DeepParticle: learning invariant measure by a deep neural network minimizing Wasserstein distance on data generated from an interacting particle method

Zhongjian Wang[a], Jack Xin[b], Zhiwen Zhang[c,*]

[a]*Department of Statistics and CCAM, The University of Chicago, Chicago, IL 60637, USA.*
[b]*Department of Mathematics, University of California at Irvine, Irvine, CA 92697, USA.*
[c]*Department of Mathematics, The University of Hong Kong, Pokfulam Road, Hong Kong SAR, China.*

Abstract

We introduce the so called DeepParticle method to learn and generate invariant measures of stochastic dynamical systems with physical parameters based on data computed from an interacting particle method (IPM). We utilize the expressiveness of deep neural networks (DNNs) to represent the transform of samples from a given input (source) distribution to an arbitrary target distribution, neither assuming distribution functions in closed form nor a finite state space for the samples. In training, we update the network weights to minimize a discrete Wasserstein distance between the input and target samples. To reduce computational cost, we propose an iterative divide-and-conquer (a mini-batch interior point) algorithm, to find the optimal transition matrix in the Wasserstein distance. We present numerical results to demonstrate the performance of our method for accelerating IPM computation of invariant measures of stochastic dynamical systems arising in computing reaction-diffusion front speeds through chaotic flows. The physical parameter is a large Peclét number reflecting the advection dominated regime of our interest.

AMS subject classification: 68T07, 65C35, 35K57, 49Q22.

Keywords: physically parameterized invariant measures, interacting particles, deep learning, Wasserstein distance, front speeds in chaotic flows.

1. Introduction

In recent years, deep neural networks (DNN) have achieved unprecedented level of success in a broad range of areas such as computer vision, speech recognition, natural language processing, and health sciences, producing results comparable or superior to human experts [21, 10]. The impacts have reached physical sciences where traditional first principle based modeling and computational methodologies have been the norm. Thanks in part to the user friendly open source computing platforms from industry (e.g. TensorFlow and PyTorch), there have been vibrant activities in applying deep learning tools for scientific computing,

*Corresponding author
Email addresses: zhongjian@statistics.uchicago.edu (Zhongjian Wang), jxin@math.uci.edu (Jack Xin), zhangzw@hku.hk (Zhiwen Zhang)

such as approximating multivariate functions and solving differential equations using DNNs; see e.g. [13, 30, 5, 6, 17, 49, 34, 41, 50, 35, 16, 47] and references therein.

There are many classical works on the approximation power of neural networks; see e.g. [4, 14, 7, 33]. For recent works on the expressive (approximation) power of DNNs; see e.g. [2, 37, 48, 30]. In [13] lately, the authors prove that DNNs with rectified linear unit (ReLU) activation function and enough width/depth contain the continuous piece-wise linear finite element space.

Solving ODEs or PDEs by a neural network (NN) approximation is known in the literature dating back at least to the 1990's, e.g. [22, 29, 20]. The main idea in these works is to train NNs to approximate the solution by minimizing the residual of the ODEs or PDEs and also for the associated initial and boundary conditions. These early works estimate neural network solutions on a fixed mesh, however. Recently in [6], DNNs are proposed for Poisson and eigenvalue problems with variational principle (deep Ritz), and in [12] for a class of high-dimensional parabolic PDEs with stochastic representations. The physics-informed neural network (PINN) method [35] and a deep Galerkin method (DGM) [41] compute PDE solutions based on their physical properties. For parametric PDEs, a Fourier neural operator method [24] directly learns the mapping from any functional parametric dependence to the solutions (a family of PDEs), in contrast to methods that solve one instance of an equation. Meanwhile, a deep operator network (DeepONet) was proposed to learn operators accurately and efficiently from a relatively small dataset based on the universal approximation theorem of operators [25]. Deep basis learning is studied in [28] to improve proper orthogonal decomposition for reduced order modeling of residual diffusivity in chaotic flows [27]. In the context of surrogate modeling and uncertainty quantification (UQ), DNN methods include Bayesian deep convolutional encoder-decoder networks [49], deep multi-scale model learning [44], physics-constrained deep learning method [50], see also [17, 37, 16, 47] and references therein.

Our recent work [26] develops a convergent interacting particle method (IPM) to compute the large-scale reaction-diffusion front speeds through chaotic flows. The method is based on a genetic algorithm that evolves a large ensemble of uniformly distributed particles at the initial time to another ensemble of particles obeying an invariant measure at a large time. The front speed is readily computed from the invariant measure. Though the method is mesh-free, the computational costs remain high as advection becomes dominant at large Peclét number. Clearly, it is desirable to initialize particle distribution with some resemblance of the target invariant measure instead of starting from the uniform distribution. Hence learning from samples of invariant measure at a smaller Peclét number with more affordable computation to predict an invariant measure at a larger Peclét number becomes a natural problem to study.

In this paper, we develop a DNN $f(\cdot;\theta)$ to map a uniform distribution μ (source) to an invariant measure $\nu = \nu(\sigma)$ (target) where σ is the reciprocal of Peclét number, and θ consists of network weights and σ. The network is deep from input to output with 12 Sigmoid layers while the first three layers are coupled to a shallow companion network to account for the effects of parameter σ on the network weights. In addition, we include skip connections along the deep direction to assist information flow. The network is trained by minimizing the 2-Wasserstein distance (2-WD) between two measures μ and ν [43]. We consider a discrete version of 2-WD for finitely many samples of μ and ν, which involves a linear program (LP) optimizing over doubly stochastic matrices [40]. Directly solving

the LP by the interior point method [45] is too costly. We devise a mini-batch interior point method by sampling smaller sub-matrices while preserving row and column sums. This turns out to be very efficient and integrated well with the stochastic gradient descent method for the entire network training.

We conducted three numerical experiments to verify the performance of the proposed DeepParticle method. The first example is a synthetic data set on R^1 where μ is a uniform distribution and ν being mean zero normal distribution with variance σ^2. In the second example, we compute the front speed of the KPP equation in a 2D steady cellular flow and learn the invariant measures corresponding to different σ's. In this experiment, μ is a uniform distribution on $[0, 2\pi]^2$, ν an empirical invariant measure from IPM simulation of reaction-diffusion particles in chaotic flows with Peclét number $O(\sigma^{-1})$. Finally, we compute the front speed of the KPP equation in a 3D time-dependent Kolmogorov flow and study the evolution of the measure in three-dimensional space. In this experiment, $\mu$is a uniform distribution on $[0, 2\pi]^3$, ν an empirical invariant measure from IPM simulation of reaction-diffusion particles of the KPP equation in time-dependent Kolmogorov flow. Numerical results show that the proposed DeepParticle method efficiently learns the σ dependent target distributions and predicts the physically meaningful sharpening effect as σ becomes small.

We remark that though there are other techniques for mapping distributions such as Normalizing Flows (NF) [18], Generative Adversarial Networks (GAN) [11], entropic regularization and Sinkhorn Distances [3, 32], Fisher information regularization [23], our method differs either in training problem formulation or in imposing less constraints on the finite data samples. For example, our mapping is not required to be invertible as in NF, the training objective is not min-max as in GAN for image generation, there is no regularization effect such as blurring or noise. We believe that our method is better tailored to our problem of invariant measure learning using moderate amount of training data generated from the IPM. Detailed comparison will be left for a future work.

The rest of the paper is organized as follows. In Section 2, we review the basic idea of DNNs and their properties, as well as Wasserstein distance. In Section 3, we introduce our DeepParticle method for learning and predicting invariant measures based on the 2-Wasserstein distance minimization. In Section 4, we present numerical results to demonstrate the performance of our method. Finally, concluding remarks are in Section 5.

2. Preliminaries

2.1. Artificial Neural Network

In this section, we introduce the definition and approximation properties of DNNs. There are two ingredients in defining a DNN. The first one is a (vector) linear function of the form $T : R^n \to R^m$, defined as $T(x) = Ax + b$, where $A = (a_{ij}) \in R^{m\times n}$, $x \in R^n$ and $b \in R^m$. The second one is a nonlinear activation function $\sigma : R \to R$. A frequently used activation function, known as the rectified linear unit (ReLU), is defined as $\sigma(x) = \max(0, x)$ [21]. In the artificial neural network literature, the sigmoid function is another frequently used activation function, which is defined as $\sigma(x) = (1 + e^{-x})^{-1}$. By applying the activation function in an element-wise manner, one can define (vector) activation function $\sigma : R^m \to R^m$.

Equipped with those definitions, we are able to define a continuous function $F(x)$ by a composition of linear transforms and activation functions, i.e.,

$$F(x) = T^k \circ \sigma \circ T^{k-1} \circ \sigma \cdots \circ T^1 \circ \sigma \circ T^0(x), \tag{1}$$

where $T^i(x) = A_i x + b_i$ with A_i be undetermined matrices and b_i be undetermined vectors, and $\sigma(\cdot)$ is the element-wisely defined activation function. Dimensions of A_i and b_i are chosen to make (1) meaningful. Such a DNN is called a $(k+1)$-layer DNN, which has k hidden layers. Denoting all the undetermined coefficients (e.g., A_i and b_i) in (1) as $\theta \in \Theta$, where θ is a high-dimensional vector and Θ is the space of θ. The DNN representation of a continuous function can be viewed as

$$F = F(x; \theta). \tag{2}$$

Let $\mathbb{F} = \{F(\cdot, \theta) | \theta \in \Theta\}$ denote the set of all expressible functions by the DNN parameterized by $\theta \in \Theta$. Then, $\mathbb{F}$ provides an efficient way to represent unknown continuous functions, in contrast with a linear solution space in classical numerical methods, e.g. the space of linear nodal basis functions in FEM.

2.2. Wasserstein distance and optimal transportation

Wasserstein distances are metrics on probability distributions inspired by the problem of optimal mass transportation. They measure the minimal effort required to reconfigure the probability mass of one distribution in order to recover the other distribution. They are ubiquitous in mathematics, especially in fluid mechanics, PDEs, optimal transporation, and probability theory [43].

One can define the $p-$Wasserstein distance between probability measures μ and ν by

$$W_p(\mu, \nu) := \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{Y \times Y} d(x,y)^p \, \mathrm{d}\gamma(x,y) \right)^{1/p}, \tag{3}$$

where $\Gamma(\mu, \nu)$ is the set of probability measures γ on $Y \times Y$ satisfying $\gamma(A \times Y) = \mu(A)$ and $\gamma(Y \times B) = \nu(B)$ for all Borel subsets $A, B \subset Y$. Elements $\gamma \in \Gamma(\mu, \nu)$ are called couplings of the measures μ and ν, i.e., joint distributions on $Y \times Y$ with prescribed marginals μ and ν on each axis.

In discrete case, the definition (3) has a simple intuitive interpretation: given a $\gamma \in \Gamma(\mu, \nu)$ and any pair of locations (x, y), the value of $\gamma(x, y)$ tells us what proportion of μ mass at x should be transferred to y, in order to reconfigure μ into ν. Computing the effort of moving a unit of mass from x to y by $d(x, y)^p$ yields the interpretation of $W_p(\mu, \nu)$ as the minimal effort required to reconfigure μ mass distribution into that of ν. We define a measure cost function $c(x, y)$ on $Y \times Y$, which tells how much it costs to transport one unit of mass from location x to location y. Notice that when the cost function $c(x, y) = d(x, y)^p$ the $p-$Wasserstein distance (3) reveals the Monge-Kantorivich optimization problem. We refer the interested read to [43] for more details.

In practical setting, like [32], closed form of μ and ν may be unknown, we only have N i.i.d samples of μ and ν. Then we approximate the probability measure μ and ν by empirical distribution functions

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \delta_{x_i}, \quad \text{and,} \quad \nu = \frac{1}{N} \sum_{j=1}^{N} \delta_{y_j}. \tag{4}$$

Any measure in $\Gamma(\mu,\nu)$ can clearly be represented by an $n\times n$ doubly stochastic matrix, denoted as transition matrix, $\gamma=(\gamma_{ij})_{i,j}$, which satisfies that all the γ_{ij}'s are non-negative and

$$\forall j,\ \sum_{i=1}^{N}\gamma_{ij}=1,\quad \text{and,}\quad \forall i,\ \sum_{j=1}^{N}\gamma_{ij}=1. \tag{5}$$

The empirical distribution functions allow us to approximate different measures. And the doubly stochastic matrix provides a practical tool to study the evolution of measures via minimizing the Wasserstein distance between different empirical distribution functions. Notice that when the number n of the particle points becomes large, it is expensive to find transition matrix γ to calculate discrete the Wasserstein distance. We propose an efficient subset sampling method to overcome this difficulty. More details will be presented in the next Section.

3. Methodology

3.1. Physical parameter dependent neural network

Comparing with general neural network, we propose a new architecture to learn the dependency of physical parameters. Such type of network is expected to have independently batched input, and the output should also rely on some physical parameter that is shared by the batched input. Thus in addition to concatenating the physical parameters as input, we also include some linear layers whose weight and bias are generated from a shallow network, see Fig.1 for the layout of the proposed network used in later numerical experiments.

To be more precise, we assume the network takes two kinds of input, X and κ. Batch size of them are denoted as N and n_κ correspondingly. Then there are n_κ sets of X, such that X in each set shares the same physical parameter κ. From `layer1` to `layer12` are general linear layers in which the weight and bias are randomly initialized and updated by Adams descent during training. They are 20 in width and Sigmoid function is applied as activation between adjacent layers. Arrows denote the direction in forward propagation. From `layer1_2` to `layer3_2` are linear layers with 20 in width, but every element of weight matrix and bias vector is individually generated from the $\texttt{par}-\texttt{net}$ specified on the right. Input of $\texttt{par}-\texttt{net}$ is the shared parameter, namely κ. From $\texttt{par}-\texttt{net}:\texttt{layer1}$ to $\texttt{par}-\texttt{net}:\texttt{layer3}$ are general linear layers with 10 in width. For example, given the physical parameter κ is three dimension, to generate a 20×20 weight matrix, we first tile κ to $\texttt{bsz}\times 20\times 20\times 3$ tensor κ^T. Here `bsz` denotes the shape of input batches which are independently forwarded in the network. In our numerical examples, $\texttt{bsz}=n_\kappa\times N$. Then we introduce a $20\times 3\times 10$ tensor as weight matrix w in $\texttt{par}-\texttt{net}:\texttt{layer1}$ and do matrix multiplication of κ^T and w on the last two dimensions while keeping dimensions in front. The third dimension in w is the width of the $\texttt{par}-\texttt{net}:\texttt{layer3}$. Shapes of weight tensor in $\texttt{par}-\texttt{net}:\texttt{layer2}$ and $\texttt{par}-\texttt{net}:\texttt{layer3}$ are $20\times 10\times 10$ and $20\times 10\times 1$ correspondingly. The parameters (e.g. w) of the $\texttt{par}-\texttt{net}$ are randomly initialized and updated by gradient descent during training. We also include some skip layers as ones do in the classic `Resnet` to improve the performance of deeper layers in our network. As input and output are within some physical space (usually Euclidean), in addition to classic `Resnet`, we impose a linear propagation directly from input to output. In numerical experiment shown later, such connection helps avoid output being over-clustered. There is no activation function between the last layer and output.

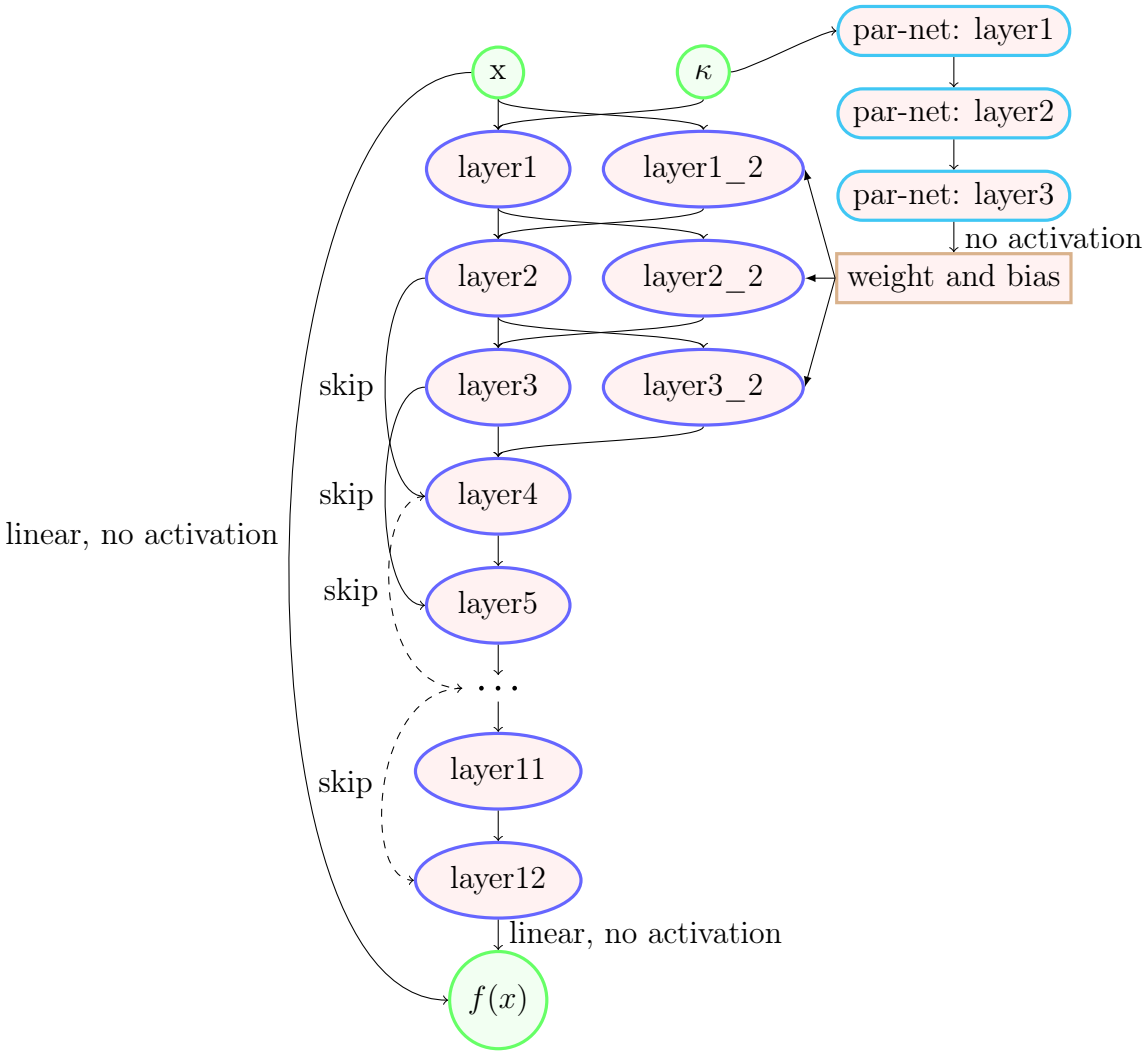


Figure 1: Layout of network

3.2. DeepParticle Algorithms

Given distributions μ and ν defined on metric spaces X and Y, we aim to construct transport map $f^0 : X \to Y$ such that

$$f^0(\mu) = \nu. \tag{6}$$

On the other side, given function $f : X \to Y$, p^{th} Wasserstein distance between $f_*(\mu)$ and ν is defined by,

$$W_p(f(\mu), \nu) := \left(\inf_{\gamma \in \Gamma(f(\mu),\nu)} \int_{Y \times Y} d(x, y)^p \, \mathrm{d}\gamma(x, y) \right)^{1/p}, \tag{7}$$

where $\Gamma(f(\mu), \nu)$ denotes the collection of all measures on $Y \times Y$ with marginals $f(\mu)$ and ν on the first and second factors respectively and d denotes the metric on Y. A straight forward derivation yields,

$$W_p(f(\mu), \nu) = \left(\inf_{\gamma \in \Gamma(\mu,\nu)} \int_{X \times Y} d(f(x), y)^p \, \mathrm{d}\gamma(x, y) \right)^{1/p}. \tag{8}$$

Network Training Objective. Our DeepParticle algorithm does not assume the knowledge of closed form distribution of μ and ν, instead we have i.i.d samples of μ and ν namely, x_i and y_i, $i = 1 \cdots N$, as training data. Then a discretization of (7) is:

$$\hat{W}(f) := \left(\inf_{\gamma \in \Gamma^N} \sum_{i,j=1}^{N} d(f(x_i), y_j)^p \gamma_{ij} \right)^{1/p} \tag{9}$$

where Γ^N denotes all $N \times N$ doubly stochastic matrix.

Let the map represented by neural network in Fig.1 be $f_\theta(x; \kappa)$ where x is the input, κ is the shared physical parameter and θ denotes all the trainable parameters in the network. Clearly $\hat{W}(f_\theta) \geq 0$. In case of $X = Y = R^d$ equipped with Euclidean metric, we take $p = 2$. The training loss function is

$$\hat{W}^2(f_\theta) := \sum_{r=1}^{N_\kappa} \left(\inf_{\gamma \in \Gamma^N} \sum_{i,j=1}^{N} |f_\theta(x_{i;r}; \kappa_r) - y_{j;r}|^2 \gamma_{ij} \right). \tag{10}$$

Iterative method in finding transition matrix γ. To minimize the penalty function (10), we update parameters θ of f_θ with the classical Adams stochastic gradient descent, and alternate with updating the transition matrix γ. Since the processes of finding γ under different input of physical parameter κ are mutually independent, we ignore the summation of $r = 1 \cdots N_\kappa$ in Eq.(10) for simplicity in later derivation.

We now present a mini-batch linear programming algorithm to find best γ given $\|f_\theta(x_i) - y_j\|^2$ in (10). Notice that the problem (10) is a linear minimization problem on the bounded convex set Γ^N of vector space of real $n \times n$ matrices. By Choquet's theorem, this problem admits solutions that are extremal points of Γ^N. Set of all doubly stochastic matrix Γ^N can be referred as Birkhoff polytope. The Birkhoff–von Neumann theorem [36] states that such polytope is the convex hull of all permutation matrices, i.e., those matrices such that $\gamma_{ij} = \delta_{j,\sigma(i)}$ for some permutation σ of $\{1, ..., n\}$, where δ_{jk} is the Kronecker symbol.

The algorithm is defined iteratively. In each iteration, we select columns and rows and solve a linear programming sub-problem under the constraint that keeps column-wise and row-wise sums of the corresponding sub-matrix of γ. To be precise, let $\{i_k\}_{k=1}^M$, $\{j_l\}_{l=1}^M$ ($M \preccurlyeq N$) denote the index chosen from $\{1, 2, \cdots, N\}$ without replacement. The cost function of sub-problem is

$$C(\gamma^*) := \sum_{k,l=1}^{M} |f_\theta(x_{i_k}) - y_{j_l}|^2 \gamma^*_{i_k j_l} \tag{11}$$

subject to

$$\begin{cases} \sum_{k=1}^M \gamma^*_{i_k,j_l} = \sum_{k=1}^M \gamma_{i_k,j_l} \quad \forall l = 1 \cdots M \\ \sum_{l=1}^M \gamma^*_{i_k,j_l} = \sum_{l=1}^M \gamma_{i_k,j_l} \quad \forall\, k = 1 \cdots M \\ \gamma^*_{i_k j_l} \geq 0 \quad \forall\, k, l = 1 \cdots M \end{cases} \tag{12}$$

where γ_{i_k,j_l} are from the previous step. This way, the column and row sums of γ are preserved by the update. The linear programming sub-problem of much smaller size is solved by the interior point method.

An important observation is that the global minimum of γ in (10) is also the solution of sub-problems (11) with arbitrary selected rows and columns. In this regard, the selection

of rows and columns can be one's own choice. In our approach, in each step after gradient descent, apart from random sampling of rows and columns, we also perform a number of searches for rows or columns with the largest entries to speed up computation; see Alg.2 1.

Algorithm 1: Random Pivot Search

Result: Given transition matrix γ, randomly search rows/columns with largest elements ("pivots").

Randomly pick i_1 as the first row;

find j_1 such that $\gamma_{i_1 j_1}$ is the largest among $\{\gamma_{i_1 1} \cdots \gamma_{i_1 N}\}$;

for $k = 2 \cdots N$ **do**

  find i_k such that $\gamma_{i_k j_{k-1}}$ is the largest among $\{\gamma_{i' j_{k-1}}\}_{i' \in \{1,\cdots,N\} \backslash \{i_1,\cdots,i_{k-1}\}}$;

  find j_k such that $\gamma_{i_k j_k}$ is the largest among $\{\gamma_{i_k j'}\}_{j' \in \{1,\cdots,N\} \backslash \{j_1,\cdots,j_{k-1}\}}$;

end

Update of training data. Note that given any fixed set of $\{x_i\}$ and $\{y_i\}$ (training data), we have developed the iterative method to calculate the optimal transition matrix γ in (10) and update network parameter θ at the same time. However, in more complicated case, more than one set of data (size of which is denoted as `bsz`) should be assimilated. The total number of data set is denoted N_{dict}. For second and later set of training data, the network is supposed to establish some accuracy. So before updating the network parameter, we first utilize our iterative method to find an approximated new γ for new data set up to some preset level. The level of γ is evaluated by Frobenius norm.

Full process of training can be found in Algorithm 2.

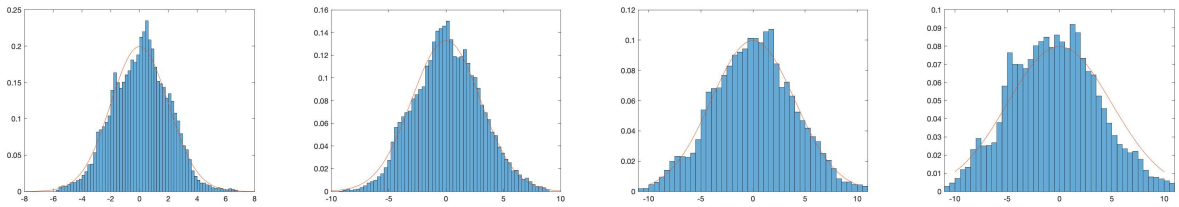
4. Numerical Examples

4.1. Mapping uniform to normal distribution

For illustration, we first apply our algorithm to learn a map from 1D uniform distribution on $[0, 1]$ to 1D normal distribution with same mean ($\mu = 0$) but different standard derivation σ. Then σ refers to one dimensional κ in the network Fig.1. As training data, we independently generate $n_\sigma = 8$ sets of $N = 1500$ uniformly distributed points, also $N = 1500$ normally distributed sample points with zero mean and standard derivation σ equally spaced in $[2, 3.75]$. During training, we aim to find a σ-dependent network such that the output from the 8 input data sets along with their σ values approximate normal distributions with standard derivation (σ) respectively. The layout of network is shown in Fig. 1. The total steps of training is 10^4 with initial learning rate 0.02 and $N_{dict} = 1$ data set. After each update of parameters, we solve the optimization problem of transition matrix γ, i.e. Eq.(11) under Eq.(12), with selection of rows and columns by Alg.1 for $N_r = 5$ times. After training, we generate $N = 40000$ uniformly distributed test data points and apply the networks with $\sigma = 2$, 3, 4, 5. In Fig. 2, we plot the output histograms. Clearly, the empirical distributions vary with σ and fit the reference normal probability distribution functions (pdfs) in red lines. We would like to clarify that even though the dependence of σ in this example is 'linear', the linear propagation directly from input to output does not depend on σ and output $\texttt{par} - \texttt{net}$ is no longer linear function of σ Our experiments are all carried out on a quad-core CPU desktop with a RTX 2080 graphic processing card.

Algorithm 2: DeepParticle Learning

Initialize parameters θ in network $f_\theta : R^d \to R^d$;
repeat
  randomly select $\{x_i\}$, $\{y_i\}$, $i = 1 : N$ from i.i.d. samples of input and target distribution;
  $\gamma_{ij} = 1/N$;
  if *not the first training set* **then**
    $P = \sum_{i,j=1}^{N} |f_\theta(x_i) - y_j|^2 \gamma_{ij}$;
    while $||\gamma||_{fro} < tol$ **do**
      randomly (or by Alg.1) choose $\{i_k\}_{k=1}^{M}$, $\{j_l\}_{l=1}^{M}$ from $\{1, 2, \cdots, N\}$ without replacement;
      solve the linear programming sub-problem (11)-(12) to get γ^*;
      update $\{\gamma_{i_k j_l}\}_{k,l=1}^{M}$ with $\{\gamma^*_{i_k j_l}\}_{k,l=1}^{M}$.
    end
  end
  repeat
    $P = \sum_{i,j=1}^{N} |f_\theta(x_i) - y_j|^2 \gamma_{ij}$;
    $\theta \leftarrow \theta - \delta_1 \nabla_\theta P$, δ_1 is the learning step size;
    repeat
      randomly (or by Alg.1) choose $\{i_k\}_{k=1}^{M}$, $\{j_l\}_{l=1}^{M}$ from $\{1, 2, \cdots, N\}$ without replacement;
      solve the linear programming sub-problem (11)-(12) to get γ^*;
      update $\{\gamma_{i_k j_l}\}_{k,l=1}^{M}$ with $\{\gamma^*_{i_k j_l}\}_{k,l=1}^{M}$.
    until *given steps* N_r;
  until *given steps for each training set*;
until *given number of training set,* N_{dict};
Return



(a) $\sigma = 2$ (b) $\sigma = 3$ (c) $\sigma = 4$ (d) $\sigma = 5$

Figure 2: Generated histograms with the lines being the ground truth pdfs.

4.2. Computing front propagation in complex fluid flows

Front propagation in complex fluid flows arises in many scientific areas such as turbulent combustion, chemical kinetics, biology, transport in porous media, and industrial deposition processes [46]. A fundamental problem is to analyze and compute large-scale front speeds. An extensively studied model problem is the reaction-diffusion-advection (RDA) equation with Kolmogorov-Petrovsky-Piskunov (KPP) nonlinearity [19]:

$$u_t = \sigma \Delta_{\mathbf{x}} u + (\mathbf{v} \cdot \nabla_{\mathbf{x}}) u + u(1-u), \quad t \in \mathbb{R}^+, \quad \mathbf{x} = (x_1, ..., x_d)^T \in \mathbb{R}^d, \tag{13}$$

where σ is diffusion constant, $\mathbf{v}$ is an incompressible velocity field (its precise definition will be discussed later), and u is the concentration of reactant. Let us consider velocity fields $\mathbf{v} = \mathbf{v}(t, \mathbf{x})$ to be T-periodic in space and time, which contain celebrated Arnold-Beltrami-Childress (ABC) and Kolmogorov flows as well their variants with chaotic streamlines [1, 15]. The solutions from compact non-negative initial data spread along direction $\mathbf{e}$ with speed [31]:

$$c^*(\mathbf{e}) = \inf_{\lambda > 0} \mu(\lambda)/\lambda,$$

where $\mu(\lambda)$ is the principal eigenvalue of the parabolic operator $\partial_t - \mathcal{A}$ with:

$$\mathcal{A}w := \sigma \, \Delta w + (2\lambda \, \mathbf{e} + \mathbf{v}) \cdot \nabla_{\mathbf{x}} w + \big(\sigma \, \lambda^2 + \lambda \, \mathbf{v} \cdot \mathbf{e} + 1\big) w, \tag{14}$$

on the space domain $\mathbb{T}^d := [0, T]^d$ (periodic boundary condition). It is known [31] that $\mu(\lambda)$ is convex in λ, and superlinear for large λ. The operator $\mathcal{A}$ in (14) is a sum $\mathcal{A} = \mathcal{L} + \mathcal{C}$, with

$$\mathcal{L} := \sigma \Delta \cdot + (2 \, \lambda \, \mathbf{e} + \mathbf{v}) \cdot \nabla_{\mathbf{x}} \cdot, \;\; \mathcal{C} := c(t, \mathbf{x}) \cdot = \big(\sigma \, \lambda^2 + \; \lambda \, \mathbf{v} \cdot \mathbf{e} + 1\big) \cdot \tag{15}$$

where $\mathcal{L}$ is the generator of a Markov process, and $\mathcal{C}$ acts as a potential. The Feynman-Kac (FK) formula [8] gives $\mu(\lambda)$ a stochastic representation:

$$\mu = \lim_{t \to \infty} \, t^{-1} \ln \left(\mathbb{E} \exp \left\{ \int_0^t \, c(t - s, \boldsymbol{X}_s^{t, \boldsymbol{x}}) \, ds \right\} \right), \tag{16}$$

where:

$$d \, \boldsymbol{X}_s^{t, \boldsymbol{x}} = \boldsymbol{b}(t - s, \boldsymbol{X}_s^{t, \boldsymbol{x}}) \, ds + \sqrt{2\sigma} \, d\boldsymbol{W}_s, \;\; \boldsymbol{X}_0^{t, \boldsymbol{x}} = \boldsymbol{x},$$

where the drift term $\mathbf{b} = 2 \, \lambda \, \mathbf{e} + \mathbf{v}$ and $\mathbf{W}(s)$ is the standard d-dimensional Wiener process. The expectation $\mathbb{E}(\cdot)$ in (16) is over $\mathbf{W}(t)$. Directly applying (16) and Monte Carlo method to compute $\mu(\lambda)$ is unstable as the main contribution to $\mathbb{E}(\cdot)$ comes from sample paths that visit maximal or minimal points of the potential function c, resulting in inaccurate or even divergent results. A computationally feasible alternative is a scaled version, the FK semi-group:

$$\Phi_t^c(\nu)(\phi) := \frac{\mathbb{E}[\phi(\boldsymbol{X}_t^{t, \boldsymbol{x}}) \, \exp\{\int_0^t \, c(t - s, \boldsymbol{X}_s^{t, \boldsymbol{x}}) \, ds\}]}{\mathbb{E}[\exp\{\int_0^t \, c(t - s, \boldsymbol{X}_s^{t, \boldsymbol{x}}) \, ds\}]} := \frac{P_t^c(\nu)(\phi)}{P_t^c(\nu)(1)}.$$

Acting on any initial probability measure ν, $\Phi_{nT}^c(\nu)$ converges weakly to an *invariant measure* ν_c (i.e. $\Phi_T^c(\nu_c) = \nu_c$) as $n \to \infty$, for any smooth test function ϕ. Moreover,

$$P_t^c(\nu_c) = \exp\{\mu \, t\} \, \nu_c \;\; \text{or} \;\; \mu = t^{-1} \, \ln \, \mathbb{E}_{\nu_c}[P_t^c(\nu_c)]. \tag{17}$$

An interacting particle method (IPM) proceeds to discretize $\boldsymbol{X}_s^{t,\boldsymbol{x}}$ as $\boldsymbol{X}_i^{\Delta t}$ by Euler's method, approximates the evolution of probability measure $\Phi_t^c(\nu)$ by a particle system with a re-sampling technique to reduce variance. Let

$$P_n^{c,\Delta t}(\nu)(\phi) := \mathbb{E}\left[\phi(\boldsymbol{X}_i^{\Delta t})\,\exp\left\{\Delta t\,\sum_{i=1}^{n} c((n-i)\Delta t, \boldsymbol{X}_i^{\Delta t})\right\}\right]$$

Let m be the number of time steps on $[0,T]$. Then sampled FK semi-group actions on ν:

$$\Phi_{n\,m}^{c,\Delta t}(\nu)(\phi) = \frac{P_{n\,m}^{c,\Delta t}(\nu)(\phi)}{P_{n\,m}^{c,\Delta t}(\nu)(1)} \longrightarrow \int_{\mathbb{T}^d} \phi\, d\,\nu_{c,\Delta t}, \quad \text{as } n \uparrow \infty,\ \forall \text{ smooth } \phi,$$

where $\nu_{c,\Delta t}$ is an invariant measure of the discrete map $\Phi_m^{c,\Delta t}$, thanks to $\mathbf{b}$ being T-periodic in time. There exists $q \in (0,1)$ so that [26]: $\mu_{\Delta t} := (\Delta t)^{-1}\,\ln[P_1^{c,\Delta t}(\nu_{c,\Delta t})(1)] = \mu + o((\Delta t)^q)$. The IPM algorithm below (Algorithm 3) approximates the evolving measure as an empirical measure of a large number of evolving particles that undergo advection-diffusion ($\mathcal{L}$) and mutations ($\mathcal{C}$). The mutation relies on fitness and its normalization defined as in the FK semi-group. The evolution in n is viewed as generations while there are m mutations in each generation with lifespan T.

Algorithm 3: Genetic Interacting Particle Method

Initialize first generation of N particles $\boldsymbol{\xi}_1^0 = (\xi_1^{0,1}, \cdots, \xi_1^{0,N})$, uniformly distributed over $\mathbb{T}^d$ ($d \geq 2$). Let g be the generation number in approximating $\nu_{c,\Delta t}$. Each generation moves and replicates m-times, with a life span T (time period), time step $\Delta t = T/m$.

while $g = 1 : G-1$ **do**

  while $j = 0 : m-1$ **do**

    $\boldsymbol{\eta}_g^j \leftarrow$ one-step-advection-diffusion update on $\boldsymbol{\xi}_g^j$. Define fitness $\boldsymbol{F} := \exp\{c(T - j\Delta t, \boldsymbol{\eta}_g^j)\,\Delta t\}$.

    $E_{g,j} := \frac{1}{\Delta t}\ln$ (mean population fitness). Normalize fitness to weight $\boldsymbol{p} := \boldsymbol{F}/SUM(\boldsymbol{F})$.

    $\boldsymbol{\xi}_g^{j+1} \leftarrow$ resample $\boldsymbol{\eta}_g^j$ via multinomial distribution with weight $\boldsymbol{p}$.

  end

  $\boldsymbol{\xi}_{g+1}^0 \leftarrow \boldsymbol{\xi}_g^m$, $E_g \leftarrow$ mean $(E_{g,j})$ over j.

end

Output: approximate $\mu_{\Delta t} \leftarrow \text{mean}(E_g)$, and $\boldsymbol{\xi}_G^0$.

Much progress has been made in finite element computation [39, 38, 51] of the KPP principal eigenvalue problem (14) particularly in steady 3D flows $\mathbf{v} = \mathbf{v}(\mathbf{x})$. However, when σ is small and the spatial dimension is 3, adaptive FEM can be extremely expensive. For 2D time periodic cellular flows, adaptive basis deep learning is found to improve the accuracy of reduced order modeling [27, 28]. Extension of deep basis learning to 3D in the Eulerian setting has not been attempted partly due to the costs of generating sufficient amount of training data.

Though IPM is spatially grid free and self-adaptive, the computational bottleneck remains at small σ when we need a large number of particles running for many generations to approach the invariant measure. To address this issue, we apply our DeepParticle method to generate particle samples at *a learned distribution* resembling the invariant measure and

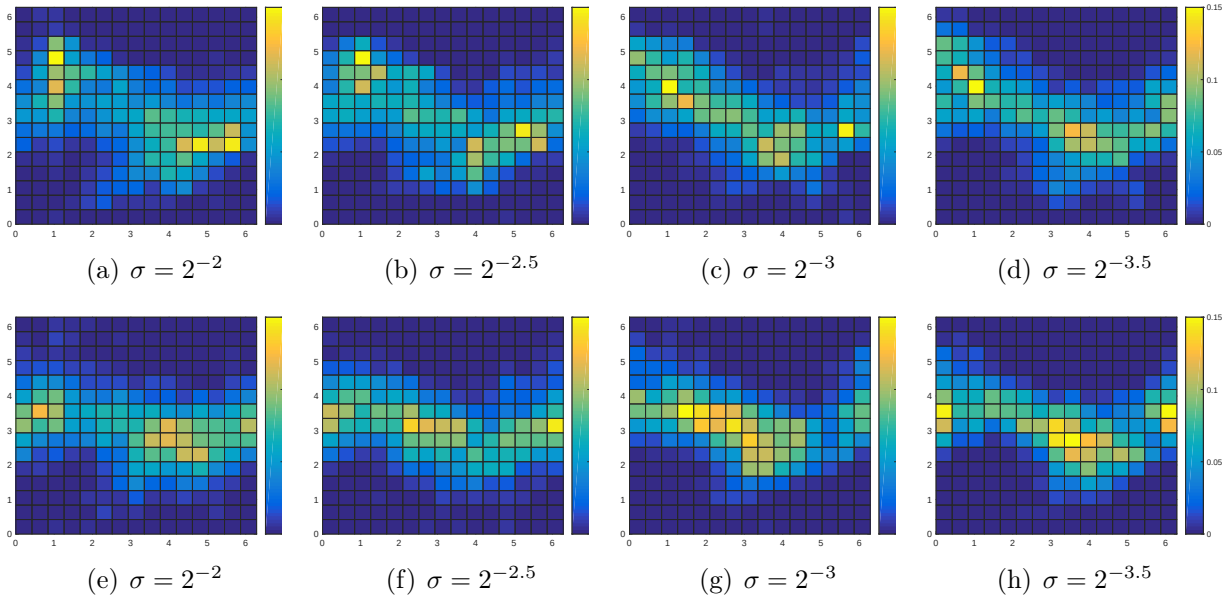
accelerate Alg. 3 (i.e. reduce G to reach convergence). An advantage of IPM is that the cost of generating training data in 3D for deep learning in this Lagrangian setting is nearly the same as that in 2D.

2D steady cellular flow. We first consider 2D cellular flow $\mathbf{v} = (-\sin x_1 \cos x_2, \cos x_1 \sin x_2)$. In this case, we apply the physical parameter dependent network described in Section 3.1 to learn the invariant measure corresponding to different σ values. To generate training data, we first use the interacting particle methods with $\sigma_i = 2^{-2+0.25*(i-1)}$, $i = 1, \cdots, n_\sigma = 8$, and $N_0 = 40000$ particles for $G = 2048$ to get samples of invariant measure with different σ values respectively. The value of G is chosen so that the direct IPM simulation of principle eigenvalue converges, see blue line of Fig.4(d). From each set of samples with different σ, we randomly pick $N = 2000$ samples without replacement, denoting as $\mathcal{Y}_1, \cdots, \mathcal{Y}_8$. Under such setting, we then seek neural networks f_θ such that given σ_i and a set of M i.i.d uniformly distributed points $\mathcal{X}_i$ on $[0, 2\pi]^2$, we have output samples $f_\theta(\mathcal{X}_i; \sigma_i)$ distributed near $\mathcal{Y}_i$ in W^2 distance. The set $\{\mathcal{Y}_1, \cdots, \mathcal{Y}_8\}$ is called one set of training data and in total we have $N_{dict} = 5$ sets through 50000 steps of training. As stated in Alg. 2, after repeated sampling (mini-batching) of training data, we optimize γ until its Frobenius norm is greater than $tol = 0.7$.

During training, we apply Adams gradient descent with initial learning rate 0.002 and a weight decay with hyper-parameter 0.005 for trainable weights in the network. In each step, we select $M = 25$ columns and rows by Alg. 1 to solve sub-optimization problem (11) and repeat for $N_r = 10$ times.

In Fig. 3, we present the performance of our algorithm in sampling distribution in the original training data. Each time, the training algorithm only has access to at most $N \times N_{dict} = 10000$ samples of each target distribution with $\sigma_i = 2^{-2+0.25*(i-1)}$, $i = 1, \cdots, n_\sigma = 8$. After training, we generate $N_0 = 40000$ uniform distribution as input of the network. We compare the histogram of network output and the $N_0 = 40000$ samples of invariant measure from the interactive particle method. We can see that our trained parameter-dependent network indeed reproduces the sharpening effect on the invariant measure when σ becomes small.

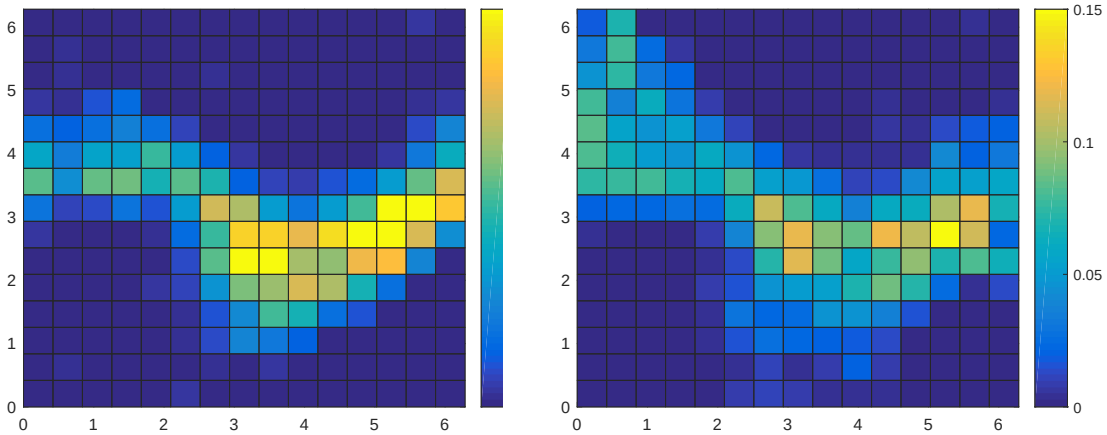
In Fig. 4, we present the performance of algorithm in predicting invariant measure of IPM with $\sigma = 2^{-4}$. First we compare the invariant measure generated from direct simulation with IPM (Fig.4(a)) and from the trained network (Fig. 4(b)). In Fig. 4(c), we show the W_2 distance between the generated distribution and the target distribution of the training data. The W_2 distances goes up when we re-sample the training data. This is because at this moment, transition matrix γ in definition of the W_2 distance is inaccurate. Note that the test $\sigma = 2^{-4}$ value is no longer within the training range $[2, 3.75]$. Our trained network predicts an invariant measure. Such prediction also serves as a 'warm start' for IPM and can accelerate its computation to quickly reach a more accurate invariant measure. As the invariant measure has no closed form analytical solution, we plot in Fig. 4(d) the principal eigenvalue approximations by IPM with warm and cold starts, i.e. one generated from DeepParticle network (warm) and the i.i.d samples from uniform distribution (cold). The warm start by DeepParticle achieves **4x to 8x speed up**. Fig. 5 displays histograms of $N = 40000$ samples generated from our DeepParticle algorithm at different stage of training. In Fig. 5(d) with only one set of data, we already get a rough prediction of target



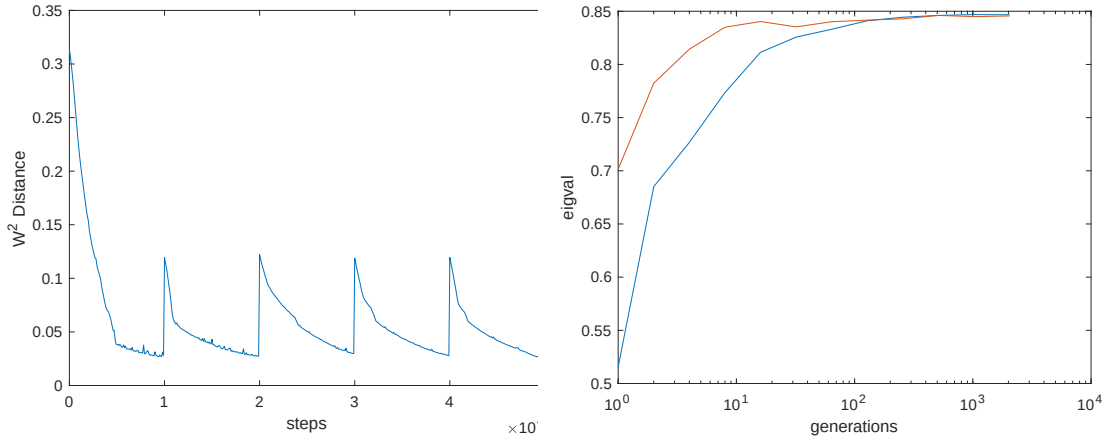
(a) $\sigma = 2^{-2}$ (b) $\sigma = 2^{-2.5}$ (c) $\sigma = 2^{-3}$ (d) $\sigma = 2^{-3.5}$

(e) $\sigma = 2^{-2}$ (f) $\sigma = 2^{-2.5}$ (g) $\sigma = 2^{-3}$ (h) $\sigma = 2^{-3.5}$

Figure 3: Generated invariant measure (upper row) and corresponding training data (reference, lower row)



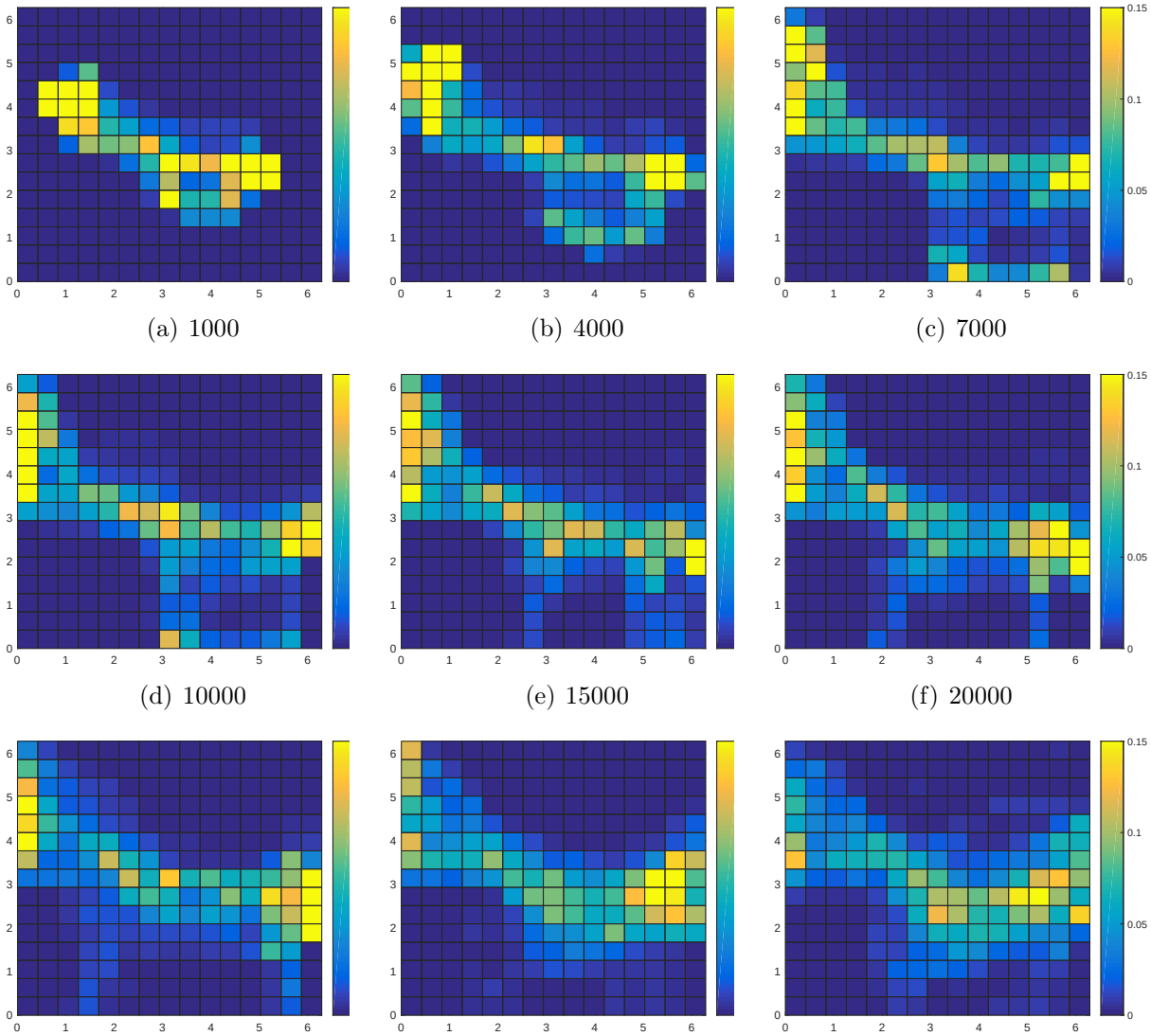
(a) Ground truth invariant measure (b) Predicted invariant measure



(c) W^2 training error vs. steps: spikes of height ≈ 0.1 occur due to transition matrix γ re-optimized in response to mini-batching of input training data.

(d) Acceleration: comparing convergence to $\mu_{\Delta t}$ value computed by Alg. 3 with warm/cold start by DeepParticle (red)/uniform distribution (blue).

Figure 4: DeepParticle prediction (top), training/acceleration (bottom) at test value $\sigma = 2^{-4}$ in a 2D front speed computation.



(a) 1000 (b) 4000 (c) 7000

(d) 10000 (e) 15000 (f) 20000

(g) 30000 (h) 40000 (i) 50000

Figure 5: Measures generated at different steps in training.

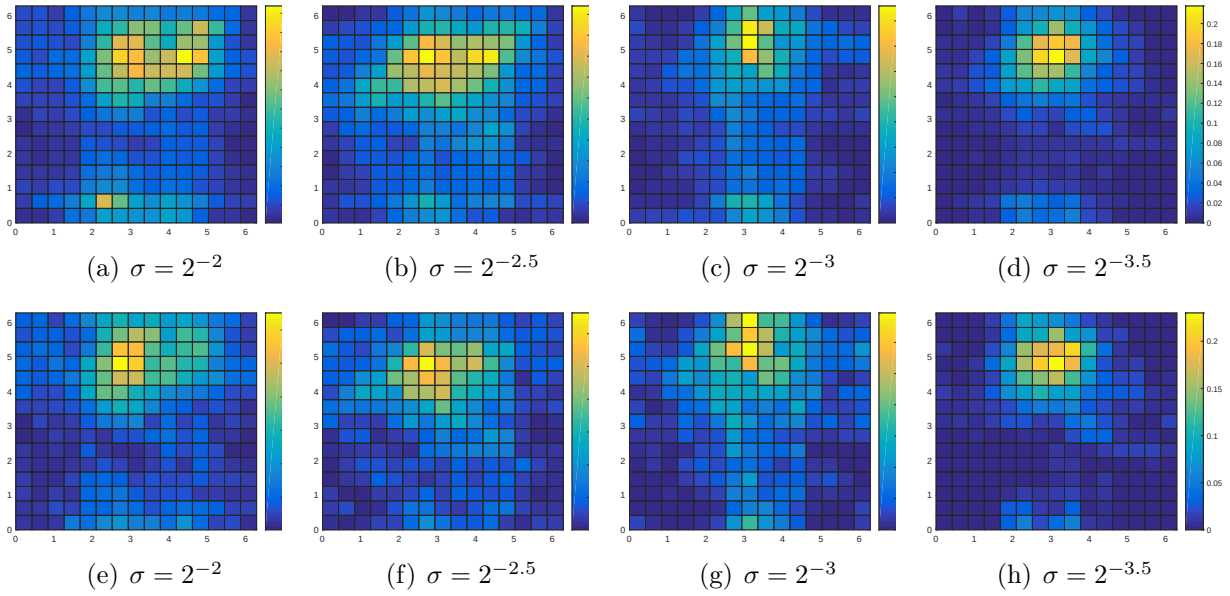
distribution. With more data sets added, the prediction quality improves; see Fig.5(e) to Fig.5(i).

3D time-dependent Kolmogorov flow. Next, we compute KPP front speed in a 3D time-dependent Kolmogorov flow [9]:

$$\mathbf{v} = \left(\sin\left(x_3 + \theta\sin(2\pi t)\right), \sin\left(x_1 + \theta\sin(2\pi t)\right), \sin\left(x_2 + \theta\sin(2\pi t)\right)\right)$$

where $\lambda = 0.5$ and $\theta = 1$. The hyper-parameter setting for network training remains the same as in the 2D case. The physical parameter dependent network in Section 3.1 learns the invariant measure corresponding to a few σ training values. Besides the input $\mathbf{x}$ and output $\mathbf{f}(\mathbf{x})$ dimension being 3, the network layout and training procedure are same as in the 2D example.

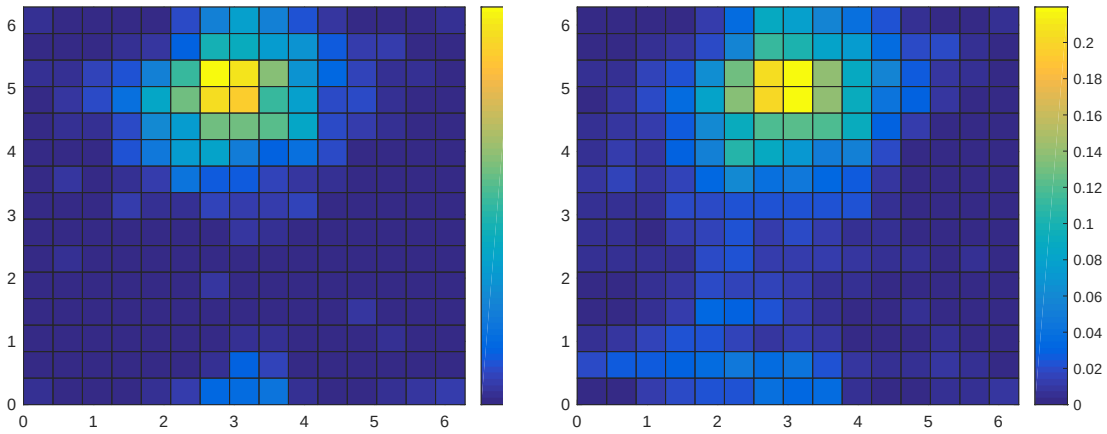
Fig. 6 shows the performance of the network interpolating training data from $\sigma = 2^{-2}$ to $\sigma = 2^{-3.5}$. Fig. 7 displays network prediction at $\sigma = 2^{-4}$. All the histograms here are 2D projections of 3D histogram to the first and third dimensions.



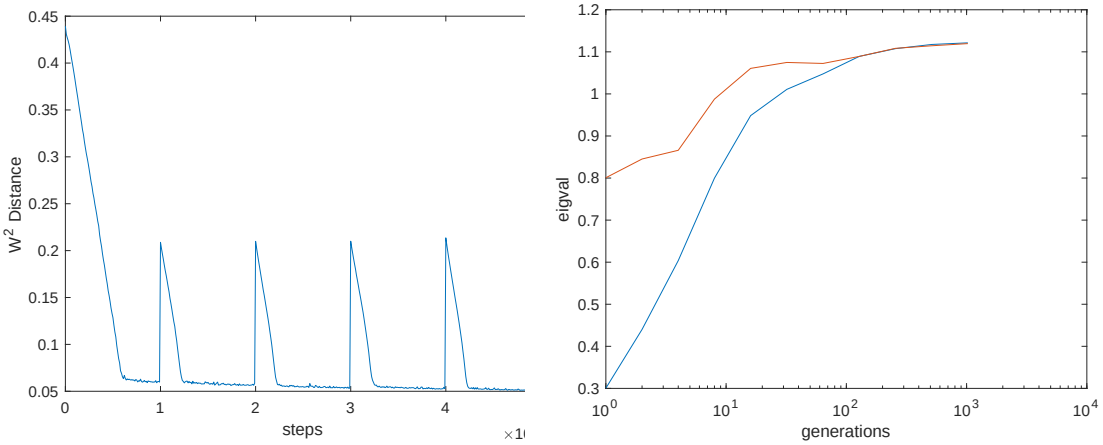
(a) $\sigma = 2^{-2}$ (b) $\sigma = 2^{-2.5}$ (c) $\sigma = 2^{-3}$ (d) $\sigma = 2^{-3.5}$

(e) $\sigma = 2^{-2}$ (f) $\sigma = 2^{-2.5}$ (g) $\sigma = 2^{-3}$ (h) $\sigma = 2^{-3.5}$

Figure 6: Generated invariant measure (upper row) and the corresponding training data (reference, lower row), viewed as projections to 1st and 3rd dimensions in a 3D KPP front speed computation.



(a) Ground truth invariant measure (b) Predicted invariant measure



(c) W^2 training error vs. steps: spikes with height ≈ 0.2 occur due to transition matrix γ re-optimized in response to mini-batching of input training data.

(d) Acceleration: comparing convergence to $\mu_{\Delta t}$ value computed by Alg. 3 with warm/cold start by DeepParticle (red)/uniform distribution (blue).

Figure 7: DeepParticle prediction (viewed in 1st/3rd dim, top) at test value $\sigma = 2^{-4}$ in a 3D computation.

5. Conclusions

We developed a DeepParticle method to generate invariant measures of stochastic dynamical (interacting particle) systems by a physically parameterized DNN that minimizes Wasserstein distance of source and target distributions. Our method is very general in the sense that we do not require closed-form distributions and the generation map to be invertible. Thus, our method is fully data-driven and applicable in fast computation of invariant measures of other interacting particle systems. During the training stage, we update network parameters based on a discretized Wasserstein distance defined on finite samples. We proposed an iterative divide-and-conquer algorithm that allows us to significantly reduce the computational cost in finding the optimal transition matrix in the Wasserstein distance. We carried out numerical experiments to show that our method is very efficient and helps accelerate the computation of invariant measures of stochastic dynamic systems. We plan to further develop the neural network architecture here by including self-attention mechanism [42] for the advancement of the DeepParticle method in the future.

Acknowledgements

The research of JX is partially supported by NSF grants DMS-1924548 and DMS-1952644. The research of ZZ is supported by Hong Kong RGC grant projects 17300318 and 17307921, National Natural Science Foundation of China No. 12171406, Seed Funding Programme for Basic Research (HKU), and Basic Research Programme (JCYJ20180307151603959) of The Science, Technology and Innovation Commission of Shenzhen Municipality.

References

[1] S. Childress and A. Gilbert. *Stretch, Twist, Fold: The Fast Dynamo*. Lecture Notes in Physics Monographs, No. 37, Springer, 1995.

[2] N. Cohen, O. Sharir, and A. Shashua. On the expressive power of deep learning: A tensor analysis. In *Conference on Learning Theory*, pages 698–728, 2016.

[3] M. Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. *Advances in neural information processing systems*, 26:2292–2300, 2013.

[4] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[5] 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, 2017.

[6] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[7] S. Ellacott. Aspects of the numerical analysis of neural networks. *Acta Numerica*, 3:145–202, 1994.

[8] M. Freidlin. *Functional Integration and Partial Differential Equations*. Princeton Univ. Press, 1985.

[9] D. Galloway and M. Proctor. Numerical calculations of fast dynamos in smooth velocity fields with realistic diffusion. *Nature*, 356(6371):691–693, 1992.

[10] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. *Advances in neural information processing systems*, 27, 2014.

[12] 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.

[13] J. He, L. Li, J. Xu, and C. Zheng. Relu deep neural networks and linear finite elements. *arXiv:1807.03973*, 2018.

[14] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[15] C. Kao, Y-Y Liu, and J. Xin. A Semi-Lagrangian Computation of Front Speeds of G-equation in ABC and Kolmogorov Flows with Estimation via Ballistic Orbits. *arXiv:2012.11129, to appear in SIAM J. Multiscale Modeling and Simulation*, 2021.

[16] S. Karumuri, R. Tripathy, I. Bilionis, and J. Panchal. Simulator-free solution of High-Dimensional Stochastic Elliptic Partial Differential Equations using Deep Neural Networks. *arXiv:1902.05200*, 2019.

[17] J. Khoo, Y.and Lu and L. Ying. Solving parametric pde problems with artificial neural networks. *arXiv:1707.03351*, 2017.

[18] I. Kobyzev, S. Prince, and M. Brubaker. Normalizing flows: An introduction and review of current methods. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[19] A. Kolmogorov, I. Petrovsky, and N. Piskunov. Investigation of the equation of diffusion combined with increasing of the substance and its application to a biology problem. *Bull. Moscow State Univ. Ser. A: Math. Mech*, 1(6):1–25, 1937.

[20] I. Lagaris, A. Likas, and D. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.*, 9(5):987–1000, 1998.

[21] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[22] H. Lee. Neural algorithm for solving differential equations. *Journal of Computational Physics*, 91:110–131, 1990.

[23] W. Li, P. Yin, and S. Osher. Computations of optimal transport distance with Fisher information regularization. *Journal of Scientific Computing*, 75(3):1581–1595, 2018.

[24] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[25] L. Lu, P. Jin, and G. Karniadakis. Deeponet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv:1910.03193*, 2019.

[26] J. Lyu, Z. Wang, J. Xin, and Z. Zhang. A convergent interacting particle method and computation of KPP front speeds in chaotic flows. *arXiv 2103.14796*, 2021.

[27] J. Lyu, J. Xin, and Y. Yu. Computing residual diffusivity by adaptive basis learning via spectral method. *Numerical Mathematics: Theory, Methods and Applications*, 10(2):351–372, 2017.

[28] J. Lyu, J. Xin, and Y. Yu. Computing residual diffusivity by adaptive basis learning via super-resolution deep neural networks. *In: H. A. Le Thi, et al (eds), Advanced Computational Methods for Knowledge Engineering. ICCSAMA 2019. Advances in Intelligent Systems and Computing*, 1121:279–290, 2020.

[29] A. Meade and A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Math. Comput. Model.*, 19(12):1–25, 1994.

[30] H. Montanelli and Q. Du. New error bounds for deep ReLU networks using sparse grids. *SIAM Journal on Mathematics of Data Science*, 1(1):78–92, 2019.

[31] J. Nolen, M. Rudd, and J. Xin. Existence of KPP fronts in spatially-temporally periodic advection and variational principle for propagation speeds. *Dynamics of PDEs*, 2(1):1–24, 2005.

[32] G. Peyré and M. Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5-6):355–607, 2019.

[33] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta numerica*, 8:143–195, 1999.

[34] M. Raissi, P. Perdikaris, and G. Karniadakis. Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems. *arXiv:1801.01236*, 2018.

[35] 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.

[36] A. Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[37] C. Schwab and J. Zech. Deep Learning in High Dimension. *Research Report*, vol. 2017, 2017.

[38] L. Shen, J. Xin, and A. Zhou. Finite element computation of KPP front speeds in 3D cellular and ABC flows. *Mathematical Modelling of Natural Phenomena*, 8(3):182–197, 2013.

[39] L. Shen, J. Xin, and A. Zhou. Finite element computation of KPP front speeds in cellular and cat's eye flows. *Journal of Scientific Computing*, 55(2):455–470, 2013.

[40] R. Sinkhorn. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

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

[42] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[43] C. Villani. *Topics in optimal transportation*, volume 58. American Mathematical Soc., 2021.

[44] Y. Wang, S. Cheung, E. Chung, Y. Efendiev, and M. Wang. Deep multiscale model learning. *arXiv:1806.04830*, 2018.

[45] S. Wright. *Primal-dual interior-point methods*. SIAM, 1997.

[46] J. Xin. *An introduction to fronts in random media*, volume 5. Springer Science & Business Media, 2009.

[47] L. Yang, X. Meng, and G.E. Karniadakis. B-PINNS: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *J. Comp. Physics*, 425:109913, 2021.

[48] D. Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103–114, 2017.

[49] Y. Zhu and N. Zabaras. Bayesian deep convolutional encoder-decoder networks for surrogate modeling and uncertainty quantification. *Journal of Computational Physics*, 366:415–447, 2018.

[50] Y. Zhu, N. Zabaras, P. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[51] P. Zu, L. Chen, and J. Xin. A computational study of residual KPP front speeds in time-periodic cellular flows in the small diffusion limit. *Physica D*, 311:37–44, 2015.