = U(\theta(t), \boldsymbol{x})$ in a situation where θ is infinite dimensional, namely when

$$u(t, \boldsymbol{x}) = \int_{\mathcal{Z}} \phi(\boldsymbol{x}, z) d\gamma_t(z) \tag{39}$$

where $\phi : \mathcal{X} \times \mathcal{Z} \to \mathbb{R}$ is some nonlinear unit (like e.g. the ReLU), and γ_t is a Radon measure with finite total variation defined on $\mathcal{Z}$: we will denote the space of such Radon measures by $\mathcal{M}(\mathcal{Z})$. Functions of the type defined in Eq. (39) form a Banach space in which the norm of $u(t)$ is the minimum total variation of γ_t among all Radon measures consistent with Eq. (39); these function can also be approximated by shallow (two-layer) neural networks, which converge to functions as defined in Eq. (39) in the limit of infinite width. In this context, one can specify the evolution of γ_t by requiring that

$$\dot{\gamma}_t \in \operatorname*{argmin}_{\dot{\gamma} \in \mathcal{M}(\mathcal{Z})} \int_{\mathcal{X}} \left| \int_{\mathcal{Z}} \phi(\boldsymbol{x}, z) d\dot{\gamma}(z) - f\left(t, \boldsymbol{x}, \textstyle\int_{\mathcal{Z}} \phi(\cdot, z) d\gamma_t(z)\right) \right|^2 d\nu(\boldsymbol{x}) \tag{40}$$

where ν is some positive measure on $\mathcal{X}$. The Euler-Lagrange equation associated with this minimization problem gives the evolution equation for γ_t:

$$\int_{\mathcal{Z}} M(z, z') d\dot{\gamma}_t(z') = F(t, z, \gamma_t) \tag{41}$$

where

$$\begin{cases} M(z, z') = \displaystyle\int_{\mathcal{X}} \phi(\boldsymbol{x}, z) \phi(\boldsymbol{x}, z') d\nu(\boldsymbol{x}), \\ F(t, z, \gamma) = \displaystyle\int_{\mathcal{X}} \phi(\boldsymbol{x}, z) f\left(t, \boldsymbol{x}, \textstyle\int_{\mathcal{Z}} \phi(\cdot, z') d\gamma(z')\right) d\nu(\boldsymbol{x}) \,. \end{cases} \tag{42}$$

B. Another derivation of the Neural Galerkin equation $M(\theta)\dot{\theta} = F(t, \theta)$

Let $u(t)$ be the solution of the PDE at time t and $U(\theta(t))$ be its parametric representation. In general we do not have access to $u(t)$ (except at initial time through the prescribed initial condition), but if we had an oracle giving us this solution one option to determine the parameters $\theta(t)$ would be to minimize an objective like

$$E(\theta(t)) = R\left(u(t), U(\theta(t))\right) \tag{43}$$

where $R : \mathcal{U} \times \mathcal{U} \to \mathbb{R}$ is such that (i) $R(u, v) \geq 0$ for all $u, v \in \mathcal{U}$; (ii) $R(u, u) = 0$ for all $u \in \mathcal{U}$ and (iii) $R(u, v)$ is strictly convex in v for all $u \in \mathcal{U}$. Assuming that $R(u, v)$ is twice differentiable in both its arguments, this requires that

$$\forall u \in \mathcal{U} \quad : \quad R(u, u) = 0, \quad D_v R(u, v)|_{v=u} = 0 \quad \text{and} \quad D^2_{v,v} R(u, v)|_{v=u} \quad \text{is positive-definite} \tag{44}$$

where D denotes derivative in $\mathcal{U}$. Together with $R(u,u)=0$, note that this also implies that $D_u R(u,v)|_{v=u} = 0$ for all $u \in \mathcal{U}$.

Suppose that the current solution $u(t)$ is representable exactly, i.e. there exists $\theta(t) \in \Theta$ such that $U(\theta(t)) = u(t)$. This implies that $E(\theta(t)) = 0$, and it is natural to require that this objective remains zero, or as close to zero as possible, as time evolves. With this in mind, we can use the following result:

Proposition 1. Assume that (i) $U(\theta(t)) = u(t)$ and (ii) the objective $R(u,v)$ satisfies the conditions (44), so that $E(\theta(t)) = 0$. Then

$$\lim_{h\to 0} h^{-2} E(\theta(t+h)) = \tfrac{1}{2}\dot\theta^T M(\theta(t))\dot\theta + \dot\theta \cdot F(t,\theta(t)) + \tfrac{1}{2} b(t,\theta(t)) =: L(\dot\theta), \tag{45}$$

where

$$\begin{aligned} M(\theta) &= \langle \nabla_\theta U(\theta), D^2_{v,v} R(U(\theta),U(\theta)) \nabla_\theta U(\theta)\rangle \\ F(t,\theta) &= \langle f(t,U(\theta)), D^2_{u,v} R(U(\theta),U(\theta)) \nabla_\theta U(\theta)\rangle \\ b(t,\theta) &= \langle f(t,U(\theta)), D^2_{u,u} R(U(\theta),U(\theta)) f(t,U(\theta))\rangle \end{aligned} \tag{46}$$

Proof: A straightforward calculation using the conditions in (44) as well as $\partial_t u(t) = f(t,u(t)) = f(t,U(\theta(t))$ indicates that

$$\begin{aligned} E(\theta(t+h)) &= R\left(u(t+h), U(\theta(t+h))\right) \\ &= \tfrac{1}{2} h^2 \langle f(t,U(\theta(t))), D^2_{u,u} R(U(\theta(t)),U(\theta(t))) f(t,U(\theta(t)))\rangle \\ &+ \tfrac{1}{2} h^2 \langle \nabla_\theta U(\theta(t))\cdot\dot\theta, D^2_{v,v} R(U(\theta(t)),U(\theta(t))) \nabla_\theta U(\theta(t))\cdot\dot\theta\rangle \\ &\quad + h^2 \langle f(t,U(\theta(t))), D^2_{u,v} R(U(\theta(t)),U(\theta(t))) \nabla_\theta U(\theta(t))\cdot\dot\theta\rangle + o(h^2) \end{aligned} \tag{47}$$

Dividing by h^2 and taking the limit as $h \to 0$ establishes (45). □

Minimizing the objective defined in Eq. (45) over $\dot\theta$ given $\theta(t)$ leads us back to an evolution equation for $\theta(t)$ that is structurally similar the Neural Galerkin equation derived in the main text, that is

$$M(\theta(t))\dot\theta(t) = F(t,\theta(t)), \tag{48}$$

and is to be solved with the initial condition

$$\theta(0) \in \operatorname*{argmin}_\theta R(u_0, U(\theta)). \tag{49}$$

In particular it is easy to see that if we take

$$R(u,v) = \int_{\mathcal{X}} |u(t,\boldsymbol{x}) - v(t,\boldsymbol{x})|^2 d\nu(x), \tag{50}$$

then the system of ODEs defined in Eq. (48) is exactly the same system as the system of ODEs given in the main text with M and F given in Eq. (12) with $\nu_\theta = \nu$. The formulation above is however more general, and offers a principled way to specify the measure ν_θ. For example, for equations like the Fokker-Planck equation that evolve a probability density $u(t) > 0$ such that $\int_{\mathcal{X}} u(t,\boldsymbol{x})d\boldsymbol{x} = 1$, it common objective is the Kullback-Leibler divergence

$$R(u,v) = \int_{\mathcal{X}} \log\left(\frac{u(\boldsymbol{x})}{v(\boldsymbol{x})}\right) u(\boldsymbol{x}) d\boldsymbol{x}\,. \tag{51}$$

With this choice we arrive at Eq. (48) with

$$\begin{aligned} M(\theta) &= \int_{\mathcal{X}} \frac{\nabla_\theta U(\theta,\boldsymbol{x}) \otimes \nabla_\theta U(\theta,\boldsymbol{x})}{U(\theta,\boldsymbol{x})} d\boldsymbol{x} \\ F(t,\theta) &= \int_{\mathcal{X}} \frac{\nabla_\theta U(\theta,\boldsymbol{x}) f(t,\boldsymbol{x},U(\theta))}{U(\theta,\boldsymbol{x})} d\boldsymbol{x} \end{aligned} \tag{52}$$

These correspond to using $d\nu_\theta(\boldsymbol{x}) = |U(\theta,\boldsymbol{x})|^{-1} d\boldsymbol{x}$.

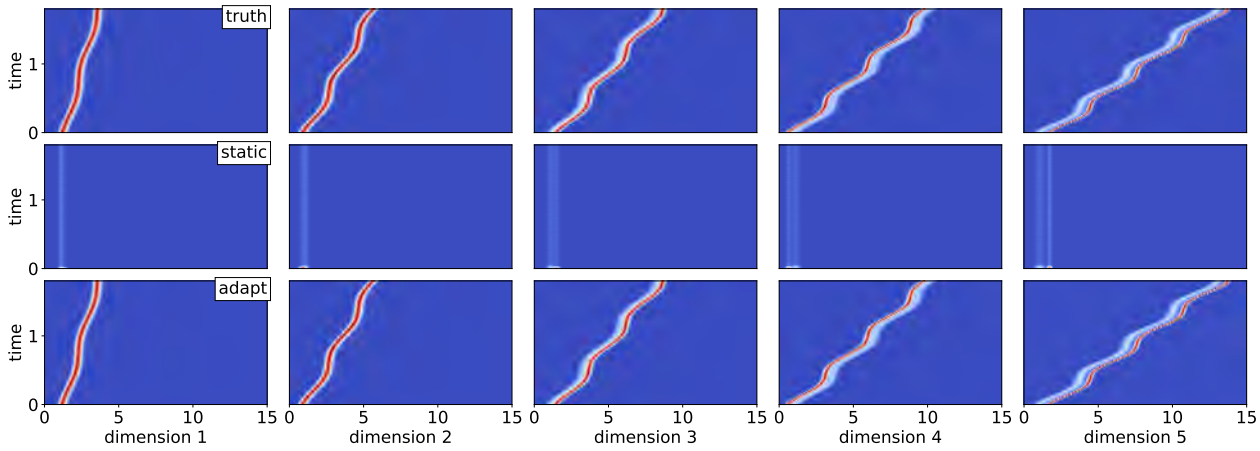


Figure 11: High-dimensional transport with time-varying coefficient: The plots labeled with "truth" show the marginals of the analytic solution. The plots labeled with "uniform" are the Neural Galerkin approximations with a static uniform sampling in $[0, 15]^d$. The plots labeled with "adapt" show the Neural Galerkin solution with adaptive sampling, which is in close agreement with the analytic solution.

C. Comparison with global methods

Let us compare the Neural Galerkin equation we use with the equations one obtains by treating the problem globally in time, i.e. by putting the spatial and the tempral variables on the same footing as in the Deep Ritz method [9, 44, 36, 4]. An optimal trajectory (in the $L_2([0,T] \times \mathcal{X}, \nu_0)$ sense) in parameter space satisfies

$$\min_{\{\theta(t)\in\Theta; t\in[0,T]\}} \int_0^T J_t(\theta, \dot{\theta}(t))dt, \tag{53}$$

subject to $\theta(0) = \theta_0$. The Euler-Lagrange equations associated with this minimization problem are

$$\frac{d}{dt}\partial_\eta J_t(\theta, \dot{\theta}) = \partial_\theta J_t(\theta, \dot{\theta}), \qquad \theta(0) = \theta_0, \qquad \partial_\eta J_T(\theta(T), \dot{\theta}(T)) = 0. \tag{54}$$

Explicitly, these read

$$\frac{d}{dt}\Big(M(t,\theta)\dot{\theta} - F(t,\theta)\Big) = H(t,\theta,\dot{\theta}), \qquad \theta(0) = \theta_0, \qquad M(T,\theta(T))\dot{\theta}(T) = F(T,\theta(T)), \tag{55}$$

where we defined

$$H(t,\theta,\eta) = \int_{\mathcal{X}} \left(\nabla_\theta \nabla_\theta U(\theta,\boldsymbol{x})\eta - \nabla_\theta U(\theta,\boldsymbol{x}) D_u f(t,\boldsymbol{x},U(\theta))\right)\left(\nabla_\theta U(\theta,\boldsymbol{x})\cdot\eta - f(t,\boldsymbol{x},U(\theta))\right) d\nu_{t,\theta}(\boldsymbol{x}) \tag{56}$$

Eqs. (55) are a boundary value problem which is harder to solve than the Neural Galerkin Eq. (11).

D. Advection in unbounded, high-dimensional domains

All five marginals for the time-varying and the time-and-space varying case are shown in Figure 11 and Figure 12, respectively.

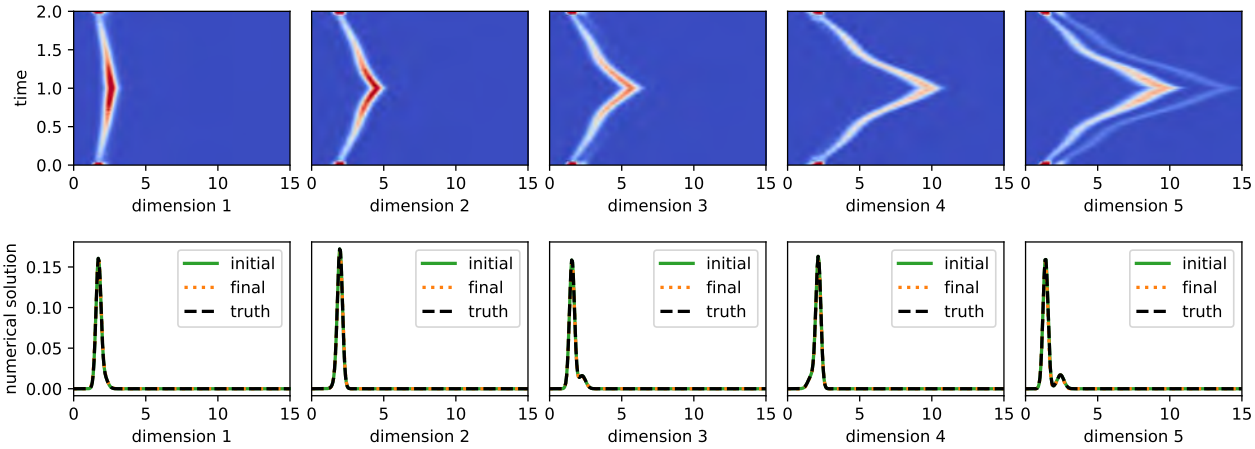


Figure 12: High-dimensional transport with time and spatially varying coefficients: The first row shows the marginals of the approximation obtained with Neural Galerkin with adaptive sampling and adaptive time integration. Because of the spatially varying coefficient, the coherent structure that is propagated over time changes shape, which is different from the time-only varying case shown in Figure 11. The transport equation is integrated forward and backward in time to compare the approximation at initial and end time, which is shown in the second row. The approximations at initial and end time match well, which indicates that the proposed adaptive approach approximates well the local, high-dimensional dynamics in this problem.

E. Fokker-Planck equation in high-dimensions

In case of the harmonic trap, the following system of ODEs describe the mean $\bar{X}_i(t)$ and covariance $C_{ij}(t)$ of particles X_i and X_j at time t for $i, j = 1, \ldots, d$ are

$$\dot{\bar{X}}_i(t) = -(1+\alpha)\bar{X}_i(t) + a(t) + \frac{\alpha}{d}\sum_{j=1}^{d}\bar{X}_j(t)\,, \qquad i = 1, \ldots, d\,, \tag{57}$$

$$\dot{C}_{ij} = -2(1+\alpha)C_{ij} + \frac{\alpha}{d}\sum_{k=1}^{d}(C_{kj} + C_{ki}) + 2\beta^{-1}\delta_{ij}\,, \qquad i, j = 1, \ldots, d\,, \tag{58}$$

where δ_{ij} is the Dirac delta. The benchmark solution is obtained by integrating these ODEs with Runge-Kutta 45. Figure 13 compares the particle positions predicted by Neural Galerkin to the benchmark solution for various dimensions.

In case of the aharmonic trap, a Monte Carlo estimate serves as benchmark. To that end, we discretize the stochastic differential equation of the particle positions via the Euler–Maruyama method with time-step size $\delta t = 10^{-4}$ and draw $100,000$ paths. From these $100,000$ paths we estimate the mean and covariance, which serve as the benchmark for the Neural Galerkin approximation. The particle positions predicted by Neural Galerkin versus the Monte Carlo benchmark are shown in Figure 14.

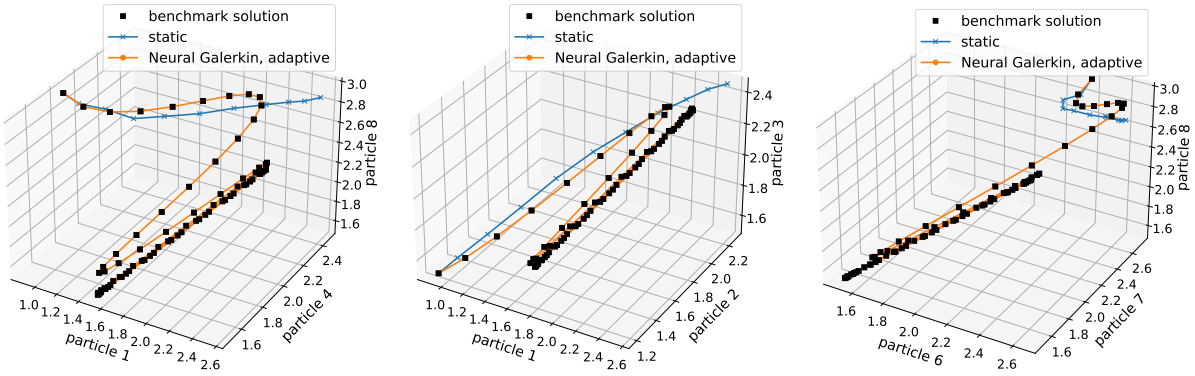


Figure 13: Particles in harmonic trap: Visualization of particle positions over time.

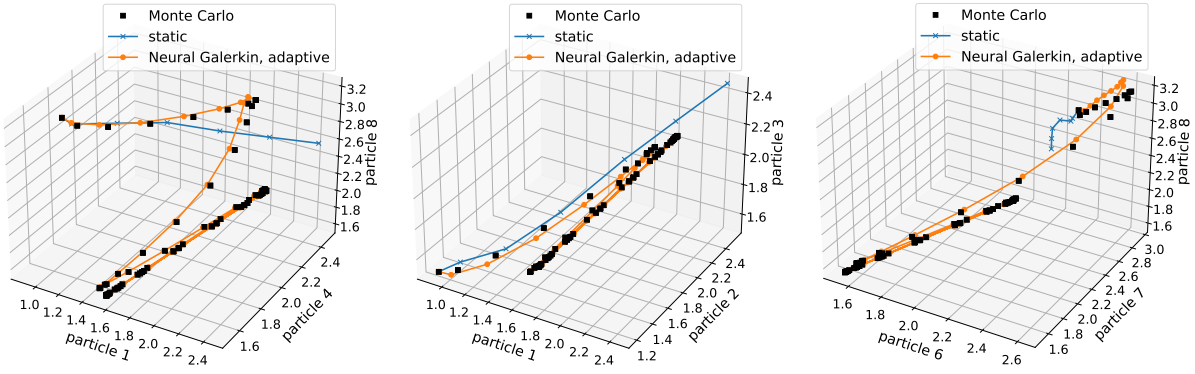


Figure 14: Particles in aharmonic trap: Visualization of particle positions over time. Note that a Monte Carlo estimate of mean serves as benchmark here.