

Generative Adversarial Networks for Physics-Informed Learning

by

Liu Yang

B.E., Tsinghua University; Beijing, China P.R., 2016

M.S., Brown University; Providence, RI, 2018

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy
in the Division of Applied Mathematics at Brown University



PROVIDENCE, RHODE ISLAND

May 2021

© Copyright 2021 by Liu Yang

This dissertation by Liu Yang is accepted in its present form
by the Division of Applied Mathematics as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date_____

George Em Karniadakis, Ph.D., Advisor

Recommended to the Graduate Council

Date_____

Paris Perdikaris, UPenn, Ph.D., Reader

Date_____

Jerome Darbon, Ph.D., Reader

Approved by the Graduate Council

Date_____

Andrew G. Campbell, Dean of the Graduate School

Vitae

Liu Yang finished his Bachelor of Engineer degree, majoring in Engineering Mechanics, in July 2016 at Tsinghua University in Beijing, China. He was awarded the Scholarship for Academic Excellence in 2013 and 2015, and graduated with the honor of Outstanding Graduate Student from Tsinghua University (10%).

He was enrolled in the Doctor of Philosophy (Ph.D.) program of Solid Mechanics at Brown University in 2016, and transferred to the Ph.D. program of Applied Mathematics at Brown University in 2017. His advisor is Prof. George Em Karniadakis. In 2018, he received a transitional Master of Science degree in Applied Mathematics. In 2021, he was awarded the David Gottlieb Memorial Award by the Division of Applied Mathematics at Brown.

Liu Yang has been teaching assistant in two courses, including Statistical Inference (Fall 2019) and Operations Research: Deterministic Models (Spring 2020). He also served as teaching assistant in Summer@ICERM 2020: Fast Learning Algorithms for Numerical Computation and Data Analysis.

Publications

1. **L. Yang**, T. Meng, and G. E. Karniadakis. Measure-conditional discriminator with stationary optimum for GANs and statistical distance surrogates. *arXiv preprint arXiv:2101.06802*, 2021.
2. **L. Yang**, C. Daskalakis, and G. E. Karniadakis. Generative ensemble regression: learning particle dynamics from observations of ensembles with physics-informed deep generative models. *arXiv preprint arXiv:2008.01915*, 2020.
3. **L. Yang**, X. Meng, and G. E. Karniadakis. B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data. *Journal of Computational Physics*, 425, 109913, 2021.
4. D. Fan, **L. Yang**, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Science of the United States of America*, 117 (42) 26091-26098, 2020.
5. **L. Yang**, and G. E. Karniadakis. Potential flow generator with L_2 optimal transport regularity for generative models. *IEEE Transactions on Neural Networks and Learning Systems*, 10.1109/TNNLS.2020.3028042, 2020.
6. **L. Yang**, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1), A292-A317, 2020.

7. G. Pang, **L. Yang**, and G. E. Karniadakis. Neural-net-induced Gaussian process regression for function approximation and PDE solution. *Journal of Computational Physics*, 384, 270-288, 2019.
8. D. Zhang, **L. Yang**, and G. E. Karniadakis. Bi-directional coupling between a PDE-domain and an adjacent data-domain equipped with multi-fidelity sensors. *Journal of Computational Physics*, 374, 121-134, 2018.

Acknowledgments

First and foremost, I would like to thank my advisor, Prof. George Em Karniadakis, who brought me into this fascinating area of physics-informed learning. Apart from providing visionary directions, he also taught me to be open, be systematic, and be focused in research. As my advisor, his help extends beyond valuable academic guidance. His optimistic and passionate attitude towards work and life will inspire me for a lifetime. I would also like to express my gratitude to the other members in my dissertation committee, Prof. Paris Perdikaris and Prof. Jerome Darbon, for sparing time reading my dissertation and providing valuable feedbacks.

Secondly, I would like to thank my collaborators, including Prof. Constantinos Daskalakis, Dr. Xuhui Meng, Dr. Dongkun Zhang, Ms. Tingwei Meng, Prof. Michael Triantafyllou, Dr. Dixia Fan, Dr. Zhicheng Wang, Dr. Xiali Chen, Prof. Jinqiao Duan, Dr. Guofei Pang, Dr. Sean Treichler, Dr. Thorsten Kurth, and so on. I have learned a lot when working with you. I would also like to express my gratitude to Prof. Hui Wang for carefully checking my proof.

Further, I would like to send my thankfulness to the current and former Crunch Group members, including Prof. Zhongqiang Zhang, Dr. Zhiping Mao, Dr. Lu Lu, Prof. Maziar Raissi, Dr. Apostolos Psaros, Mr. Zhen Zhang, Dr. Jone-Hoon Ahn, Mr. Enrui Zhang, Mr. Zongren Zou, Mr. Minglang Yin, Ms. Yixiang Deng,

Mr. Qian Zhang, Dr. Sheng Chen, Dr. Qiang Zheng, Dr. He Li, Dr. Zhen Li, Dr. Mengjia Xu, Dr. Yuhang Tang, Dr. Xiaoning Zheng, Prof. Fangyin Song, Prof. Xuejing Li, and so on, for helpful discussions. Also, I would like to thank all the faculty and staff members in the Division of Applied Mathematics at Brown University. I enjoyed working in this supportive department.

Moreover, I would like to thank my other friends, including Dr. Linhan Li, Dr. Yuan Liu, Mr. Changgeng He, Mr. Qinghao He, Mr. Yusi Zhang, Mr. Ming Ma, Mr. Xiaotian Zhu, Ms. Suqin Zhang, and so on.

Last but not least, I have my utmost gratitude towards my family members, especially to my parents, Longmei Qin and Zhixiang Yang. No words can express my gratitude for your endless love, support and encouragement.

The dissertation work is supported by the following grants:

- ARO, W911NF-18-1-0301;
- DOE, DE- SC0019434;
- DOE, DE-SC0019453;
- OSD/AFOSR MURI FA9550-20-1-0358;
- DARPA-AIRA, HR00111990025;
- NIH-Yale, U01 HL142518.

Contents

Vitae	iv
Acknowledgments	vii
1 Introduction	1
1.1 What is Physics-Informed Learning	2
1.2 Why Generative Adversarial Networks (GANs) for Physics-Informed Learning	4
1.3 Organization of Dissertation	6
2 Physics-Informed GANs for Stochastic Differential Equations	10
2.1 Introduction	11
2.2 Problem Setup	14
2.3 A Brief Review of GANs and WGANs	16
2.4 Methodology	20
2.4.1 Approximating Stochastic Processes with GANs	20
2.4.2 Solving Stochastic Differential Equations with PI-GANs	22
2.5 Computational Results	26
2.5.1 A Pedagogical Problem: Approximating Stochastic Processes	27
2.5.2 Forward Problem	36
2.5.3 Inverse and Mixed Problems	43
2.5.4 Multiple Groups of Training Data	45
2.6 Summary of Chapter	46
3 Generative Ensemble Regression: Learning Particle Dynamics from Observations of Ensembles with GANs	49
3.1 Introduction	50
3.2 Problem Setup	54
3.3 Methodology	56
3.3.1 Non-Interacting Particle Systems	58
3.3.2 Interacting Particle Systems	60

3.3.3	Modifications to the Distributions	62
3.4	Computational Results	63
3.4.1	1D Problems: Brownian Motion and Lévy Process	64
3.4.2	2D Problems: Various Scenarios of Observations	67
3.4.3	Higher Dimensional Problems	69
3.4.4	Interacting Particle Systems	72
3.5	Paired Observations	75
3.6	Supplementary Material	77
3.6.1	Sliced Wasserstein Distance and Comparison with WGAN-GP	77
3.6.2	A Study on the Density Estimation	81
3.6.3	Training data in 2D Problems with Various Scenarios of Ob- servations	84
3.6.4	Effect of Clipping Radius in the Interacting Particle System	84
3.6.5	Proof for Theorem 1	85
3.6.6	A Counterexample in Continuous Sample Space	92
3.7	Summary of Chapter	93
4	Potential Flow Generator with L_2 Optimal Transport Regularity for GANs	95
4.1	Introduction	96
4.2	Generative Models and Optimal Transport Map	99
4.3	Methodology	101
4.3.1	Potential Flow Formulation of Optimal Transport Map	101
4.3.2	Potential Flow Generator	103
4.3.3	Training the Potential Flow Generator	106
4.4	Computational Results	111
4.4.1	2D Problems	112
4.4.2	Image Translation Tasks	119
4.5	Summary of Chapter	125
5	Measure-Conditional Discriminator with Stationary Optimum for GANs and Statistical Distance Surrogates	127
5.1	Introduction	128
5.2	Why Stationary Target Optimum for the Discriminator	129
5.2.1	Non-Stationary Target Optimum Hurts: An Illustrative Ex- ample	130
5.2.2	Benefits of Stationary Target Optimum	133
5.3	Measure-Conditional Discriminator in GANs	136
5.4	Measure-Conditional Discriminator for Statistical Distances Surrogate	138
5.4.1	Unsupervised Training with Variational Formula	138
5.4.2	Transfer Learning with Statistical Distance Surrogate	140
5.5	Computational Results	140
5.5.1	2D Distributions	141
5.5.2	Stochastic Dynamic Inference	142
5.5.3	Surrogate Model for KL Divergence	145

5.6	Supplementary Material	146
5.6.1	Neural Network Architectures	146
5.6.2	More Results for 2D Problems	147
5.6.3	More Results for Stochastic Dynamic Inference	148
5.6.4	More Results for the Statistical Distance Surrogate	151
5.6.5	Surrogate Model for Optimal Transport Map	153
5.7	Summary of Chapter	155
6	Bayesian Uncertainty Quantification for Partial Differential Equations with Physics-Informed GANs	157
6.1	Introduction	158
6.2	B-PINNs: Bayesian Physics-Informed Neural Networks	160
6.2.1	Prior for Bayesian Physics-Informed Neural Networks	163
6.2.2	Posterior Sampling Approaches for Bayesian Physics-Informed Neural Networks	165
6.2.3	Computational Results	169
6.2.4	Comparison with PINNs	190
6.2.5	Comparison with the Truncated Karhunen-Loève Expansion	192
6.2.6	Supplementary Material	197
6.3	Physics-Informed GANs for Priors in Bayesian Uncertainty Quan- tification	204
6.3.1	Learning Priors with Physics-Informed GANs	205
6.3.2	Different Generator Architectures for Learning Priors	208
6.3.3	Why the Prior Learned by Wasserstein GANs Fails for Pos- terior Estimation	214
6.3.4	Improve the Learned Priors with Gradient Penalty for Gen- erators	215
6.3.5	Supplementary Material	218
6.4	Summary of Chapter	220
7	Conclusion	222
7.1	Summary	223
7.2	Other Generative Models	224
7.3	Open Problems and Future Work	226

List of Tables

2.1	Forward problem: correlation coefficient between the generated random processes. The mean and two standard deviations are calculated from the selected 33 generators for each case. The reference for $C(k, f)$ comes from the assumption that $k(x; \omega)$ and $f(x; \omega)$ are independent, while the reference for $C(k, u)$ is calculated from Equation 2.21 with 1×10^5 independent samples of $k(x; \omega)$ and $u(x; \omega)$ paths.	40
4.1	Comparison between different generators on two problems	114
4.2	Comparison between discrete regularized OT solver and C-PFG on two problems	117
6.1	1D diffusion-reaction system with nonlinear source term: Predicted mean and standard deviation for k using different uncertainty quantification methods. The exact value for k is 0.7.	183
6.2	1D diffusion-reaction system with nonlinear source term: Predicted mean and standard deviation for k using different priors with exact value $k = 50$	186
6.3	1D diffusion-reaction system with nonlinear source term: Predicted mean and standard deviation for exact $k = 0.7$ and $k = 100$, using the prior $\log(k) \sim \mathcal{N}(0, 1^2)$	186
6.4	2D nonlinear diffusion-reaction system: Predicted mean and standard deviation for the reaction rate k using different uncertainty-induced methods. $k = 1$ is the exact solution.	188
6.5	Source inversion problem: Predicted mean and standard deviation for the locations using different uncertainty-induced methods. The exact locations for three sources are: $x_{c,1} = (0.3, 0.3)$, $x_{c,2} = (0.75, 0.75)$, and $x_{c,3} = (0.2, 0.7)$	190
6.6	1D diffusion-reaction system with nonlinear source term (KL): Predicted mean and standard deviation for k using KL. The exact value for k is 0.7.	195

List of Figures

1.1	Schematic to illustrate three possible categories of physical problems and associated methods.	2
2.1	Schematic to illustrate three possible categories of physical problems and associated available data. We use the term “PHYSICS” to imply the known physics for the target problem.	12
2.2	Evolution of generated distributions P_g for two different input dimensions and two different target distributions during training. The red dots are samples from P_g and the black dots are samples from P_r . We visualize the distributions in $\mathbb{R}^{10}$ by plotting sample projections in the first two dimensions. (a) The support of P_r is a hypercube, $Q_z = N(0, I_{10})$. (b) The support of P_r is a hypercube, $Q_z = N(0, 1)$. Notice that the support of P_g is always a curve, but it gets increasingly twisted in filling the hypercube and minimizing $W_1(P_r, P_g)$. (c) The support of P_r is a curve, $Q_z = N(0, I_{10})$. Notice that P_g gets increasingly concentrated in fitting the curve and minimizing $W_1(P_r, P_g)$. (d) The support of P_r is a curve, $Q_z = N(0, 1)$	19
2.3	Schematic of solving SDEs with PI-GANs. The round corner rectangles in the blue and yellow boxes represent feed forward neural networks. The parallelograms in the green box represent the neural networks induced by operators (N_x and B_x). The ellipses represent snapshots from sensors, and the gray arrows represent sampling procedures. The bracket in the yellow box represents concatenation.	23
2.4	Sample paths of Gaussian processes with the sensor locations denoted by the vertical dashed lines. We use the correlation length $l = 1$ (left), 0.5 (middle), 0.2 (right), and 6 sensors (top) and 11 sensors (bottom).	28
2.5	W_1 distances $W_1(\hat{P}_g^n, \hat{P}_t^n)$ versus training steps, for different correlation lengths and number of sensors. The decay of W_1 distances becomes slow after about 2×10^4 steps and hard to see after 8×10^4 steps. We set the number of sample snapshots for empirical distribution to be $n = 1000$. The means and two standard deviations are from 3 independent runs and 10 batches of n generated snapshots and training snapshots in each run.	29

2.6	Spectra of the correlation structure for the generated processes of different correlation lengths with the squared exponential kernel. The training sets consist of the processes where the training snapshots are collected from, and the reference is calculated from 1×10^5 independent sample paths. The means and two standard deviations are calculated from the selected 33 generators for each case; the standard deviations are very small.	30
2.7	Spectra of the correlation structure for the generated processes for different correlation lengths with the exponential kernel. The training sets consist of the processes where the training snapshots are collected from, and the reference solution is calculated from 1×10^5 independent sample paths. The means and two standard deviations are calculated from the selected 33 generators for each case; the standard deviations are very small.	31
2.8	Scatter plot of covariance of the generated processes evaluated at randomly selected pairs of coordinates, for both squared exponential kernel and the exponential kernel. Each blue or orange dot represents the covariance of generated random processes with different structures, evaluated at a randomly selected pair of points (x, x') . The black and purple lines indicate the reference covariance from the exact expressions of the squared exponential kernel and the exponential kernel, respectively.	32
2.9	Mean and standard deviation of random processes generated by WGAN-GP and vanilla GANs: For case 1, both versions of GANs perform well, but the vanilla GANs failed in case 2, while the WGAN-GP can still provide accurate results. The vertical dashed lines represent the positions of sensors.	33
2.10	Overfitting of the discriminator <i>and</i> generator. (a) Negative discriminator loss on the training set and validation set versus the training steps in three independent runs. (b) W_1 distances $W_1(\hat{P}_g^n, \hat{P}_t^n)$ and $W_1(\hat{P}_g^n, \hat{P}_v^n)$ versus the training steps in three independent runs. The means (markers) and two standard deviations (vertical lines on the marker) come from ten groups of $(\hat{P}_g^n, \hat{P}_t^n, \hat{P}_v^n)$. The black horizontal line is the empirical expectation of $W_1(\hat{P}_{r1}^n, \hat{P}_{r2}^n)$ from 50 groups of $(\hat{P}_{r1}^n, \hat{P}_{r2}^n)$	34
2.11	Forward problem: the sample paths and sensor locations of k , u , and f . The vertical dashed lines denote the locations of sensors.	36
2.12	Forward problem for different input noise dimensionality (left part of the panel) and the number of training snapshots (right part of the panel): relative errors of inferred mean and standard deviation of the stochastic solution $u(x; \omega)$. The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. The grey bars and the black lines represent the expectation and two standard deviations of the relative error if we calculate the mean and standard deviation of $u(x; \omega)$ from a random draw of 1000 $u(x; \omega)$ sample paths. The black bar (before the vertical dashed line) is the relative error from stochastic collocation method with 630 collocation points. The reference solutions are calculated from 1×10^5 $u(x; \omega)$ sample paths.	38

2.13	Forward problem: spectra of generated right-hand-side $f(x; \omega)$ processes. (a): Varying input noise dimension for a fixed number of snapshots at 1000. (b): Varying the number of training snapshots for a fixed dimension of input noise at 20. The inset plots are the zoom-ins of the eigenvalues of high frequency modes. The means (markers) and two standard deviations (vertical lines on the markers) are calculated from the selected 33 generators. The reference curves are calculated from another 1×10^5 independent $f(x; \omega)$ sample paths.	39
2.14	Forward problem with stochastic right-hand-side process $f(x; \omega)$ of high dimensionality. Figure (a) and (b) correspond to case 1 where the effective dimension is 30, while figure (c) and (d) correspond to case 2 where the effective dimension is about 120. Left: relative errors of inferred mean and standard deviation of $u(x; \omega)$. The red bars and their associated black lines represent the mean and two standard deviations of relative errors from the selected generators. The grey bars and their associated black lines represent the expectation and two standard deviations of relative errors calculated from 1×10^4 sample paths of $u(x; \omega)$. Right: the spectra of generated process versus the reference values. The means (red markers) and two standard deviations (red vertical lines on the markers) are calculated from the selected generators. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $u(x; \omega)$ and $f(x; \omega)$	42
2.15	Relative errors of inferred mean and two standard deviations for both $k(x; \omega)$ and $u(x; \omega)$ in inverse problem (blue left most bars), mixed problems (green and purple bars) and forward problem (pink bars). The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. Also shown on the right part of the panel are two reference cases, the green shaded bars corresponding to NN-aPC method [148] for the case 2 on the left, and the grey bars and their associated black lines corresponding to the expectation and two standard deviations of relative error calculated from 1000 independent sample paths of $k(x; \omega)$ or $u(x; \omega)$. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $k(x; \omega)$ and $u(x; \omega)$	44
2.16	Relative errors of inferred mean and two standard deviations for both $k(x; \omega)$ and $u(x; \omega)$ using 1 group of data (pink) and 2 groups of data (light green). The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $k(x; \omega)$ and $u(x; \omega)$	46
3.1	Schematic showing the two paradigms of point-regression and ensemble-regression for dynamic inference. In point-regression problems (black labels), we aim to fit the point coordinates (blue dots) from data with the inferred curve (orange curve), determined by the initial coordinate x_0 as well as the ODE $dx_t/dt = f_\theta(x_t, t)$, where f_θ is a function. As an analogue, in ensemble-regression problems (red labels), we aim to fit the distributions of the ensemble in the snapshots (blue dots) with the inferred curve (orange curve), determined by the initial distribution ρ_0 as well as the PDE $\partial \rho_t / \partial t = L_\theta(\rho_t, t)$, where L_θ is an operator.	51

3.2	Schematic of the generative model for ensemble-regression. We first use a feed-forward neural network to map the input noise to the output $\tilde{\mathbf{X}}_0 \in \mathbb{R}^d$, whose distribution $\tilde{\rho}_0$ is intended to approximate the initial distribution ρ_0 . Subsequently, we apply the discretized ODE or SODE with trainable parameters to generate particle trajectories $\tilde{\mathbf{X}}_t$ for $t > 0$ with $\tilde{\mathbf{X}}_0$ as the initial condition (brown curves). Differences between the distributions of $\tilde{\mathbf{X}}_t$ and the snapshots from data are quantified as (a part of) our loss function. For interacting particle systems, a neural network is employed as the velocity surrogate to generate the particle trajectories. Another loss function term, namely the Newton loss L_{Newton} , is defined to quantify the consistency between the particle accelerations derived from the material derivatives, and the forces calculated from the particle interactions. With L_{Newton} , we enforce the inferred velocity to be consistent with our partial knowledge of the dynamics.	57
3.3	Results for 1D problems with (a) Brownian Motion and (b) Lévy process as the stochastic term. The two rows of (b) show the results with (top row) and without (bottom row) bounded map pre-processing, respectively. The first column visualizes densities estimated from training data. The second column shows the inferred drift functions in the end of training. The third column shows the inferred coefficient for the stochastic term during training. The fourth and fifth columns show the inferred densities in the two cases of using a cubic polynomial and a neural network to parameterize the drift, respectively. The solid lines and shaded areas refer to the mean and two standard deviations of three runs with different data.	64
3.4	Inferred drift and diffusion in 2D problems, with (a) ideal observations, (b) noisy observations, and (c) truncated observations. In (a), the left column shows the results in case 1 where the drift is parameterized by four parameters, while the right column shows the results in case 2 where the drift is parameterized by a neural network. In the top right figure of (a), we visualize the drift field where the training data are dense. The red and black arrows represent the inferred drift and the exact drift, respectively. For each arrow, the length represents the norm of the drift, scaled by 0.1. The green dots are samples from the merged training data.	68
3.5	Inferred drift in the end of training and diffusion parameters during training, in 5D, 10D, 50D and 100D problems. The solid lines and shaded areas are the mean and two standard deviations over all i and three runs with different random seeds.	71
3.6	Results for the 1D and 2D problems of interacting particle systems. (a): The inferred alpha in 1D and 2D problems. (b): The inferred velocity field against the reference velocity from simulation in the 1D problem. The dots also show the distribution of training data. (c): The inferred velocity field (red arrows) against the reference velocity from simulation (black arrows) in the 2D problem. For each arrow, the length represents the norm of the velocity scaled by 0.3. The green dots are random samples from training data.	73
3.7	Inferred drift function in the end of training and diffusion coefficient during training for the OU process problem, using a linear function or a neural network to parameterize the drift function.	76

3.8	Inferred drift and diffusion coefficient during the training, using (a) sliced Wasserstein distance and (b) WGAN-GP. The black dashed lines represent the ground truth, while the multiple colored lines represent the results in three independent runs.	78
3.9	Results of 2D, 5D, 10D and 20D problems, using (a) the squared sliced Wasserstein-2 distance or (b) WGAN-GP for d. The solid lines and shaded areas represent the mean and two standard deviations of the errors over all dimensions.	80
3.10	Integrated squared errors (ISE) of the inferred density functions at different time instants in (a) case 1 and (b) case 2. The squared errors are integrated from -4 to 4. The colored lines show the ISE averaged over three independent runs, while the markers with the corresponding colors show the ISE in each run.	83
3.11	Samples from the training data in 2D problems with ideal, noisy, or truncated observations.	84
3.12	Inferred α in the 2D interacting particle system during training, with different clipping radius $r_{\min}$	85
4.1	Schematic of generator without and with L_2 optimal transport regularity.	97
4.2	Comparison of different methods: vanilla generator with L_2 transport cost penalty, discrete potential flow generator (D-PFG), continuous potential flow generator (C-PFG), and CycleGAN in (a) problem 1 and (b) problem 2. The red and blue arrows represent the map of generators, the black arrows represent the analytical optimal transport map. . . .	114
4.3	Potential flow generator in (a) WGAN-GP and (b) continuous normalizing flow for different problems with (middle row) first-order Euler scheme and (bottom row) second-order midpoint scheme. Each column shows the setup and results of one problem. The top row shows the samples or the unnormalized density functions of μ (purple) and ν (orange), the other rows show the map estimated by potential flow generator G and samples of $G_{\#}\mu$	118
4.4	Potential flow generator on the MNIST. In each row, the top images are the inputs, while the bottom images are the corresponding outputs. . .	120
4.5	Potential flow generator on the MNIST (a) training and (b) testing datasets. In each row, the top images are reconstructed from the input vectors, while the bottom images are reconstructed from the corresponding output vectors.	121
4.6	Comparison between our method, vanilla WGAN-GP, CycleGAN with WGAN-GP as GAN loss and L_1 or L_2 loss as cycle-consistency loss, as well as official CycleGAN. The first two rows are the original images and their projections on the 700-dimensional Euclidean space induced by PCA, which are similar. The next five rows are the corresponding outputs. The last three rows are the reconstructed images from different CycleGANs.	122
4.7	Potential flow generator on the CelebA (a) training and (b) testing datasets. In each row, the top images are reconstructed from the input vectors, while the bottom images are reconstructed from the corresponding output vectors.	124

5.1	Results for the illustrative example of oscillation. (a): vanilla discriminator using gradient descent with learning rate 0.01. (b-d): vanilla discriminator using optimistic mirror descent, with learning rate 0.1, 0.01, and 0.001, respectively. Note that the multiplications of learning rate and training iterations are kept the same. (e): measure-conditional discriminator using gradient descent with learning rate 0.01. The red and blue lines show the generator and discriminator parameters in both two dimensions, while the black horizontal lines represent the ground truth for the generator parameters.	131
5.2	Comparison between the vanilla discriminator and measure-conditional discriminator D_{mc} (ours) in three 2D problems. Each row represents the results for one problem. (a): The three target distributions. (b-g): $\widehat{W}_1(P, Q)$ against generator iterations, using different versions of GANs, discriminator/generator iteration ratios, and (β_1, β_2) in Adam optimizer. (b): Vanilla GAN, 1:1, (0.5, 0.9), (c): Vanilla GAN, 1:1, (0.9, 0.999), (d): WGAN-GP, 1:1, (0.5, 0.9), (e): WGAN-GP, 1:1, (0.9, 0.999), (f): WGAN-GP, 5:1, (0.5, 0.9), (g): WGAN-GP, 5:1, (0.9, 0.999). The y-axes are shared for each row, and the black dashed lines represent $\widehat{W}_1(Q, Q)$	141
5.3	Comparison between different set-ups in the task of stochastic dynamic inference. (a): WGAN-GP, vanilla discriminator, Adam optimizer, (b): WGAN-GP, vanilla discriminator, Optimistic Adam optimizer, (c): WGAN-GP, D_{mc} , Adam optimizer, (d): WGAN-GP, D_{sr} , Adam optimizer, (e): BGAN with Adam optimizer. First row: the absolute error of the inferred dynamic parameters $\{a_i\}_{i=0}^3$ and σ against generator iterations. Second row: the generated distributions at $t = 0.2, 0.5, 1.0$, in the end of the training, with the dashed black lines showing the ground truth.	143
5.4	Comparison between the surrogate model and direct calculation for the KL divergence. For each line, the sample size, i.e., n_1 and n_2 in Equation 5.18, increases from 1000 to 10000, to improve accuracy in the cost of FLOPs.	146
5.5	More results for the comparison between the vanilla discriminator and measure-conditional discriminator (ours) in three 2D problems. (a): The three target distributions. (b): Vanilla GAN, 5:1, (0.5, 0.9), (c): Vanilla GAN, 5:1, (0.9, 0.999). See a more detailed caption in Figure 5.2	148
5.6	Comparison between different set-ups in the task of stochastic dynamic inference. (a): WGAN-GP, D_{mc} , Optimistic Adam optimizer, (b): WGAN-GP, D_{sr} , Optimistic Adam optimizer. See a more detailed caption in Figure 5.3.	149
5.7	Comparison between different set-ups in the task of stochastic dynamic inference, using smaller discriminator neural networks. (a): WGAN-GP, vanilla discriminator, Adam optimizer, (b): WGAN-GP, vanilla discriminator, Optimistic Adam optimizer, (c): WGAN-GP, D_{mc} , Adam optimizer, (d): WGAN-GP, D_{mc} , Optimistic Adam optimizer. See a more detailed caption in Figure 5.3.	150

5.8	Results for the KL divergence surrogate model. (a-d): Inference against the ground truth. (a): 2D, 1000 samples, (b): 2D, 10000 samples, (c): 3D, 1000 samples, (d): 3D, 10000 samples. (e-f) Generator parameters during the GAN training with different generator and discriminator set-ups. (e): The first generator set-up, with 4 degrees of freedom. (f): The second generator set-up, with 5 degrees of freedom. Different colors represent different generator parameters, while different line styles represent the results from different discriminator setups and the ground truth.	152
5.9	Results of the surrogate model for optimal transport maps between 16 pairs of Gaussian distributions. The red arrows represent the barycentric projection maps given by the surrogate model $D_{sr,G}$, while the black arrows represent the reference optimal transport maps from linear programming.	154
6.1	Schematic of the Bayesian physics-informed neural network (B-PINN). $P(\theta)$ is the prior for hyperparameters as well as the unknown terms in PDEs, $P(\mathcal{D} \theta)$ represents the likelihood of observations (e.g., u , b , f), and $P(\theta \mathcal{D})$ is the posterior. The blue panel represents the Bayesian neural network while the green panel represents the physics-informed part.	159
6.2	Comparison between the B-PINNs' prior distributions and Gaussian distributions. The red lines and histograms represent the density of B-PINNs' outputs (a) $\tilde{u}(x)$, (b) $d\tilde{u}(x)/dx$, and (c) $d^2\tilde{u}(x)/dx^2$ at $x = 0, 0.5$, and 1. The black lines are the density functions of the corresponding Gaussian distributions with zero mean and the same standard deviations as the B-PINNs' outputs.	164
6.3	Function approximation and comparison among different approaches. (a) BNN-GPR, (b) BNN-HMC, (c) BNN-VI, (d) Dropout with drop rate 1%, (e) Dropout with drop rate 5%, (f) Dropout with drop rate 20% (4 hidden layers with 100 neurons per layer).	172
6.4	Function approximations for cases with larger noise. (a) $\epsilon_u \sim \mathcal{N}(0, 0.3^2)$. (b) $\epsilon_u \sim \mathcal{N}(0, 0.5^2)$	173
6.5	1D linear Poisson equation - forward problem: predicted u and f from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	174
6.6	1D linear Poisson equation: Application of active learning. (a) $N = 8$, (b) $N = 12$, (c) $N = 16$, (d) $N = 18$. N represents the number of training samples for f . Blue circle: Training samples at current step. Magenta plus: Added training sample for the next step.	176
6.7	1D flow through porous media with boundary layer: predicted u from different methods with two data noise scales. (a) $\epsilon_g \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_g \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	178
6.8	1D nonlinear Poisson equation - forward problem: predicted u and f from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	179
6.9	2D Reaction-Diffusion equation - forward problem: Predicted errors and standard deviations for u from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	181

6.10	1D diffusion-reaction system with nonlinear source term - inverse problem: predicted u and f from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	184
6.11	2D nonlinear diffusion-reaction system: Training data for u and f . (a) Distribution of u . Black circle: training sample for u , (b) Distribution of f . Black cross: training samples for f	187
6.12	1D diffusion-reaction system with nonlinear source term (PINNs): Predicted u and f with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	191
6.13	1D nonlinear Poisson equation (KL) - forward problem: Predicted u and f with two data noise scales. (a): $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b): $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	194
6.14	1D diffusion-reaction system with nonlinear source term (KL): Predicted u and f with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$	195
6.15	Covariance functions $k(x_1, x_2) = \text{cov}(\tilde{u}(x_1), \tilde{u}(x_2))$ for BNNs with different architectures. $L = 2, \sigma_{b,l} = 1$ for $l = 0, 1, 2$ in all the cases. (a) $N_1 = N_2 = 20$, $\sigma_{w,0} = 1.0$, $\sigma_{w,1} = \sigma_{w,2} = \sqrt{5/2}$ (b) $N_1 = N_2 = 50$, $\sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$, (c) $N_1 = N_2 = 100$, $\sigma_{w,0} = 1.0$, $\sigma_{w,1} = \sigma_{w,2} = \sqrt{1/2}$. (d) $N_1 = N_2 = 20$, $\sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$, (e) $N_1 = N_2 = 100$, $\sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$	197
6.16	BNN-HMC with different priors for function approximation. $L = 2, \sigma_{b,l} = 1$ for $l = 0, 1, 2$ in all the cases. (a) $N_1 = N_2 = 20$, $\sigma_{w,0} = 1.0$, $\sigma_{w,1} = \sigma_{w,2} = \sqrt{5/2}$, (b) $N_1 = N_2 = 100$, $\sigma_{w,0} = 1.0$, $\sigma_{w,1} = \sigma_{w,2} = \sqrt{1/2}$, (c) $N_1 = N_2 = 20$, $\sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$, (d) $N_1 = N_2 = 100$, $\sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$	198
6.17	Space-dependent reaction rate: Predicted k , u and f using B-PINN-HMC. (a) $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, and $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, and $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$	202
6.18	Extrapolation with B-PINNs: Predicted u and f using B-PINN-HMC with a limited number of observations on u . (a) $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$	203
6.19	Target priors for u and f , as well as the learned priors with different generator architectures. (a) The target priors, the grey vertical dashed lines represent the locations of sensors. (b-d) The learned priors with the first, second, and third generator architecture, respectively. The red curves and shaded areas represent the mean and two standard deviations, while the black dashed curves represent the sample functions.	210
6.20	Posteriors for u and f , using the target priors and the learned priors with different generator architectures. (a) The posteriors using the target priors. (b-d) The posteriors using the learned priors, with the first, second, and third generator architecture, respectively. The curves and shaded areas represent the mean and two standard deviations.	211

6.21	Covariences of learned priors at x and y against the distance $ x - y $, with different generator architecture. (x, y) are randomly sampled from $[-1, 1]^2$. (a-c): The first, second, and third generator architecture, respectively. The three rows show the results with three random seeds.	212
6.22	Posteriors for u and f using the target prior and learned priors. (a) The posteriors corresponding to the target Gaussian process prior. (b-d) The posteriors estimated with the priors learned by the first, second, and third generator architecture, respectively. The three rows show the results with three random seeds. The curves and shaded areas represent the mean and two standard deviations.	213
6.23	Illustration of the generated distributions Q_n for $n = 2, 5, 20, 50$. Q_n is the uniform distributions on the graph of the function.	214
6.24	Covariences of learned priors at x and y against the distance $ x - y $, with different gradient penalty coefficients γ for the second generator architecture. (x, y) are randomly sampled from $[-1, 1]^2$. (a-c): $\gamma = 0.1, 1.0, 10.0$, respectively. The three rows show the results with three random seeds.	216
6.25	Posteriors for u and f using the learned priors, with different gradient penalty coefficients γ for the second generator architecture. (a-c) $\gamma = 0.1, 1.0, 10.0$, respectively. The three rows show the results with three random seeds. The curves and shaded areas represent the mean and two standard deviations.	217
6.26	The posterior conditioned on ten sets of observations, predicted by the deep normalizing flow as a surrogate model. The first and third row are for u , while the second and forth row are for corresponding f . The red lines and shaded areas represent the mean and two standard deviations.	219

CHAPTER ONE

Introduction

1.1 What is Physics-Informed Learning

Based on how much information we have for the physical laws and data, physical problems and associated methods can be classified into three categories, shown in Figure 1.1. In the first category, i.e., the small data regime, we assume that we know lots of physics, usually in the form of partial differential equations (PDEs), while data are provided for the initial and boundary conditions of a PDE. The classic PDE solvers like the finite element method, finite difference method, lie in this category. At the opposite extreme is the big data regime, where we may have little or no knowledge of physics. In such regime, a pure data-driven learning approach may be the most effective. The most ubiquitous regime in applications is the middle one, where we have some data and some physics, possibly missing some parameter values or even an entire term in the PDE. This is the regime where we need physics-informed learning, aiming to integrate data and partial knowledge of the governing physical laws in a unified way.

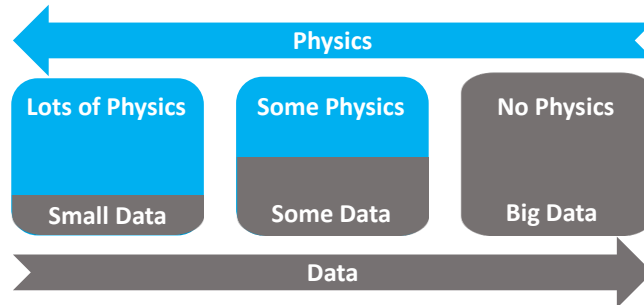


Figure 1.1: Schematic to illustrate three possible categories of physical problems and associated methods.

While there have been other physics-informed learning frameworks, e.g., kernel methods like Gaussian processes for solving PDEs [104, 108], methods based on deep neural networks for physics-informed learning, e.g., [44, 110], have recently attracted increasingly more attention, due to their ability to tackle high dimensional, nonlinear PDE problems. There are various ways to incorporate the

partial knowledge of physics into the deep learning system. Here we introduce two approaches that are used in this dissertation:

- Employ neural networks with specially designed architectures so that the neural network functions satisfy some hard constraints. For example, we can use the gradient of a neural network to approximate a curl-free field. Other examples in literature include using the determinant function to obtain anti-symmetry under the exchange of input variables [98], and use summation/averaging operations to obtain invariance under permutation in many-body systems [151].
- Design loss functions to apply soft constraints of physical laws, for example, the residual of PDE in the strong form [110], the energy based on variational form of the PDE [132], etc. Compared with classic grid-based numerical methods where the derivatives are approximated by numerical schemes, neural networks have analytical derivatives by performing automatic differentiation, and get rid of the numerical error introduced by discretization.

In general, the incorporated physical knowledge benefits the learning algorithm in the following aspects:

- Make the results more reliable, in particular when the neural network model is specially designed. The soft penalty in the physics-informed loss could also act as a regularization term to avoid overfitting.
- Integrate and leverage the data collected from different sources or at different time instants, which are intrinsically connected via physical laws.
- Reduce the number of data required in the learning algorithm. For example,

if we use our knowledge of the PDE and data for the source term, we may only need the data at the boundary to learn the solution. The built-in symmetry and invariance can also reduce the data requirement, and possibly accelerate the training.

1.2 Why Generative Adversarial Networks (GANs) for Physics-Informed Learning

The aim of generative adversarial networks (GANs) is to train a generator that takes noise as input and outputs “fake” samples, and we wish the generated samples to be able to approximate real data distributions. GANs deal with this problem by introducing two neural networks, namely the generator G and the discriminator D , and train them in an adversarial way.

In general, there are two perspectives for GANs. The first perspective is to view GANs as a two-player zero-sum game between the generator G and the discriminator D . The discriminator takes a sample as input and aims to tell if it is “real”, i.e., sampled from the data set, or “fake”, i.e., generated by the generator. Meanwhile, the generator tries to “deceive” the discriminator by mimicking the true data distribution. The hope is that the iterative adversarial training of G and D will lead to the Nash equilibrium of this zero-sum game, where the generated distribution will be identical to the target distribution of real data. The second perspective is that the discriminator gives the distance between the generated distribution and the target distribution in a variational form. For example, vanilla

GANs can be formulated as:

$$\begin{aligned}
& \min_G \max_D V(G_{\#}\mathcal{N}, D), \\
V(G_{\#}\mathcal{N}, D) &= \mathbb{E}_{x \sim G_{\#}\mathcal{N}} \log(1 - D(x)) + \mathbb{E}_{x \sim Q} \log(D(x)) \\
&= \mathbb{E}_{z \sim \mathcal{N}} \log(1 - D(G(z))) + \mathbb{E}_{x \sim Q} \log(D(x)),
\end{aligned} \tag{1.1}$$

while Wasserstein GANs (WGANs) can be formulated as:

$$\begin{aligned}
& \min_G \max_{D \text{ is 1-Lipschitz}} V(G_{\#}\mathcal{N}, D), \\
V(G_{\#}\mathcal{N}, D) &= -\mathbb{E}_{x \sim G_{\#}\mathcal{N}} D(x) + \mathbb{E}_{x \sim Q} D(x) \\
&= -\mathbb{E}_{z \sim \mathcal{N}} D(G(z)) + \mathbb{E}_{x \sim Q} D(x),
\end{aligned} \tag{1.2}$$

where Q represents the target distribution, $\mathcal{N}$ represents input noise, $\#$ is the push-forward operator, thus $G_{\#}\mathcal{N}$ represents the generated distribution. In vanilla GANs and WGANs, $\max_D V(G_{\#}\mathcal{N}, D)$ and $\max_{D \text{ is 1-Lipschitz}} V(G_{\#}\mathcal{N}, D)$ are nothing but the Jensen-Shannon (JS) divergence and the Wasserstein-1 distance up to constants between $G_{\#}\mathcal{N}$ and Q , respectively.

GANs have achieved remarkable success within a short time for diverse tasks of generating synthetic data, such as images [11, 50, 63, 155], texts [146, 153, 32, 69], and even music [146, 87, 141, 42]. The ability of GANs in learning probability distributions from data suggests its potential application in physics-informed learning for stochastic physical and biological systems, which motivates this dissertation. We list the specific attributes of GANs for such tasks as follows:

- Instead of artificially designing a form to parameterize the random variables or stochastic processes etc., GANs can learn the parameterization from data, significantly reducing human effort. For example, we don't need a clear

knowledge of the hidden stochastic dimensions in advance. This is because GANs map the input noise to the output distribution in a nonlinear way. If the input noise dimension is lower than the target one, the generator can approximate the target distribution by “twisting” the manifold, where the generated distribution is concentrated on. If the input noise dimension is higher than the target one, the generator map can be in a “contracted” way so that the generated distribution is concentrated on a low dimensional manifold.

- GANs can tackle high dimensional stochastic problems. The dimension here could refer to that of the hidden random variable, e.g. in Chapter 2 we tackle a stochastic differential equation with over 100 stochastic dimensions, or that of the physical space, e.g., in Chapter 3 we tackle a stochastic ordinary differential equation with 100 physical dimensions.
- GANs can extract information from the “ensemble” instead of individual samples and leverage unpaired data. An example in the context of image translation is CycleGAN [155], where the GANs learn how to translate images into other styles, with unpaired images as data. In this example, the GANs learn the information of image styles from the ensemble. In the context of physics-informed learning, GANs can also extract the information from the distributions of observed particles or sensor outputs.

1.3 Organization of Dissertation

This dissertation is organized as follows. The chapters are written in a self-contained way, so that the readers can start from any of them.

In Chapter 2 we introduce physics-informed GANs (PI-GANs) to solve forward, inverse and mixed problems of stochastic differential equations (SDEs) in a unified manner based on limited number of scattered measurements. Unlike standard GANs relying solely on data for training, here we encoded into the architecture of GANs the governing physical laws in the form of SDEs using automatic differentiation. We solved forward, inverse, and mixed problems without changing the framework of PI-GANs, obtaining both the means and standard deviations of the stochastic solution and the diffusion coefficient in good agreement with ground truth.

The stochastic systems tackled in Chapter 2 are time-independent. In Chapter 3 we extend the application of GANs to stochastic dynamic inference of particle systems governed by stochastic ordinary differential equations (SODEs) with Brownian motions and Lévy processes. By training a physics-informed generative model that generates “fake” sample paths, we aim to fit the observed particle distributions with a curve in the probability measure space, which is induced from the inferred particle dynamics. Here the differences between distributions can be quantified by the adversarial losses in GANs, especially in high dimensional problems. We name this method as generative “ensemble-regression” (GER), in analogy to the classic “point-regression”, where we infer the dynamics by performing regression in the Euclidean space. We illustrate the GER by learning the drift and diffusion terms of SODEs with Brownian motions and Lévy processes up to 100 dimensions. We also discuss how to treat cases with noisy or truncated observations.

Apart from systems consisting of independent particles, in Chapter 3 we also tackle nonlocal interacting particle systems with unknown interaction parameters. Specifically, we employ a neural network as a surrogate of the velocity field in the

spatio-temporal domain. We introduce a physics-informed loss function based on the differences between the accelerations calculated by material derivative and particle interactions in the generated particle system. By doing so, we enforce the velocity surrogate to be consistent with our partial knowledge of dynamics and the learned interaction parameters.

The idea of employing neural networks as velocity surrogates in generators, and using loss functions augmented to the adversarial losses to apply additional constraints to the generators, can be extended to a wide range of applications. In Chapter 4 we propose a potential flow generator with L_2 optimal transport regularity, which can be easily integrated into different versions of GANs. The generator not only tries to transport the input distribution to the target one, but also aims to find the one with minimum L_2 transport cost. The loss function for training the generator is the original adversarial loss augmented with the residual of a Hamilton-Jacobi equation, which is the optimality conditions for the transport map. We show that the potential flow generator can be applied in image translation tasks, e.g., translating a no-smiling faces to smiling faces belonging to the same people, using unpaired training data.

While GANs proved to be powerful for high dimensional distributions, in Chapter 3 we also observed that it could fail for low dimensional problems. In Chapter 5 we argue that a possible underlying reason is that the target optimum function for the discriminator varies as the generator evolves, and propose the measure-conditional discriminator with stationary optimum as a plug-and-play module for different GANs. We show that compared with the vanilla discriminator, the measure-conditional discriminator is more robust w.r.t. the GAN losses, discriminator/generator iteration ratios, and optimizer hyperparameters, and significantly outperforms the vanilla one in generative ensemble regression. More-

over, a variant of measure-conditional discriminator can also handle multiple target distributions, perfectly matching the scenario of generative ensemble regression, or act as a surrogate model of statistical distances such as KL divergence.

Finally, we present the application of GANs in Bayesian uncertainty quantification in Chapter 6. In the first part of the chapter, we introduce Bayesian physics-informed neural networks (B-PINNs) to solve linear or nonlinear PDEs with noisy data for both forward and inverse problems. The uncertainties arising from the scattered noisy data could be naturally quantified in the Bayesian framework [75]. The choice of priors for B-PINNs may have a significant influence on the posterior predictions especially in the cases with small data. Therefore, in the second part of this chapter, we show the effectiveness of physics-informed GANs in learning informative functional priors. We will also discuss why sometimes GANs fail to learn priors for posterior estimation, and how to make improvements.

CHAPTER TWO

Physics-Informed GANs for Stochastic Differential Equations

2.1 Introduction

Generative adversarial networks (GANs) have achieved remarkable success within a short time for diverse tasks of generating synthetic data, such as images [11, 50, 63, 155], texts [146, 153, 32, 69], and even music [146, 87, 141, 42]. GANs can learn probability distributions from data, an attribute suggestive of its potential application to modeling the *inherent stochasticity* and *extrinsic uncertainty* in physical and biological systems.

The specific data-driven approach to modeling physical and biological systems depends crucially on the amount of data available as well as on the complexity of the system itself, as illustrated in Figure 2.1. The classical paradigm for which many different numerical methods have been developed over the last fifty years is shown in the top plot of Figure 2.1, where we assume that the only data available are the boundary and initial conditions while the specific governing partial differential equation (PDE) and its associate parameters are precisely known. On the other extreme (lower plot), we may have a lot of data, e.g. in the form of time series, but we may not know the governing physical law, e.g. the underlying PDE, at the continuum level; many problems in social dynamics fall under this category although work so far has focused on recovering known PDEs from data only, e.g. see [120, 18, 102]. Perhaps the most interesting category is sketched in the middle plot, where we assume that we know the physics partially, e.g. in an advection-diffusion-reaction system the reaction terms may be unknown, but we have several scattered measurements in addition to the boundary and initial conditions that we can use to infer the missing functional terms and other parameters in the PDE and simultaneously recover the solution. It is clear that this middle category is the most general case, and in fact it is representative of the other two

categories, if the measurements are too few or too many. This is the *mixed* case that we address in this chapter but with the significantly more complex scenario, where the solution is a stochastic process due to stochastic excitation or an uncertain material property, e.g. permeability or diffusivity in a porous medium, or uncertainty in the initial and boundary conditions. Hence, we employ stochastic differential equations (SDEs) to govern these stochastic solutions and other stochastic fields.

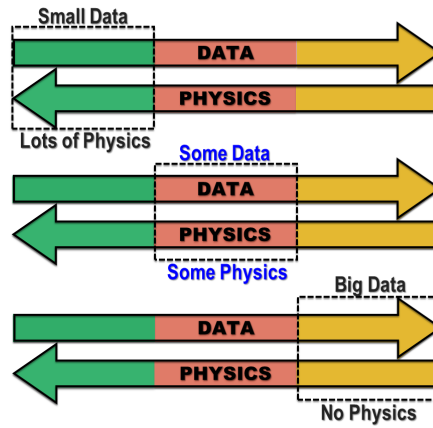


Figure 2.1: Schematic to illustrate three possible categories of physical problems and associated available data. We use the term “PHYSICS” to imply the known physics for the target problem.

Taking inspiration from the work on physics-informed neural networks for deterministic PDEs [106, 107], here we employ GANs for stochastic problems in the second category of Figure 2.1. We wish to encode the known physics, more specifically, the SDEs, into the architecture of GANs while at the same time exploit the feature of GANs to model and learn the unknown stochastic terms in the equations from data. This approach represents a seamless integration of mathematical models and data, for system inference/identification as well as mixed problems. This is an emerging paradigm in machine learning research addressing engineering applications, where typically the physics is very complex and we only have partial measurements of forcing or material properties. To this end, recent advances include using Gaussian process regression [38, 118, 13, 109, 94, 142] and

deep neural networks (DNNs) [59, 60, 52, 106, 88, 124] to solve forward problems as well as Bayesian estimation [125] and DNNs [105, 157] for inverse problems. However, to the best of our knowledge, published works are mostly dealing with deterministic systems. There have only been very few works published on data-driven methods for SDEs, e.g., [29, 101] for forward problems, and they deal with parabolic partial differential equations, backward stochastic differential equations, or forward-backward stochastic differential equations, which are different from the problems we are going to deal with in this chapter; For the inverse problem, Zhang et.al. [148] have recently proposed a DNN based method that learns the modal functions of the quantity of interest, which could also be the unknown system parameters. While robust, this method suffers from the “curse of dimensionality” in that the number of polynomial chaos expansion terms grows exponentially as the effective dimension increases, leading to prohibitive computational costs for modeling stochastic systems in high dimensions. In this chapter, we start from a new perspective, and propose the physics-informed GANs (PI-GANs) to solve SDEs. Our method is flexible in that without changing the general framework, it is capable of solving a wide range of problems, from forward problems to inverse problems and in between, i.e., mixed problems. Moreover, PI-GANs learn probability distributions from data, and do not suffer from the “curse of dimensionality”. As a result, we can tackle SDEs involving stochastic processes with high effective dimensions. In addition, PI-GANs can make use of data collected from multiple groups consisting of multiple sensors with no alignment between data in the groups.

The organization of this chapter is as follows. In Section 2.2, we set up the data-driven problems. In Section 2.3, we give a brief review of GANs and the specific version we use, namely Wasserstein GANs with gradient penalty (WGAN-GP).

Our main algorithms are introduced in Section 2.4, followed by a detailed study of the performance of our methods in Section 2.5, where we first consider the learning of stochastic processes from a limited number of realizations and discuss the currently under-studied issue of overfitting in WGAN-GP. Subsequently, we present solutions of stochastic PDEs for different types of available data and different dimensions. We conclude in Section 2.6 with a brief summary.

2.2 Problem Setup

To illustrate the main idea of PI-GANs we consider the following stochastic differential equation:

$$\begin{aligned} \mathcal{N}_x[u(\mathbf{x}; \omega); k(\mathbf{x}; \omega)] &= f(\mathbf{x}; \omega), \quad \mathbf{x} \in \mathcal{D}, \quad \omega \in \Omega, \\ B_x[u(\mathbf{x}; \omega)] &= b(\mathbf{x}; \omega), \quad \mathbf{x} \in \Gamma, \end{aligned} \tag{2.1}$$

where $\mathcal{N}_x$ is a general differential operator, $\mathcal{D}$ is the d -dimensional physical domain in $\mathbb{R}^d$, $\mathbf{x}$ is the d -dimensional spatial coordinate, Ω is the probability space, and ω is a random event. The coefficient $k(\mathbf{x}; \omega)$ and the forcing term $f(\mathbf{x}; \omega)$ are modeled as random processes, and thus the solution $u(\mathbf{x}; \omega)$ will depend on both $k(\mathbf{x}; \omega)$ and $f(\mathbf{x}; \omega)$. B_x is the boundary condition operator acting on the domain boundary Γ . As a pedagogical example, in this chapter we consider the one-dimensional elliptic SDE, which retains most of the main features of more complex SDEs:

$$\begin{aligned} -\frac{1}{10} \frac{d}{dx} \left[k(x; \omega) \frac{d}{dx} u(x; \omega) \right] &= f(x; \omega), \quad x \in [-1, 1], \\ u(-1) &= u(1) = 0, \end{aligned} \tag{2.2}$$

where $k(x; \omega)$ and $f(x; \omega)$ are independent stochastic processes, and $k(x; \omega)$ is strictly positive. For simplicity, we impose homogeneous Dirichlet boundary conditions on $u(x; \omega)$.

We consider a general scenario for Equation 2.1, where we have a limited number of measurements from scattered sensors for the stochastic processes. Specifically, we place sensors at $\{\mathbf{x}_i^k\}_{i=1}^{n_k}$, $\{\mathbf{x}_i^u\}_{i=1}^{n_u}$, $\{\mathbf{x}_i^f\}_{i=1}^{n_f}$ and $\{\mathbf{x}_i^b\}_{i=1}^{n_b}$ to collect “snapshots” of $k(\mathbf{x}; \omega)$, $u(\mathbf{x}; \omega)$, $f(\mathbf{x}; \omega)$ and $b(\mathbf{x}; \omega)$, where n_k , n_u , n_f and n_b are the numbers of sensors for $k(\mathbf{x}; \omega)$, $u(\mathbf{x}; \omega)$, $f(\mathbf{x}; \omega)$ and $b(\mathbf{x}; \omega)$, respectively. Here, one “snapshot” represents a simultaneous read of all the sensors, and we assume that the data in one snapshot correspond to the same random event ω in the random space Ω , while ω varies for different snapshots. Note that each snapshot is the concatenation of snapshots from $k(\mathbf{x}; \omega)$, $u(\mathbf{x}; \omega)$, $f(\mathbf{x}; \omega)$ and $b(\mathbf{x}; \omega)$, thus it is actually a vector of size $(n_k + n_u + n_f + n_b)$. Suppose we have a group of N snapshots, then we denote the accessible data set as $\{T(\omega^{(j)})\}_{j=1}^N$ defined as

$$\begin{aligned} \{T(\omega^{(j)})\}_{j=1}^N &= \{(K(\omega^{(j)}), U(\omega^{(j)}), F(\omega^{(j)}), B(\omega^{(j)}))\}_{j=1}^N, \\ K(\omega^{(j)}) &= (k(\mathbf{x}_i^k; \omega^{(j)}))_{i=1}^{n_k}, \\ U(\omega^{(j)}) &= (u(\mathbf{x}_i^u; \omega^{(j)}))_{i=1}^{n_u}, \\ F(\omega^{(j)}) &= (f(\mathbf{x}_i^f; \omega^{(j)}))_{i=1}^{n_f}, \\ B(\omega^{(j)}) &= (b(\mathbf{x}_i^b; \omega^{(j)}))_{i=1}^{n_b}. \end{aligned} \tag{2.3}$$

The corresponding terms in Equation 2.3 are omitted if we put no sensors for that specific process.

In this chapter, we assume that we always have a sufficient number of sensors for $f(x; \omega)$ in Equation 2.2. The type of the problems varies depending on the

number of sensors on $k(x; \omega)$ and $u(x; \omega)$: as we decrease the number of sensors on $k(x; \omega)$ while increase the number of sensors on $u(x; \omega)$, the problems gradually transform from forward to mixed and finally to inverse problems.

Moreover, we could have more than one group of snapshots of measurements. For example, in addition to the existing old sensors, we may place some new sensors, collecting and utilizing data from both the new and the old sensors. In this scenario, we have two groups of data: the newly collected data, and the previously collected data solely from the old sensors. In the case of multiple groups, our accessible data are

$$\{\{T_t(\omega^{(t,j)})\}_{j=1}^{N_t}\}_{t=1}^M = \{\{(K_t(\omega^{(t,j)}), U_t(\omega^{(t,j)}), F_t(\omega^{(t,j)}), B_t(\omega^{(t,j)}))\}_{j=1}^{N_t}\}_{t=1}^M, \quad (2.4)$$

where M is the number of groups, t is the index for the groups, and N_t is the number of snapshots in group t . The random event $\omega^{(t,j)}$ denotes the random instance of the j -th snapshot in group t . Note that the sensor setups in different groups could be different, and we use $\{\mathbf{x}_i^{k,t}\}_{i=1}^{n_{k,t}}$ to denote the position setup of $n_{k,t}$ sensors for k in group t (similarly for other terms). More importantly, snapshots from different groups could be collected independently, so that the snapshots in different groups do not have to be aligned or paired.

2.3 A Brief Review of GANs and WGANs

Before moving to our main algorithms, we briefly review GANs and WGANs. Consider the problem of learning a distribution P_r on $\mathbb{R}^d$, given data sampled from P_r . Suppose we have a DNN $G_\theta(\cdot)$ parameterized by θ that takes the random

variable $\xi \in \mathbb{R}^m$ as input and outputs a sample $G_\theta(\xi) \in \mathbb{R}^d$. The random input ξ is sampled from a prescribed distribution Q_z (e.g., uniform, Gaussian), and we denote the distribution of $G_\theta(\xi)$ as P_g . We want to approximate P_r with P_g .

GANs deal with this problem by defining a two-player zero-sum game between the generator $G_\theta(\cdot)$ and the discriminator $D_\rho(\cdot)$, which is another DNN parameterized by ρ . The discriminator takes a sample $x \in \mathbb{R}^d$ as input and aims to tell if it is sampled from P_r or P_g . Meanwhile, the generator tries to “deceive” the discriminator by mimicking the true distribution P_r . In vanilla GANs [37], the two-player zero-sum game is defined as follows:

$$\inf_{\theta} \sup_{\rho} \mathbb{E}_{x \sim P_r} [\log(D_\rho(x))] + \mathbb{E}_{\xi \sim Q_z} [\log(1 - D_\rho(G_\theta(\xi)))]. \quad (2.5)$$

Correspondingly, the loss functions for the generator and discriminator are

$$\begin{aligned} L_g &= \mathbb{E}_{\xi \sim Q_z} [\log(1 - D_\rho(G_\theta(\xi)))], \\ L_d &= -\mathbb{E}_{x \sim P_r} [\log(D_\rho(x))] - \mathbb{E}_{\xi \sim Q_z} [\log(1 - D_\rho(G_\theta(\xi)))]. \end{aligned} \quad (2.6)$$

If the discriminator is optimal, L_g measures the Jensen-Shannon (JS) divergence between P_r and P_g up to multiplication and addition by constants:

$$JS(P_r, P_g) = \frac{1}{2}KL(P_r||M) + \frac{1}{2}KL(P_g||M), \quad (2.7)$$

where $M = (P_r + P_g)/2$, and $KL(\cdot||\cdot)$ is the Kullback-Leibler divergence [37]. By training the generator and discriminator iteratively, ideally we can make P_g approach P_r in the sense of the JS divergence.

However, the JS divergence does not always provide a usable gradient for the generator, especially when the two distributions concentrate on low dimen-

sional manifolds [4]. As a consequence, training vanilla GANs is quite a delicate process and could be unstable [4]. To fix this problem, Wasserstein GANs with clips on weights (weight-clipped WGANs) [4] and WGANs with gradient penalty (WGAN-GP) [43] were proposed. Similar to vanilla GANs, the two-play zero-sum game in WGANs is formulated as

$$\inf_{\theta} \sup_{\substack{\rho \\ D_{\rho} \text{ is 1-Lipschitz}}} \mathbb{E}_{\mathbf{x} \sim P_r}[D_{\rho}(\mathbf{x})] - \mathbb{E}_{\xi \sim Q_z}[D_{\rho}(G_{\theta}(\xi))]. \quad (2.8)$$

The difference between weight-clipped WGANs and WGAN-GP is mainly on the technique of imposing the Lipschitz constraint for D_{ρ} : weight-clipped WGANs force ρ to be bounded during the training, while WGAN-GP adds a gradient penalty to the loss function for the discriminator¹. To be specific, in WGAN-GP the loss functions for the generator and discriminator are defined as

$$\begin{aligned} L_g &= -\mathbb{E}_{\xi \sim Q_z}[D_{\rho}(G_{\theta}(\xi))], \\ L_d &= \mathbb{E}_{\xi \sim Q_z}[D_{\rho}(G_{\theta}(\xi))] - \mathbb{E}_{\mathbf{x} \sim P_r}[D_{\rho}(\mathbf{x})] + \lambda \mathbb{E}_{\hat{\mathbf{x}} \sim P_{\hat{\mathbf{x}}}}[(\|\nabla_{\hat{\mathbf{x}}} D_{\rho}(\hat{\mathbf{x}})\|_2 - 1)^2], \end{aligned} \quad (2.9)$$

where $P_{\hat{\mathbf{x}}}$ is the distribution generated by uniform sampling on straight lines between pairs of points sampled from P_r and P_g , and λ is the gradient penalty coefficient.

Instead of the JS divergence in vanilla GANs, in WGANs the loss function for the generator corresponds to the Earth Mover or Wasserstein-1 distance ($\mathbb{W}_1$) between P_r and P_g :

$$\mathbb{W}_1(P_r, P_g) = \inf_{\gamma \in \Gamma(P_r, P_g)} \mathbb{E}_{(\mathbf{x}, \mathbf{y}) \sim \gamma}[\|\mathbf{x} - \mathbf{y}\|], \quad (2.10)$$

¹In [4] and [43], the discriminators are names as “critics”, but in this dissertation we still use the name of discriminators for consistency with other versions of GANs, including vanilla GANs.

where $\Gamma(P_r, P_g)$ denotes the set of all joint distributions $\gamma(x, y)$ whose marginals are P_r and P_g , respectively. W_1 distance is continuous and differentiable almost everywhere with respect to the parameters in the generator under a mild constraint [4]. As a result, WGANs do not suffer from the problem of mode collapse as would occur to vanilla GANs [4]. Moreover, according to [43], training WGAN-GP is more stable than weight-clipped WGANs.

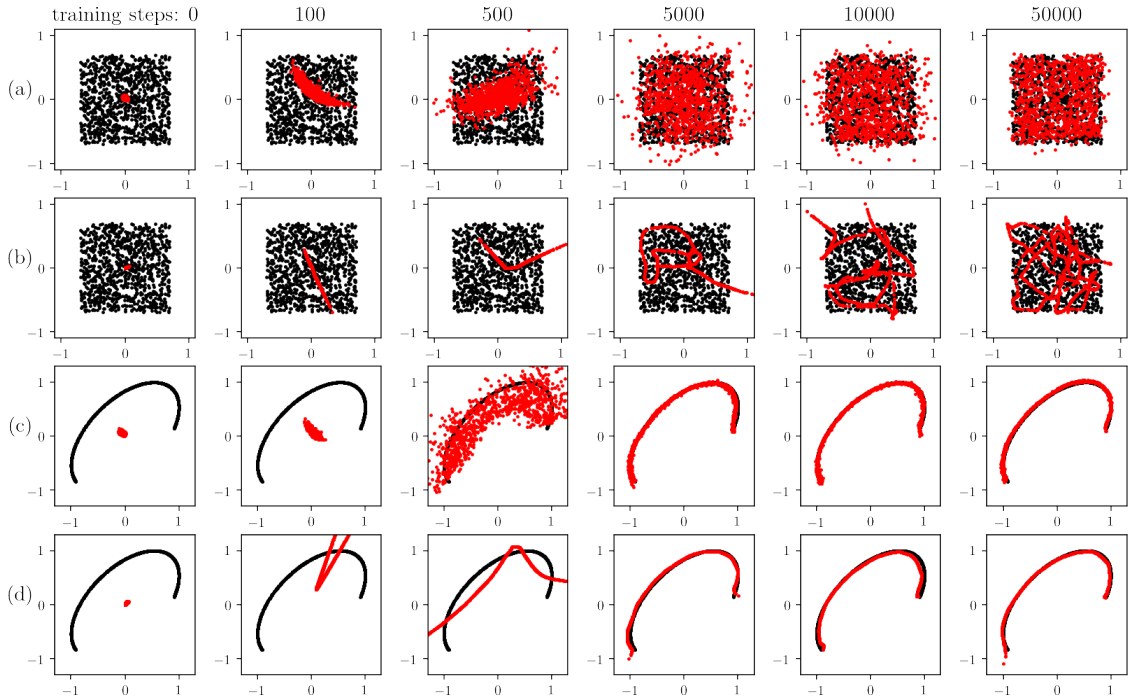


Figure 2.2: Evolution of generated distributions P_g for two different input dimensions and two different target distributions during training. The red dots are samples from P_g and the black dots are samples from P_r . We visualize the distributions in $\mathbb{R}^{10}$ by plotting sample projections in the first two dimensions. (a) The support of P_r is a hypercube, $Q_z = N(0, I_{10})$. (b) The support of P_r is a hypercube, $Q_z = N(0, 1)$. Notice that the support of P_g is always a curve, but it gets increasingly twisted in filling the hypercube and minimizing $W_1(P_r, P_g)$. (c) The support of P_r is a curve, $Q_z = N(0, I_{10})$. Notice that P_g gets increasingly concentrated in fitting the curve and minimizing $W_1(P_r, P_g)$. (d) The support of P_r is a curve, $Q_z = N(0, 1)$.

To demonstrate the effectiveness of WGAN-GP, we apply it on four toy problems of approximating distributions in $\mathbb{R}^{10}$, as illustrated in Figure 2.2. We consider two types of P_r : one is a uniform distribution on a hypercube in $\mathbb{R}^{10}$, while the other one is a uniform distribution concentrated on a curve embedded in $\mathbb{R}^{10}$. For both cases of P_r , we test with an input of either 1-D or 10-D standard Gaussian

noise, i.e., $Q_z = N(0, 1)$ or $Q_z = N(0, I_{10})$. In all four cases, the generator converges and generates samples with distribution P_g close to the real distribution P_r , even for the cases where the dimension of Q_z mismatches the support of P_r . Due to its robustness, we use WGAN-GP as our default version of GANs in this chapter.

2.4 Methodology

2.4.1 Approximating Stochastic Processes with GANs

As a pedagogical problem, let us first consider the problem of modeling a stochastic process $f(\mathbf{x}; \omega)$ on the domain $D \in \mathbb{R}^d$ with GANs. We use the total N snapshots of $f(\mathbf{x}; \omega)$ sensor data as our training set:

$$\{F(\omega^{(j)})\}_{j=1}^N = \{(f(\mathbf{x}_i; \omega^{(j)}))_{i=1}^{n_s}\}_{j=1}^N, \quad (2.11)$$

where n_s is the number of sensors and $\{\mathbf{x}_i\}_{i=1}^{n_s}$ are positions of the sensors. We use a feed forward DNN $\tilde{f}_\theta(\mathbf{x}; \boldsymbol{\xi})$ parameterized by θ as our generator to model the stochastic process $f(\mathbf{x}; \omega)$. The generator takes the concatenation of a noise vector $\boldsymbol{\xi} \sim Q_z$ from $\mathbb{R}^m$ and the coordinate $\mathbf{x}$ as the input, while the output is a real number representing $\tilde{f}_\theta(\mathbf{x}; \boldsymbol{\xi})$. Let

$$\{\tilde{F}(\boldsymbol{\xi}^{(j)})\}_j = \{(\tilde{f}_\theta(\mathbf{x}_i; \boldsymbol{\xi}^{(j)}))_{i=1}^{n_s}\}_j \quad (2.12)$$

be the generated “fake” snapshots, where $\{\boldsymbol{\xi}^{(j)}\}_j$ are instances of $\boldsymbol{\xi}$ with j as the index.

The discriminator $D_\rho(\cdot)$ is implemented as another feed forward DNN parameterized by ρ . The GAN strategy shall be applied by feeding both the real and generated snapshots into the discriminator, and train the generator and discriminator iteratively with the Adam optimizer [54] according to Equation 2.9. The detailed algorithm is presented in Algorithm 1.

Algorithm 1 GANs for approximating stochastic processes.

Require: training steps n_t , the gradient penalty coefficient λ , the number of discriminator iterations per generator iteration n_d , the batch size n , Adam hyperparameters α, β_1, β_2 , initial values θ_0 and ρ_0 for the parameters generator θ and ρ .

```

1: for  $s_t = 1, 2, \dots, n_t$  do
2:   for  $s_d = 1, 2, \dots, n_d$  do
3:     Sample  $n$  snapshots  $\{F^{(j)}\}_{j=1}^n$  from the training data set.
4:     Sample  $n$  random vectors  $\{\xi^{(j)}\}_{j=1}^n \sim Q_z$ .
5:     Sample  $n$  uniform random numbers  $\{\epsilon^{(j)}\}_{j=1}^n \sim U[0, 1]$ .
6:     for  $j = 1, 2, \dots, n$  do
7:        $\hat{F}^{(j)} \leftarrow \epsilon^{(j)} F^{(j)} + (1 - \epsilon^{(j)}) \tilde{F}(\xi^{(j)})$ 
8:        $L^{(j)} \leftarrow D_\rho(\tilde{F}(\xi^{(j)})) - D_\rho(F^{(j)}) + \lambda(\|\nabla_{\tilde{F}^{(j)}} D_\rho(\hat{F}^{(j)})\|_2 - 1)^2$ 
9:     end for
10:     $\rho \leftarrow \text{Adam}(\nabla_\rho \frac{1}{n} \sum_{j=1}^n L^{(j)}, \rho, \alpha, \beta_1, \beta_2)$ 
11:  end for
12:  Sample  $n$  random vectors  $\{\xi^{(j)}\}_{j=1}^n \sim Q_z$ .
13:   $\theta \leftarrow \text{Adam}(\nabla_\theta \frac{1}{n} \sum_{j=1}^n -D_\rho(\tilde{F}(\xi^{(j)})), \theta, \alpha, \beta_1, \beta_2)$ 
14: end for

```

2.4.2 Solving Stochastic Differential Equations with PI-GANs

Consider Equation 2.1, as illustrated in Figure 2.3, solving SDEs with PI-GANs consists of the following three steps.

First, we use two independent feed forward DNNs, namely $\tilde{k}_{\theta_k}(\mathbf{x}; \boldsymbol{\xi})$ and $\tilde{u}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi})$ parameterized by θ_k and θ_u , to represent the stochastic processes $k(\mathbf{x}; \omega)$ and $u(\mathbf{x}; \omega)$ in the aforementioned way.

Second, inspired by the physics-informed neural networks for deterministic PDEs [106], we encode the equation into the neural networks system by applying the operator $\mathcal{N}_x$ and B_x on the feed forward DNNs $\tilde{k}_{\theta_k}(\mathbf{x}; \boldsymbol{\xi})$ and $\tilde{u}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi})$ to generate “induced” neural networks, which are formulated as

$$\tilde{f}_{\theta_u, \theta_k}(\mathbf{x}; \boldsymbol{\xi}) = \mathcal{N}_x[\tilde{u}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi}); \tilde{k}_{\theta_k}(\mathbf{x}; \boldsymbol{\xi})] \quad (2.13)$$

and

$$\tilde{b}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi}) = B_x[\tilde{u}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi})]. \quad (2.14)$$

Differentiation in $\mathcal{N}_x$ and B_x are performed by automatic differentiation [6]. We then use $\tilde{f}_{\theta_u, \theta_k}(\mathbf{x}; \boldsymbol{\xi})$ and $\tilde{b}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi})$ as the generators of $f(\mathbf{x}; \omega)$ and $b(\mathbf{x}; \omega)$, respectively. Note that both $\tilde{f}_{\theta_u, \theta_k}(\mathbf{x}; \boldsymbol{\xi})$ and $\tilde{b}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi})$ are induced from $\tilde{u}_{\theta_u}(\mathbf{x}; \boldsymbol{\xi})$ and $\tilde{k}_{\theta_k}(\mathbf{x}; \boldsymbol{\xi})$, thus the parameters θ_k and θ_u are directly inherited from $\tilde{u}$ and $\tilde{k}$.

In the third step, we incorporate our training data to conduct adversarial training. The training data are collected as a group of snapshots in Equation 2.3.

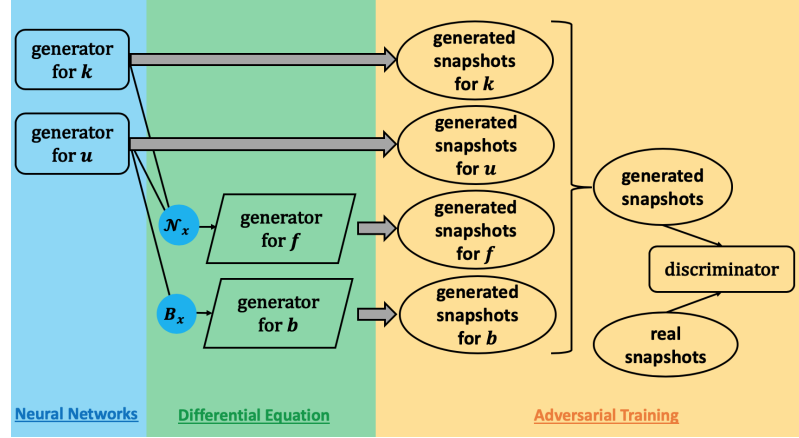


Figure 2.3: Schematic of solving SDEs with PI-GANs. The round corner rectangles in the blue and yellow boxes represent feed forward neural networks. The parallelograms in the green box represent the neural networks induced by operators ($\mathcal{N}_x$ and B_x). The ellipses represent snapshots from sensors, and the gray arrows represent sampling procedures. The bracket in the yellow box represents concatenation.

The corresponding generated “fake” snapshots are

$$\begin{aligned}
 \{G(\xi^{(j)})\}_j &= \{(\tilde{K}(\xi^{(j)}), \tilde{U}(\xi^{(j)}), \tilde{F}(\xi^{(j)}), \tilde{B}(\xi^{(j)}))\}_j, \\
 \tilde{K}(\xi^{(j)}) &= (\tilde{k}_{\theta_k}(x_i^k; \xi^{(j)}))_{i=1}^{n_k}, \\
 \tilde{U}(\xi^{(j)}) &= (\tilde{u}_{\theta_u}(x_i^u; \xi^{(j)}))_{i=1}^{n_u}, \\
 \tilde{F}(\xi^{(j)}) &= (\tilde{f}_{\theta_k, \theta_u}(x_i^f; \xi^{(j)}))_{i=1}^{n_f}, \\
 \tilde{B}(\xi^{(j)}) &= (\tilde{b}_{\theta_u}(x_i^b; \xi^{(j)}))_{i=1}^{n_b},
 \end{aligned} \tag{2.15}$$

where $\{\xi^{(j)}\}_j$ are instances of ξ with j as the index. We could then feed the generated snapshots and real snapshots into discriminator, and train the generators and discriminators iteratively. With well trained generators, we can then calculate all the statistics with sample paths created by the generators.

If the snapshots are collected in M groups ($M > 1$), we will also need to generate

M groups of “fake” snapshots:

$$\begin{aligned}
\{\{G_t(\boldsymbol{\xi}^{(t,j)})\}_j\}_{t=1}^M &= \{\{\tilde{K}_t(\boldsymbol{\xi}^{(t,j)}), \tilde{U}_t(\boldsymbol{\xi}^{(t,j)}), \tilde{F}_t(\boldsymbol{\xi}^{(t,j)}), \tilde{B}_t(\boldsymbol{\xi}^{(t,j)})\}\}_j\}_{t=1}^M, \\
\tilde{K}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{k}_{\theta_k}(\mathbf{x}_i^{k,t}; \boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{k,t}}, \\
\tilde{U}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{u}_{\theta_u}(\mathbf{x}_i^{u,t}; \boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{u,t}}, \\
\tilde{F}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{f}_{\theta_k, \theta_u}(\mathbf{x}_i^{f,t}; \boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{f,t}}, \\
\tilde{B}_t(\boldsymbol{\xi}^{(t,j)}) &= (\tilde{b}_{\theta_u}(\mathbf{x}_i^{b,t}; \boldsymbol{\xi}^{(t,j)}))_{i=1}^{n_{b,t}},
\end{aligned} \tag{2.16}$$

where ts are the indices for the groups, $\boldsymbol{\xi}^{(t,j)}$ is an instance of $\boldsymbol{\xi}$ for each (t, j) , $\{\mathbf{x}_i^{k,t}\}_{i=1}^{n_{k,t}}$ is the position setup of $n_{k,t}$ sensors for k in group t (similarly for other terms). We will also use multiple discriminators $\{D_{\rho_t}(\cdot)\}_t^M$, with each discriminator focusing on one group of snapshots, while the generators need to “deceive” all the discriminators simultaneously.

We give a formal and detailed description of our method in Algorithm 2. For the case where we only have one group of data, we set $M = 1$. For simplicity, here we set the weight in the generator loss function $a_t = 1$ for each t , which works well, but the method of setting $\{a_t\}_{t=1}^M$ requires further study.

Algorithm 2 PI-GANs for solving stochastic differential equations.

Require: training steps n_t , the gradient penalty coefficient λ , the number of discriminator iterations per generator iteration n_d , the batch size n , Adam hyperparameters α, β_1, β_2 , initial values θ_{k0}, θ_{u0} and $\{\rho_{t0}\}_{t=1}^M$ for the parameters θ_k, θ_u and $\{\rho_t\}_{t=1}^M$, the weights in the loss function for the generators $\{a_t\}_{t=1}^M$

- 1: **for** $s_t = 1, 2, \dots, n_t$ **do**
- 2: **for** $s_d = 1, 2, \dots, n_d$ **do**
- 3: **for** $t = 1, 2, \dots, M$ **do**
- 4: Sample n snapshots $\{T_t^{(j)}\}_{j=1}^n$ from training data group t .
- 5: Sample n random vectors $\{\xi^{(t,j)}\}_{j=1}^n \sim Q_z$.
- 6: Sample n uniform random numbers $\{\epsilon^{(t,j)}\}_{j=1}^n \sim U[0, 1]$.
- 7: **for** $j = 1, 2, \dots, n$ **do**
- 8: $\hat{G}_t^{(j)} \leftarrow \epsilon^{(j)} T_t^{(j)} + (1 - \epsilon^{(j)}) G_t(\xi^{(t,j)})$
- 9: $L_t^{(j)} \leftarrow D_{\rho_t}(G_t(\xi^{(t,j)})) - D_{\rho_t}(T_t^{(j)}) + \lambda(\|\nabla_{\hat{G}_t^{(j)}} D_{\rho_t}(\hat{G}_t^{(j)})\|_2 - 1)^2$
- 10: **end for**
- 11: $\rho_t \leftarrow \text{Adam}(\nabla_{\rho_t} \frac{1}{n} \sum_{j=1}^n L_t^{(j)}, \rho_t, \alpha, \beta_1, \beta_2)$
- 12: **end for**
- 13: **end for**
- 14: Sample n random vectors $\{\xi^{(j)}\}_{j=1}^n \sim Q_z$.
- 15: $\theta \leftarrow \text{Adam}(\nabla_{\theta} \sum_{t=1}^M a_t (\frac{1}{n} \sum_{j=1}^n -D_{\rho_t}(G_t(\xi^{(j)}))), \theta, \alpha, \beta_1, \beta_2)$, where $\theta = (\theta_k, \theta_u)$
- 16: **end for**

Note that our method does not explicitly distinguish the three types of problems described Section 2.2. Solving forward problems, inverse problems or mixed problems actually uses the same framework.

2.5 Computational Results

The following settings are commonly shared by all the test cases. We use $\tanh$ as the activation function instead of the commonly used ReLU activation function for generator neural networks, because piece-wise linear functions are not suitable for solving SDEs, where we may need to take high order derivatives. For discriminator neural networks, we also use $\tanh$ as the activation function, if not specifically mentioned. The feed forward DNNs for generators in the following numerical experiments have 4 hidden layers of width 128, if not specifically mentioned. The sizes of the input layer into the generators vary and will be specified case by case. As the default settings, the discriminators have hidden layers of the same size as that of the generators, except for the ones in Section 2.5.1 with 4 hidden layers of width 64, the one in the 120-dimensional problem in Section 2.5.2 which we will specify its size later, and the one for the additional group of snapshots on $u(x; \omega)$ in Section 2.5.3 that has 4 hidden layers of width 16. To initialize the DNNs, we use the uniform Xavier initializer for weights and zero initializer for biases. The distribution Q_z of the noise input into the generators is an independent multi-variate standard Gaussian distribution. For the hyper-parameters in loss functions and optimizers, we use the default values of $\lambda = 0.1, n_d = 5, \alpha = 0.0001, \beta_1 = 0.5, \beta_2 = 0.9$, as in the toy problem in [43]. The sensors are placed equidistantly in the domain. Our algorithms are implemented with Tensorflow. For the cases of solving PDEs, in order to generate the data as well as the references solutions, we numerically simulated the sensor measurements with Monte Carlo method using the finite difference scheme. All the references solutions are generated from 1×10^5 Monte Carlo sample paths.

2.5.1 A Pedagogical Problem: Approximating Stochastic Processes

In this section, we test the problem of approximating stochastic processes. Consider the following Gaussian processes with zero mean and squared exponential kernel:

$$f(x) \sim \mathcal{GP}\left(0, \exp\left(\frac{-(x - x')^2}{2l^2}\right)\right), \quad (2.17)$$

$$x \in D = [-1, 1],$$

where l is the correlation length.

Effect of the Number of Sensors and Snapshots

We consider three different choices of correlation length l : $l = 1, 0.5, 0.2$, and two sensor numbers for $f(x; \omega)$: 11 and 6, and fix the number of snapshots to be 1000. For $l = 0.2$ we also consider a supplementary case, where we have 1×10^4 snapshots. The training sample paths and positions of sensors are illustrated in Figure 2.4. For each case, we run the code three times with different random seeds. The batch size in all the cases is 1000. The input layer of the generators has width of 5, i.e., the input noise is a four-dimensional random vector.

To decide when to stop the training process, we calculate the $\mathbb{W}_1$ distances between the empirical distributions of generated snapshots and training snapshots $\mathbb{W}_1(\hat{P}_g^n, \hat{P}_t^n)$, where $\hat{P}_g^n$ is the empirical distribution of n generated snapshots from Monte Carlo sampling and $\hat{P}_t^n$ is the empirical distribution of n snapshots from the training set, using the python POT package [33]. In our tests, we set $n = 1000$. We plot the $\mathbb{W}_1$ distances in Figure 2.5. Note that the $\mathbb{W}_1$ distances decay and converge during the training, indicating that the generated distributions approach the real

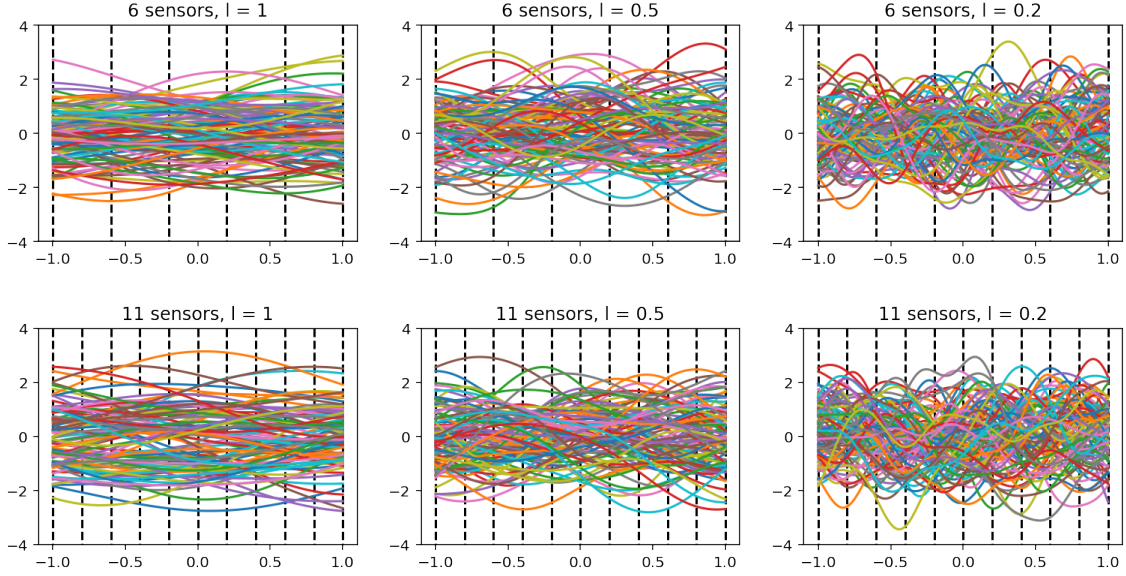


Figure 2.4: Sample paths of Gaussian processes with the sensor locations denoted by the vertical dashed lines. We use the correlation length $l = 1$ (left), 0.5 (middle), 0.2 (right), and 6 sensors (top) and 11 sensors (bottom).

distributions. We stop the training after 1×10^5 steps since $W_1(\hat{P}_g^n, \hat{P}_t^n)$ is stable. In each run, we select the 11 generators at training step in the last 10001 steps with a stride of 1×10^3 ; all together, we have 33 generators for each case. From each selected generator, we generate 1×10^4 sample paths based on the Halton quasi-Monte Carlo method, and calculate its spectra, i.e. the eigenvalues of the covariance matrices from the principal component analysis.

The results are illustrated in Figure 2.6, from which we conclude that:

1. When we fix the number of snapshots to be 1000, as the correlation length decreases, the gap between our generated processes and the reference becomes wider. This is because smaller l results in higher effective dimension, and more subtle local behavior of the stochastic processes; however, this gap could be narrowed if we increase the number of snapshots.
2. The approximations in the cases with 11 sensors are better than the approxi-

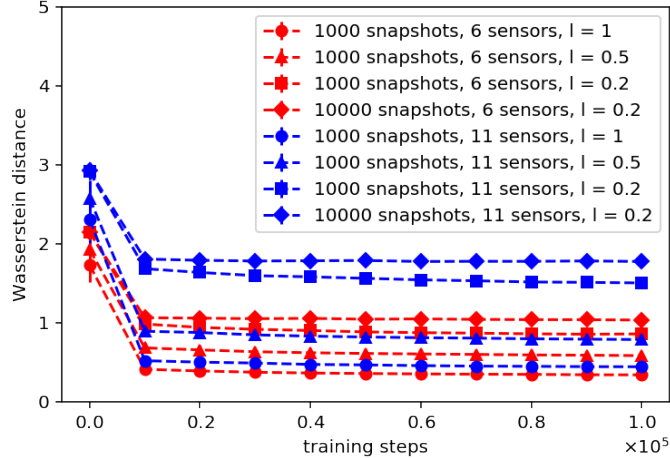


Figure 2.5: W_1 distances $W_1(\hat{P}_g^n, \hat{P}_t^n)$ versus training steps, for different correlation lengths and number of sensors. The decay of W_1 distances becomes slow after about 2×10^4 steps and hard to see after 8×10^4 steps. We set the number of sample snapshots for empirical distribution to be $n = 1000$. The means and two standard deviations are from 3 independent runs and 10 batches of n generated snapshots and training snapshots in each run.

mations in the cases with 6 sensors if we have sufficient training data. This is reasonable since we need more sensors to describe the stochastic processes with small correlation length.

3. Despite the fact that the input noise into the generator is a four-dimensional vector, the spectra of the generated processes approximate the spectra of the target processes with much higher effective dimensionality. This makes sense since the low dimensional manifold could fold and twist itself to fill the high dimensional region, as discussed earlier.

Different Kernels

To show that our method can learn various spatial correlation structures, apart from the squared exponential kernel shown above, we also test the problem of

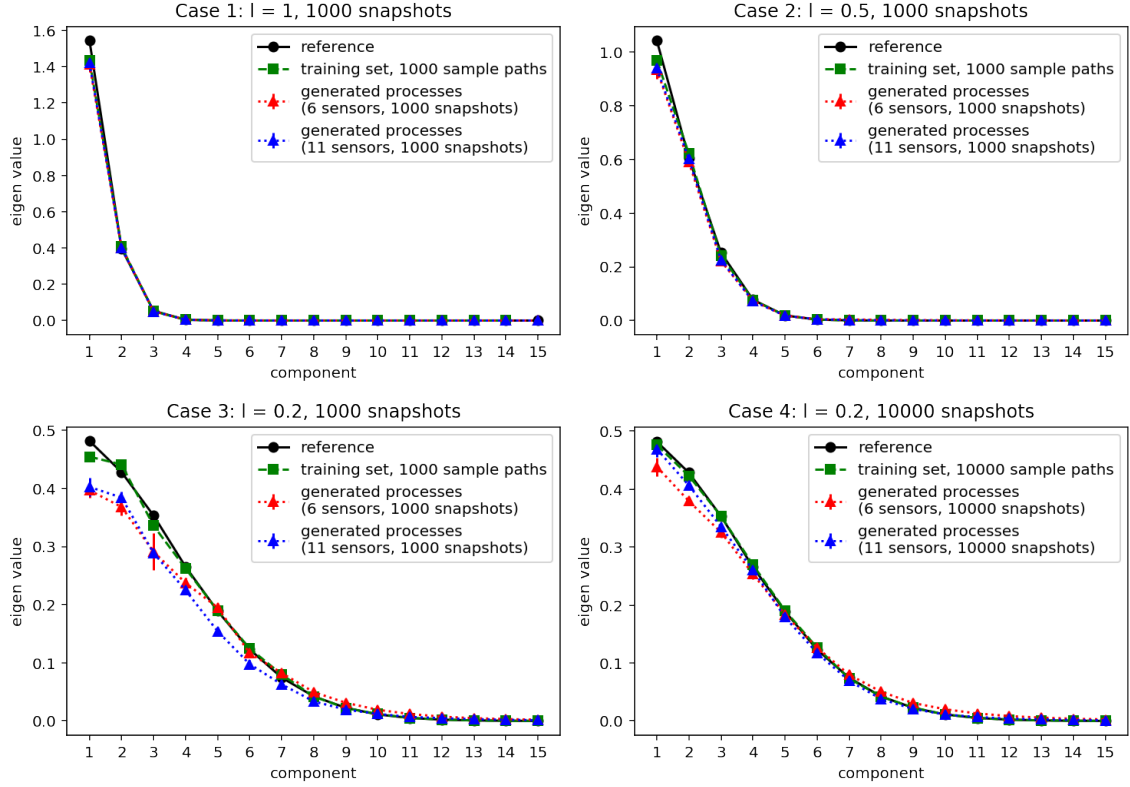


Figure 2.6: Spectra of the correlation structure for the generated processes of different correlation lengths with the squared exponential kernel. The training sets consist of the processes where the training snapshots are collected from, and the reference is calculated from 1×10^5 independent sample paths. The means and two standard deviations are calculated from the selected 33 generators for each case; the standard deviations are very small.

approximating stochastic processes with an exponential kernel:

$$f(x) \sim \mathcal{GP}\left(0, \exp\left(-\frac{|x - x'|}{l}\right)\right), \quad x, x' \in D = [-1, 1], \quad (2.18)$$

where l is the correlation length. Similar to the squared exponential kernel above, we test the 4 cases with the different correlation lengths and number of snapshots. The results are illustrated in Figure 2.7, and we can see the improvement as we increase the number of sensors and snapshots, as well as that the spectra of generated processes fit the reference values given sufficient sensors and snapshots, just like in the squared exponential kernel case.

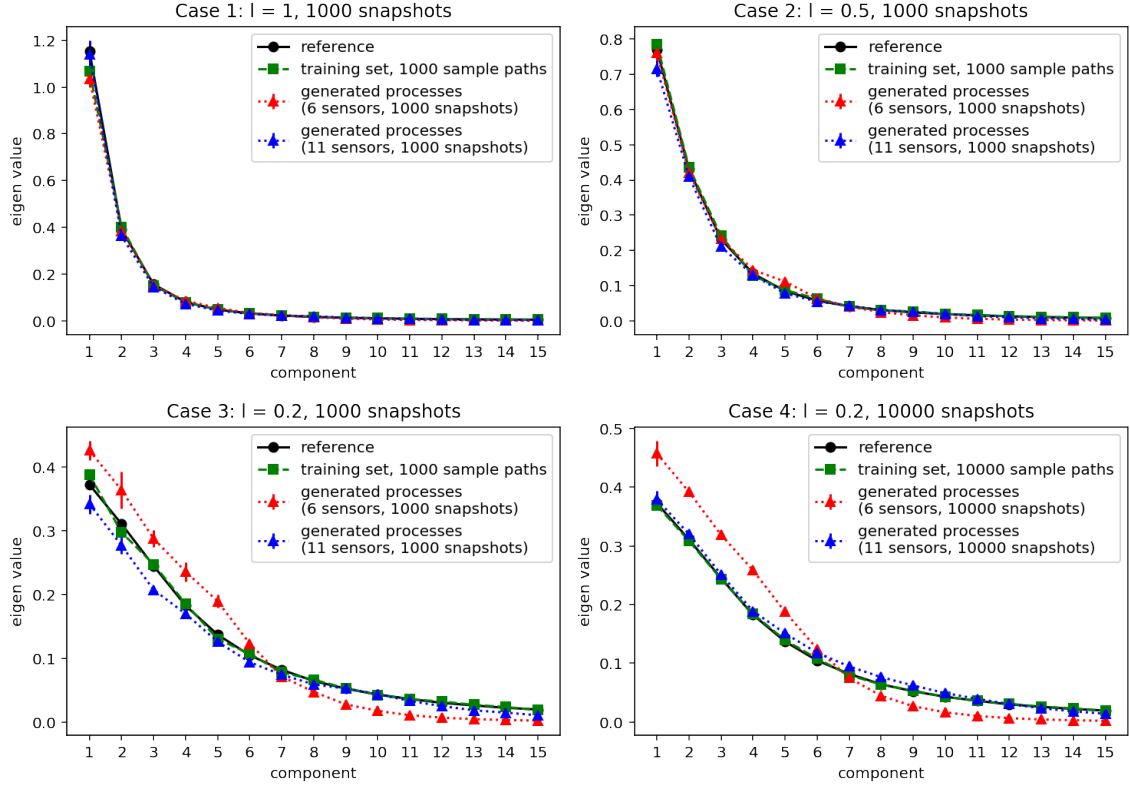


Figure 2.7: Spectra of the correlation structure for the generated processes for different correlation lengths with the exponential kernel. The training sets consist of the processes where the training snapshots are collected from, and the reference solution is calculated from 1×10^5 independent sample paths. The means and two standard deviations are calculated from the selected 33 generators for each case; the standard deviations are very small.

To further illustrate that our method can learn different spatial correlation structures, in Figure 2.8 we plot the covariance of the generated processes evaluated at randomly selected pairs of coordinates, for both types of covariance kernels. To be more specific, for each case we randomly select 1000 pairs of (x, x') from the domain, calculate the covariance of the generated processes at x and x' , and then show the scatter plots of the covariance against the distance between x and x' . We can see that for both cases, the scatter plots fit well with the analytic reference, showing that our method can reveal the different spatial correlation structures.

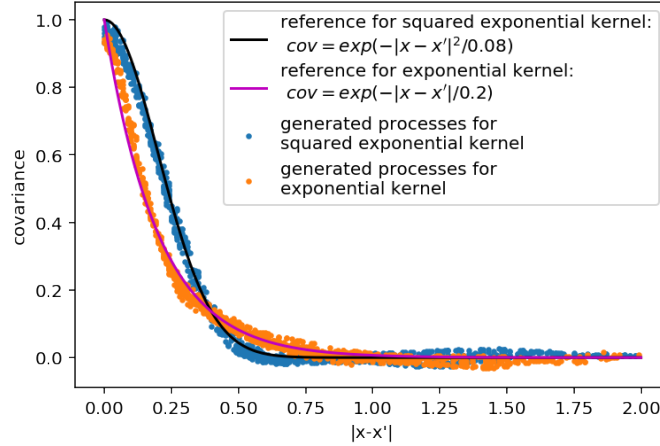


Figure 2.8: Scatter plot of covariance of the generated processes evaluated at randomly selected pairs of coordinates, for both squared exponential kernel and the exponential kernel. Each blue or orange dot represents the covariance of generated random processes with different structures, evaluated at a randomly selected pair of points (x, x') . The black and purple lines indicate the reference covariance from the exact expressions of the squared exponential kernel and the exponential kernel, respectively.

WGAN-GP Versus Vanilla GANs

We compare the performance of WGAN-GP and the vanilla GANs in approximating stochastic processes for the following two cases:

1. A Gaussian process in Equation 2.17 with $l = 0.2$.
2. A stochastic process with fixed 0 boundary condition. Specifically, we consider $f(x) = (x^2 - 1)g(x)$, where $g(x)$ is a Gaussian process:

$$g(x) \sim \mathcal{GP}\left(0, \exp\left(\frac{-(x - x')^2}{2 \times 0.2^2}\right)\right), \quad x, x' \in D = [-1, 1].$$

In both cases, we use 1×10^4 snapshots collected from 11 sensors as training data. In Figure 2.9 we show the means and standard deviations of generated processes trained by WGAN-GP and the vanilla GANs. We can see that both versions of GANs produce good approximations for case 1. However, the vanilla GANs fail in

case 2, while WGAN-GP still generates a good approximation. This agrees with the theory in [43] that vanilla GANs are not suitable for approximating distributions concentrated on low dimensional manifolds (the two fixed boundaries in this case), since the JS divergence cannot provide a usable gradient for the generators.

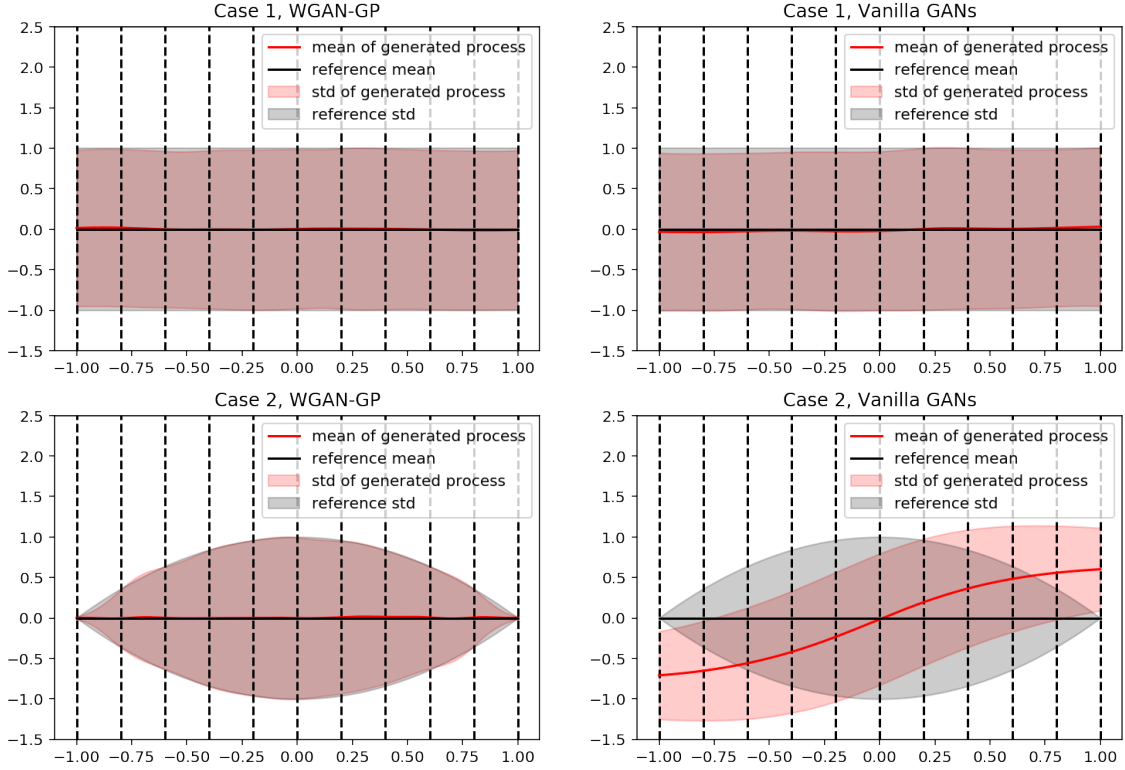


Figure 2.9: Mean and standard deviation of random processes generated by WGAN-GP and vanilla GANs: For case 1, both versions of GANs perform well, but the vanilla GANs failed in case 2, while the WGAN-GP can still provide accurate results. The vertical dashed lines represent the positions of sensors.

Overfitting Issues

It was pointed out in [43] that the discriminator can overfit the training data given sufficient capacity but too little training data. Here we report the same issue in our method. Take the case of approximating the stochastic process where the correlation length is $l = 0.2$ with 11 sensors and 10000 training snapshots as an example. We plot the negative discriminator loss $-L_d$ for the training set and the

validation set in Figure 2.10a. As training goes on, the negative discriminator loss gradually increases for the training set while still decreases for the validation set. The gap between them implies that the discriminator overfits the training data, and gives a biased estimation of the $\mathbb{W}_1$ distance between the real distribution and generated distribution, just as is reported in [43].

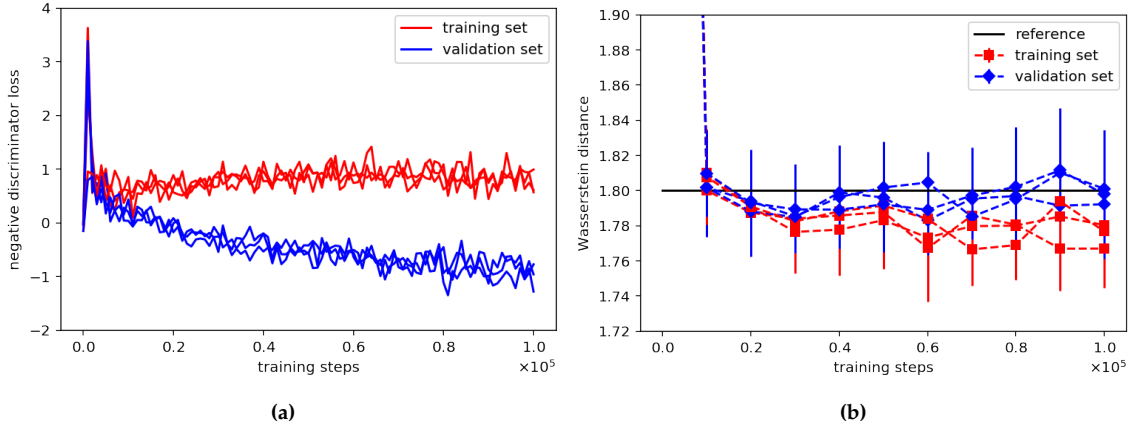


Figure 2.10: Overfitting of the discriminator *and* generator. (a) Negative discriminator loss on the training set and validation set versus the training steps in three independent runs. (b) $\mathbb{W}_1$ distances $\mathbb{W}_1(\hat{P}_g^n, \hat{P}_t^n)$ and $\mathbb{W}_1(\hat{P}_g^n, \hat{P}_v^n)$ versus the training steps in three independent runs. The means (markers) and two standard deviations (vertical lines on the marker) come from ten groups of $(\hat{P}_g^n, \hat{P}_t^n, \hat{P}_v^n)$. The black horizontal line is the empirical expectation of $\mathbb{W}_1(\hat{P}_{r1}^n, \hat{P}_{r2}^n)$ from 50 groups of $(\hat{P}_{r1}^n, \hat{P}_{r2}^n)$.

How about the overfitting of generators? In our problem, the overfitting of generators comes in two types:

Type-1 : Overfitting in the random space: The distribution of generated snapshots recovers the empirical distribution of the training snapshots instead of the true underlying distribution. In the worst case, the generated snapshots are concentrated on or near the support of training data.

Type-2 : Overfitting in the physical space: The generated stochastic processes become worse after extensive training, and tend to match the real processes only at the sensor locations and display large variations where there is no sensor.

We first report that we did not detect type-2 overfitting in our experiments. Actually, this type of overfitting would be reflected on the mean and standard deviation of generated processes, which fit the reference values pretty well in our experiments. We attribute this to the property of GANs that the target for the generator is to approximate a distribution rather than a single point on the sensors. As a result, the generator does not need to overfit a specific value on the sensors in order to decrease the loss.

As for the type-1 overfitting, we could detect it in our experiments. As depicted in Figure 2.10b, we can see this from the W_1 distances between empirical distributions of the generated snapshots and the training snapshots or the validation snapshots, i.e., $W_1(\hat{P}_g^n, \hat{P}_t^n)$ or $W_1(\hat{P}_g^n, \hat{P}_v^n)$, where $\hat{P}_g^n$, $\hat{P}_t^n$ and $\hat{P}_v^n$ are empirical distributions of generated snapshots, training snapshots and validation snapshots, n is the number of snapshots. Here, we set $n = 1000$. We can see that as the training goes on, $W_1(\hat{P}_g^n, \hat{P}_v^n)$ converges around the expectation of $W_1(\hat{P}_{r1}^n, \hat{P}_{r2}^n)$, where $\hat{P}_{r1}^n$ and $\hat{P}_{r2}^n$ are independent empirical distributions of n snapshots from real distribution P_r . However, $W_1(\hat{P}_g^n, \hat{P}_t^n)$ goes down below the reference line, indicating that the generated snapshots tend to approximate the empirical distribution of the training snapshots instead of the true underlying distribution, in other words, type-1 overfitting actually happened.

Finally, we point out that type-1 overfitting is less harmful than underfitting: in our problems, even in the worst case where the generated distribution is concentrated on or near the support of training data, we can still recover the sample paths whose snapshots are concentrated on or near the training snapshots, and based on these sample paths we can still obtain decent estimations.

2.5.2 Forward Problem

Effects of Input Noise Dimension and Number of Training Snapshots

We consider the elliptic SDE 2.2, and $k(x; \omega)$ and $f(x; \omega)$ as the following independent stochastic processes:

$$\begin{aligned} k(x) &= \exp\left[\frac{1}{5} \sin\left(\frac{3\pi}{2}(x+1)\right) + \hat{k}(x)\right] \\ \hat{k}(x) &\sim \mathcal{GP}\left(0, \frac{4}{25} \exp(-(x-x')^2)\right) \\ f(x) &\sim \mathcal{GP}\left(\frac{1}{2}, \frac{9}{400} \exp(-25(x-x')^2)\right) \end{aligned} \quad (2.19)$$

In this case, we need 13 dimensions to retain 99% energy of $f(x; \omega)$. We put 13 k -sensors, 21 f -sensors and 2 u -sensors on the boundary of physical domain $\mathcal{D}$. The positions of the sensors and some sample paths of $k(x; \omega)$, $u(x; \omega)$ and $f(x; \omega)$ are illustrated in Figure 2.11.

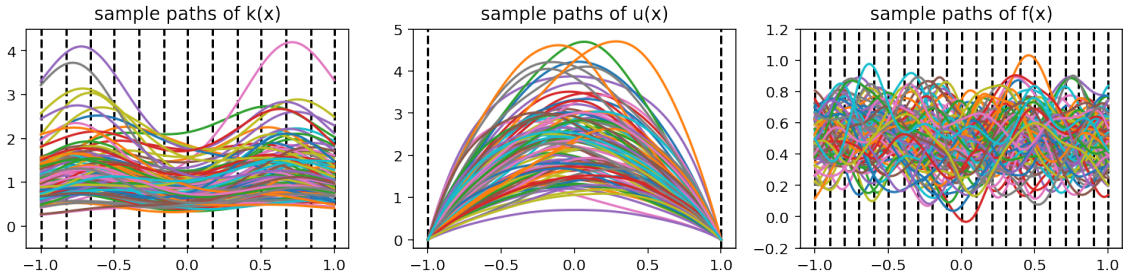


Figure 2.11: Forward problem: the sample paths and sensor locations of k , u , and f . The vertical dashed lines denote the locations of sensors.

To study the influence of input noise dimension, we fix the number of training snapshots to be 1000, and vary the input noise dimension to be 2, 4, 20 and 50. Subsequently, we fix the input noise dimension as 20 and vary the number of training snapshots to be 300, 1000 and 3000 to study the influence of the number of training snapshots. During the training process, we keep the batch size to

be the total number of training snapshots. One notable condition here is the independence of $k(x; \omega)$ and $f(x; \omega)$. To reflect this, we shuffle the alignment of snapshots from $k(x; \omega)$ and $f(x; \omega)$ in each training step. For each case, we run the code three times with different random seeds. We stop the training after 1×10^5 steps, and then select 33 generators and generate 1×10^4 sample paths from each generator in the same way as in Section 2.5.1 to calculate the following statistics.

Our main quantity of interest in this problem is the mean and standard deviation of $u(x; \omega)$, i.e., $\mu(x) = \mathbb{E}_\omega[u(x; \omega)]$ and $\sigma(x) = \sqrt{\mathbb{E}_\omega[(u(x; \omega) - \mu(x))^2]}$. We look into the relative error defined as

$$\begin{aligned} \text{relative error in mean: } & \frac{\|\hat{\mu}(x) - \mu(x)\|_2}{\|\mu(x)\|_2}, \\ \text{relative error in standard deviation: } & \frac{\|\hat{\sigma}(x) - \sigma(x)\|_2}{\|\sigma(x)\|_2}, \end{aligned} \tag{2.20}$$

where $\|\cdot\|_2$ is the L_2 norm, $\hat{\mu}(x)$ and $\hat{\sigma}(x)$ are the inferred mean and standard deviation of $u(x; \omega)$. In Figure 2.12, we show the relative error of inference from generated processes, and compare it with the relative error of the stochastic collocation method and the Monte Carlo full trajectory sample paths of $u(x; \omega)$. We can see that:

1. Our errors are of the same magnitude with the errors calculated from 1000 full trajectory sample paths of $u(x; \omega)$, showing the effectiveness of our method considering that we only have 1000 snapshots on sparsely placed sensors for $k(x; \omega)$, $f(x; \omega)$ and boundary of $u(x; \omega)$.
2. The stochastic collocation method gives a better solution, but it requires a full knowledge of $k(x; \omega)$ and $f(x; \omega)$, including the covariance kernel function, which is far beyond our accessible data.

3. When we increase the dimension of input noise, we can see that the error of mean does not change too much, but the error of the standard deviation decreases. We attribute this to the fact that although a low dimensional manifolds could twist itself to fill in the high dimensional region, higher dimensional manifolds produce better approximations by filling in the high dimensional region more efficiently.
4. With fixed input noise dimension, the error decreases as the number of training snapshots increases.

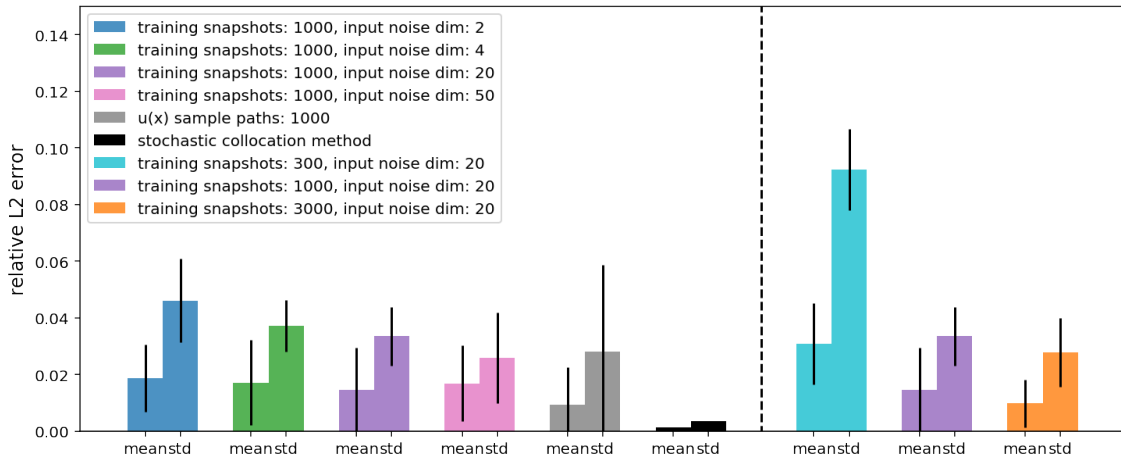


Figure 2.12: Forward problem for different input noise dimensionality (left part of the panel) and the number of training snapshots (right part of the panel): relative errors of inferred mean and standard deviation of the stochastic solution $u(x; \omega)$. The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. The grey bars and the black lines represent the expectation and two standard deviations of the relative error if we calculate the mean and standard deviation of $u(x; \omega)$ from a random draw of 1000 $u(x; \omega)$ sample paths. The black bar (before the vertical dashed line) is the relative error from stochastic collocation method with 630 collocation points. The reference solutions are calculated from 1×10^5 $u(x; \omega)$ sample paths.

We also illustrate the spectra of the generated processes for $f(x; \omega)$ in Figure 2.13 to verify that the generated processes captured the covariance structure of $f(x; \omega)$. We can see that the spectra of our generated processes fit the reference solution well. With fixed number of training snapshots, as we increase the input noise dimension, the gap between our generated processes and the reference solutions

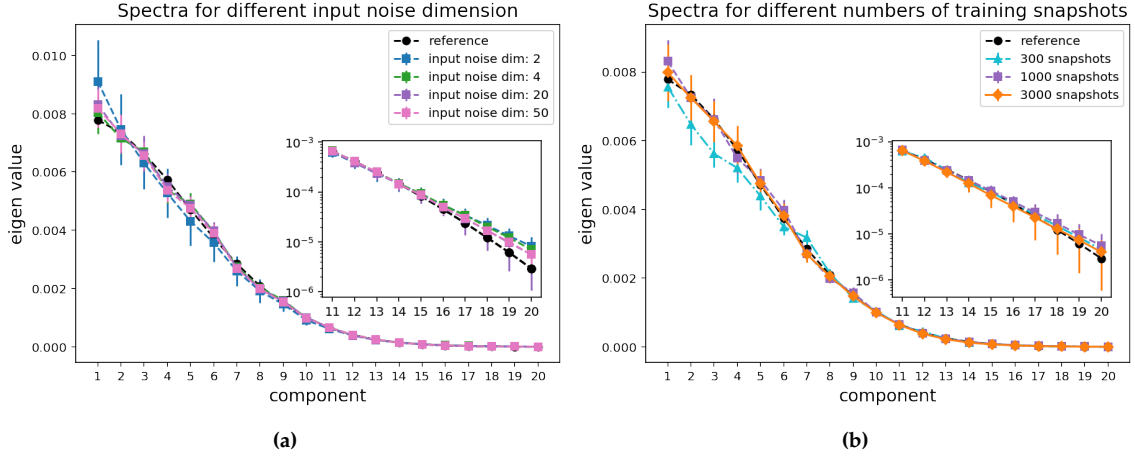


Figure 2.13: Forward problem: spectra of generated right-hand-side $f(x; \omega)$ processes. (a): Varying input noise dimension for a fixed number of snapshots at 1000. (b): Varying the number of training snapshots for a fixed dimension of input noise at 20. The inset plots are the zoom-ins of the eigenvalues of high frequency modes. The means (markers) and two standard deviations (vertical lines on the markers) are calculated from the selected 33 generators. The reference curves are calculated from another 1×10^5 independent $f(x; \omega)$ sample paths.

narrows. With fixed input noise dimension, the gap narrows as we increase the number of training snapshots. These observations on the spectra are similar with those about the error of u .

We further check the correlation between the generated processes. The correlation coefficient $C(f_1(x; \omega), f_2(x; \omega))$ between two stochastic processes $f_1(x; \omega)$ and $f_2(x; \omega)$ ($x \in D$) is defined as

$$C(f_1(x; \omega), f_2(x; \omega)) = \frac{1}{n} \sum_{i=1}^n \left| \frac{\text{Cov}(f_1(x_i; \omega), f_2(x_i; \omega))}{\sqrt{\text{Var}(f_1(x_i; \omega))\text{Var}(f_2(x_i; \omega))}} \right|, \quad (2.21)$$

where x_i is the uniform grid points in D . Here, we set $n = 201$ for $C(k, f)$, while $n = 199$ for $C(k, u)$ to exclude the boundary points for u .

From Table 2.1 we can see that our generated processes for $k(x; \omega)$ and $f(x; \omega)$ are weakly correlated, while the generated processes for $k(x; \omega)$ and $u(x; \omega)$ are strongly correlated. All the correlation coefficients are close to the reference values.

	2 dim	4 dim	20 dim	40 dim
$C(k, f)$	3.4±2.1 %	2.3±1.6 %	2.5±2.3 %	2.7±1.8 %
$C(k, u)$	74.3±1.3 %	73.6±1.0 %	73.8± 0.9 %	73.2±0.8 %

	300 snapshots	3000 snapshots	reference
$C(k, f)$	2.5±1.8 %	2.5±1.5 %	0
$C(k, u)$	74.0±0.9 %	72.4±1.5 %	72.5 %

Table 2.1: Forward problem: correlation coefficient between the generated random processes. The mean and two standard deviations are calculated from the selected 33 generators for each case. The reference for $C(k, f)$ comes from the assumption that $k(x; \omega)$ and $f(x; \omega)$ are independent, while the reference for $C(k, u)$ is calculated from Equation 2.21 with 1×10^5 independent samples of $k(x; \omega)$ and $u(x; \omega)$ paths.

High Dimensional Problems

Here, we solve the stochastic differential equation where the correlation length of $f(x; \omega)$ is relatively small. Let $k(x; \omega)$ and $f(x; \omega)$ be independent stochastic processes as follows:

$$\begin{aligned}
 k(x) &= \exp\left[\frac{1}{5} \sin\left(\frac{3\pi}{2}(x+1)\right) + \hat{k}(x)\right], \\
 \hat{k}(x) &\sim \mathcal{GP}\left(0, \frac{4}{25} \exp(-(x-x')^2)\right), \\
 f(x) &\sim \mathcal{GP}\left(\frac{1}{2}, \frac{9}{400} \exp\left(\frac{-(x-x')^2}{a^2}\right)\right).
 \end{aligned} \tag{2.22}$$

Two cases are considered, i.e., case 1: $a = 0.08$, and case 2: $a = 0.02$. Note that to retain 99% energy of $f(x; \omega)$, we need 30 dimensions for case 1 and about 120 dimensions for case 2. For case 1, we put 13 k -sensors, 41 f -sensors, and 2 u -sensors on the boundaries of our domain of interest $\mathcal{D}$, and set the input noise dimension to be 20. For case 2 we put 13 k -sensors, 161 f -sensors, and 2 u -sensors on the boundaries, and set the input noise dimension to be 100. In both cases, we

use 1×10^4 snapshots as training data while we set the batch size to be 1000. Again, we shuffle the alignment of snapshots of $k(x; \omega)$ and $f(x; \omega)$ during the training to reflect their independence.

In practice, for case 2, we increase the number of hidden layer to 8, and follow the design of ResNet [45] by adding shortcut connections in the neural networks, since it has been reported that shortcut connections help the training of deep neural networks [45]. More specifically, for two adjacent hidden layers z_l and z_{l+1} , $z_{l+1} = z_l + \sigma(\mathbf{W}_l z_l + \mathbf{b}_l)$ where $\mathbf{W}_l$ and $\mathbf{b}_l$ are weights and biases respectively, and σ is the activation function. While keeping using *tanh* activation function in the generator neural network for case 2, in the discriminator neural network we use a leaky ReLu activation function [76] with negative slope of 0.01, since it works well especially for higher resolution modeling in other application of GANs [100].

For case 1, we run the code three times with different random seeds and stop the training after 2×10^5 steps. We then select 33 generators and generate 1×10^4 sample paths from each generator, in the same way as in Section 2.5.1, to calculate the error of $u(x; \omega)$ as well as the spectra of $f(x; \omega)$. For case 2, we run the code one time (due to the high computational cost) and stop the training after 4×10^5 steps. We then select 11 generators in the last 10001 training steps, with the stride of 1000 steps, and generate 1×10^4 sample paths from each generator to calculate the error of $u(x; \omega)$ as well as the spectra of $f(x; \omega)$. We report that for a single run, compared with case 1, the wall clock computational time in case 2 is about 18 times (about $\times 9$ for each step, and $\times 2$ for total number of steps), while the number of stochastic dimensions in case 2 is about four times larger than in case 1.

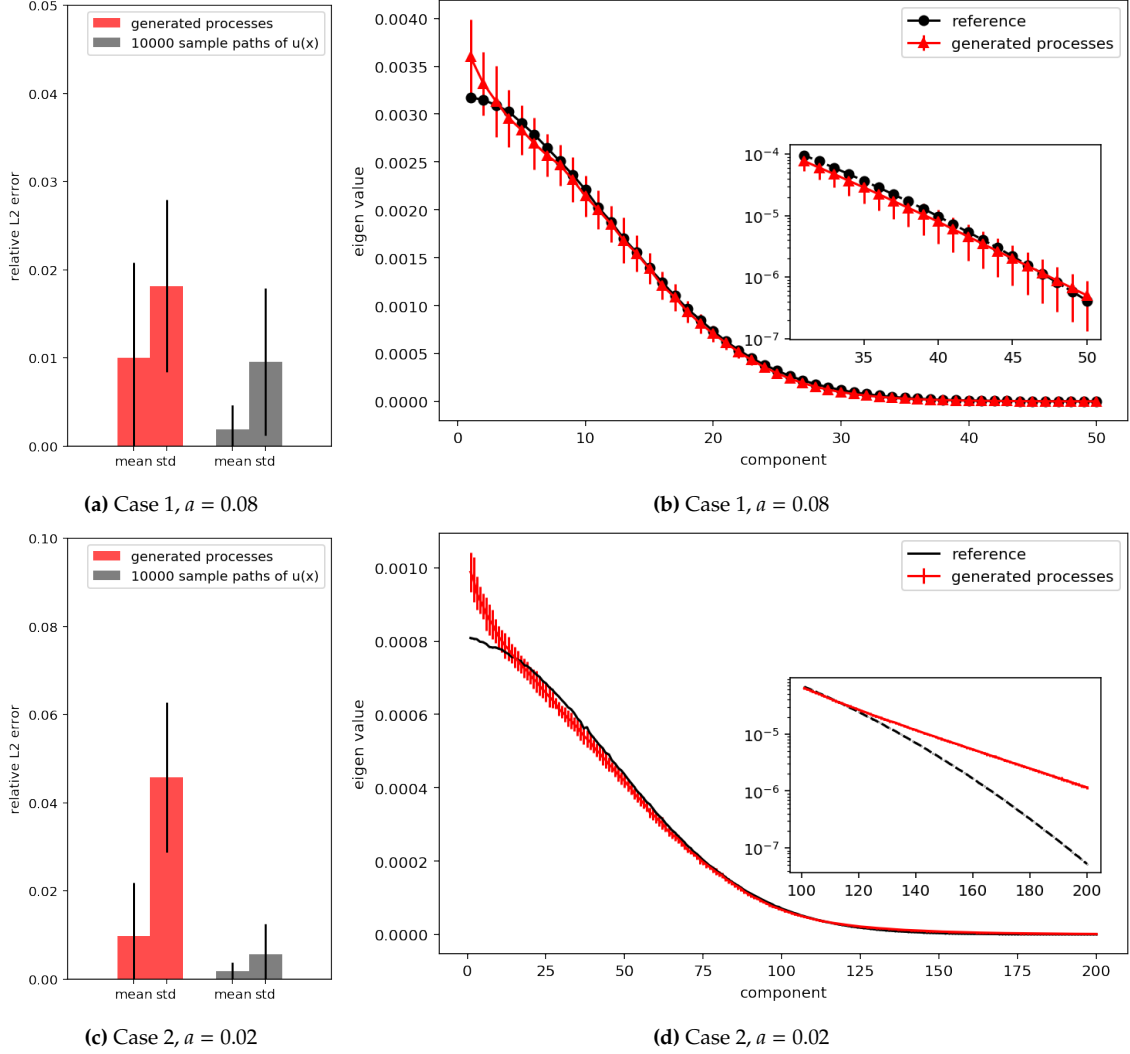


Figure 2.14: Forward problem with stochastic right-hand-side process $f(x; \omega)$ of high dimensionality. Figure (a) and (b) correspond to case 1 where the effective dimension is 30, while figure (c) and (d) correspond to case 2 where the effective dimension is about 120. Left: relative errors of inferred mean and standard deviation of $u(x; \omega)$. The red bars and their associated black lines represent the mean and two standard deviations of relative errors from the selected generators. The grey bars and their associated black lines represent the expectation and two standard deviations of relative errors calculated from 1×10^4 sample paths of $u(x; \omega)$. Right: the spectra of generated process versus the reference values. The means (red markers) and two standard deviations (red vertical lines on the markers) are calculated from the selected generators. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $u(x; \omega)$ and $f(x; \omega)$.

The results are illustrated in Figure 2.14. For case 1, we can see that the spectra fit the reference values well. The error is in the same magnitude with the error calculated from 1×10^4 sample paths of $u(x; \omega)$. In case 2, we can see that we

managed to capture the spatial correlation roughly for the first 120 components, while after the first 120 components the eigenvalues of generate processes deviated from the reference values. Compared with case 1, in case 2 the error in mean is similar, while the error in standard deviation is larger but still within 8%.

2.5.3 Inverse and Mixed Problems

In this section, we show that our method can manage a wide range of problems, from forward problems to inverse problems, and mixed problems in between. In particular, we solve the three types of problems governed by Equation 2.2, where $k(x; \omega)$ and $f(x; \omega)$ are independent processes as follows:

$$\begin{aligned} k(x) &= \exp\left[\frac{1}{5} \sin\left(\frac{3\pi}{2}(x+1)\right) + \tilde{k}(x)\right], \\ \tilde{k}(x) &\sim \mathcal{GP}\left(0, \frac{4}{25} \exp(-(x-x')^2)\right), \\ f(x) &\sim \mathcal{GP}\left(\frac{1}{2}, \frac{9}{400} \exp(-(x-x')^2)\right). \end{aligned} \tag{2.23}$$

We consider the following four cases of sensor placement:

Case 1 : 1 k -sensor, 13 u -sensors (including 2 on the boundary), 13 f -sensors.

Case 2 : 5 k -sensors, 9 u -sensors (including 2 on the boundary), 13 f -sensors.

Case 3 : 9 k -sensors, 5 u -sensors (including 2 on the boundary), 13 f -sensors.

Case 4 : 13 k -sensors, 2 u -sensors on the boundary of u , 13 f -sensors.

Note that case 1 is an inverse problem, case 4 is a forward problem, while case 2 and 3 represent mixed problems. For each case, we use 1000 snapshots for

training. Note that in cases 1-3, we *cannot* shuffle the alignment of snapshots from $k(x; \omega)$ and $f(x; \omega)$ as in Section 2.5.2 since both $k(x; \omega)$ and $f(x; \omega)$ are correlated with $u(x; \omega)$. For consistency, in this section, we do not shuffle the alignment of snapshots for any of the four cases. We set the batch size to be 1000 and the input noise dimension to be 20. For each case, we run the code three times with different random seeds. The training stops after 2×10^5 steps, and we select 33 generators and generate 1×10^4 sample paths from each generator in the same way as in Section 2.5.1 to calculate the mean and standard deviation of $k(x; \omega)$ and $u(x; \omega)$.

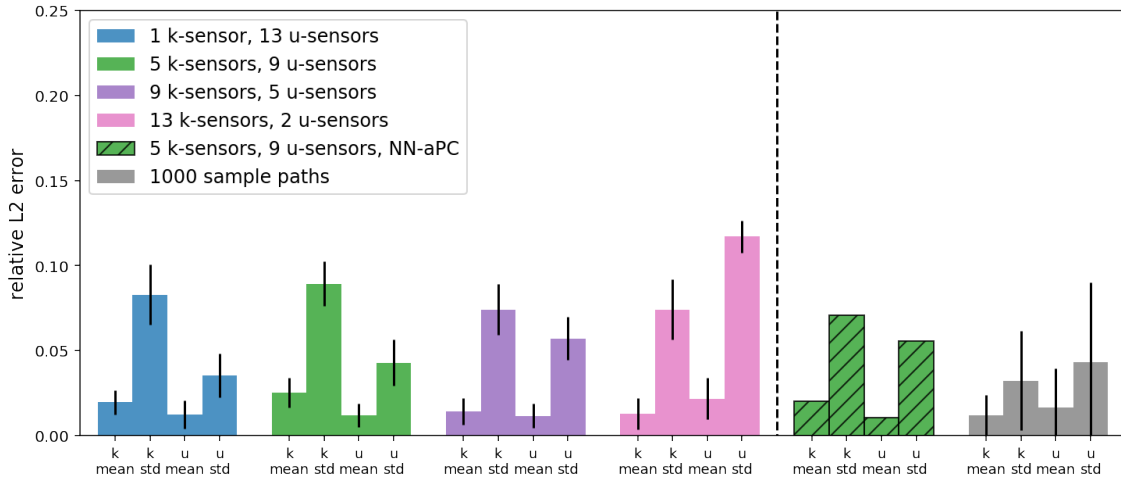


Figure 2.15: Relative errors of inferred mean and two standard deviations for both $k(x; \omega)$ and $u(x; \omega)$ in inverse problem (blue left most bars), mixed problems (green and purple bars) and forward problem (pink bars). The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. Also shown on the right part of the panel are two reference cases, the green shaded bars corresponding to NN-aPC method [148] for the case 2 on the left, and the grey bars and their associated black lines corresponding to the expectation and two standard deviations of relative error calculated from 1000 independent sample paths of $k(x; \omega)$ or $u(x; \omega)$. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $k(x; \omega)$ and $u(x; \omega)$.

In Figure 2.15 we compare the relative errors with reference solutions calculated from Monte Carlo sample paths as well as the method proposed in [148]. The relative errors of $u(x; \omega)$ and $k(x; \omega)$ are defined as in Equation 2.20. We can see that our errors are in the same order of magnitude with errors from 1000 sample paths, showing the effectiveness of our method in solving all three types of problems. Also, for the case of 5 k -sensors and 9 u -sensors, our method achieves comparable

accuracy with the method in [148].

2.5.4 Multiple Groups of Training Data

Finally, we test our method for the case where we have multiple groups of snapshots as training data. In particular, we perform our test based on the case 4 in Section 2.5.3. Apart from the sensors in that case, we put one additional u -sensor at position $x = 0$, and collect another group of 1000 snapshots at this additional sensor. Hence, we have two groups of training data:

Group 1 : 13 k -sensors, 2 u -sensors on the boundary of u , 13 f -sensors. 1000 snapshots.

Group 2 : 1 u -sensor at $x = 0$. 1000 snapshots.

Again, we emphasize that the two groups of snapshots cannot be aligned. Note that a single snapshot in group 2 is almost useless since we cannot align it with another snapshot in group 1. However, the *ensemble* of the snapshots in group 2 actually can tell the *distribution* for $u(x; \omega)$ at $x = 0$.

To utilize the data in two groups, we apply Algorithm 2 with the number of discriminators $M = 2$. In this case, we set the training batch size as 1000 while the input noise dimension as 20. We run the code three times with different random seeds. The training stops after 2×10^5 steps, and we select 33 generators and generate 1×10^4 sample paths from each generator in the same way as in Section 2.5.1 to calculate the mean and standard deviation of $k(x; \omega)$ and $u(x; \omega)$. In Figure 2.16, we plot the errors of the inferred $k(x; \omega)$ and $u(x; \omega)$ in this case as

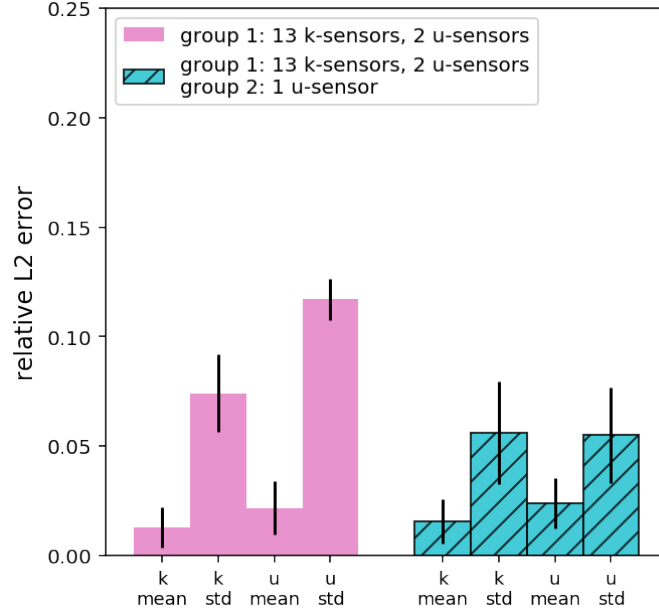


Figure 2.16: Relative errors of inferred mean and two standard deviations for both $k(x; \omega)$ and $u(x; \omega)$ using 1 group of data (pink) and 2 groups of data (light green). The colored bars and the corresponding black lines represent the mean and two standard deviations of the relative errors calculated from the selected 33 generators for each case. The reference values are calculated from 1×10^5 Monte Carlo sample paths of $k(x; \omega)$ and $u(x; \omega)$.

well as errors from case 4 in Section 2.5.3. Compared with the results obtained by only using one group of data, in this case, the error of the standard deviation of $u(x)$ decreases significantly, showing the capability of our method in *learning from the ensemble* of snapshots in group 2. To the best of our knowledge, our method is so far the only one that can manage this case.

2.6 Summary of Chapter

We proposed physics-informed generative adversarial networks (PI-GANs) as a data-driven method for solving stochastic differential equations (SDEs) based on a limited number of scattered measurements. PI-GANs are composed of a discriminator, which is represented by a simple feed forward deep neural network

(DNN) and of generators, which are a combination of feed forward DNNs and a neural network induced by the SDE. We assumed that partial data are available in terms of different realizations of the stochastic process obtained simultaneously at different locations in the domain. We trained the generators and discriminator iteratively with the loss functions employed in WGAN-GP [43] so that the joint distribution of generated processes approximates the target stochastic processes. We also proposed a more general architecture with multiple discriminators to deal with cases, where data are collected in multiple groups, i.e., data collected independently from different sets of sensors.

We first tested WGAN-GP in approximating Gaussian processes for different correlation lengths. As shown in Figure 2.6, we obtained good approximation of the generated stochastic processes to the target ones even for a mismatch between the input noise dimensionality and the effective dimensionality of the target stochastic processes. The approximations were improved by increasing the number of sensors and snapshots. We also compared WGAN-GP and vanilla GANs, and concluded that vanilla GANs are not suitable for approximating stochastic processes with deterministic boundary condition, as shown in Figure 2.9. We further studied the overfitting issue by monitoring the negative discriminator loss (Figure 2.10a) and Wasserstein distance between empirical distributions (Figure 2.10b). We found that overfitting occurs also in the generator in addition to the discriminator as previously reported.

Subsequently, we considered the solution of elliptic SDEs requiring approximations of three stochastic processes, namely the solution $u(x; \omega)$, the forcing $f(x; \omega)$, and the diffusion coefficient $k(x; \omega)$. Without changing the framework, we were able to solve a wide range of problems, from forward to inverse problems, and in between, i.e., mixed problems where we have incomplete information for

both the solution and the diffusion coefficient. As shown in Figure 2.12 and Figure 2.15, we obtained both the means and standard deviations of the stochastic solution and the diffusion coefficient in good agreement with benchmarks. In the case of the forward problem, we studied the influence of dimensionality of the input noise into the generators in Figure 2.12, and found that although small dimensionality can also work, high dimensionality for the input noise leads to better results. Moreover, we tested PI-GANs for a relatively high dimensional problem with $f(x; \omega)$ of stochastic dimension 30. In Figure 2.14 we showed that the inferred mean and standard deviation of $u(x; \omega)$ as well as the spectra of $f(x; \omega)$ match the reference values well. Finally, we applied PI-GANs consisting of two discriminators to a stochastic problem with two groups of snapshots available for training, where the second group includes snapshots from only a single sensor for $u(x; \omega)$. We could see in Figure 2.16 that the error decreases compared with the error of the case where we only have the first group of data. This demonstrates the capability of PI-GANs to utilize information from multiple groups of data and learn from an *ensemble* of snapshots, even when a single snapshot is useless.

Finally, we comment on the computational cost due to high dimensionality of stochastic problems. In Section 2.5.2 we investigate the two cases with 30 and about 120 stochastic dimensions respectively. For a single run, the wall clock computational time in case 2 is about $\times 18$ times of that in case 1 and the stochastic dimension is about $\times 4$ times, while the relative errors in both cases are of the same magnitude, suggesting that the overall computational cost increases with a low-order polynomial growth, hence PI-GANs can, in principle, tackle such problems with very high dimensional stochastic dimensions.

CHAPTER THREE

Generative Ensemble Regression: Learning Particle Dynamics from Observations of Ensembles with GANs

3.1 Introduction

Classic methods for inferring the ordinary differential equation (ODE) dynamics from data usually require observations of a point or particle governed by the ODE at different time instants. We refer this learning paradigm as “point-regression”. More specifically, as illustrated in Figure 3.1, in point-regression problems we aim to infer the governing ODE of a point and perhaps also the initial condition, given the (possibly noisy) observations of its coordinates at different time instants. Typically, we optimize the dynamics and the initial coordinate so that the inferred curve matches the data in the Euclidean distance. Let us consider, for simplicity, the one-dimensional linear regression with quadratic loss as an example. Given observations of x at multiple t , we want to optimize the parameter a in the ODE $dx_t/dt = a$ as well as the initial point $x_0 = b$ so that the mean squared L_2 distance between predictions and data points is minimized, where a and b are the slope and the intercept of the linear function. Other examples in this category include logistic regression, recurrent neural networks [47] and the neural ODE [19] for time series, etc.

For systems consisting of an ensemble of particles, point-regression may fail to apply. For example, we want to infer the governing stochastic ordinary differential equations (SODE) from observations of particle ensembles at discrete and sparse time instants, but the data of an individual particle are not sufficiently informative for dynamic inference. Another example is a system consisting of a large number of interacting particles, even close to the mean field limit, e.g., we may want to infer how the fish interact with each other from discrete snapshots of the fish school. In such scenarios, instead of learning from individual particles, we need to *learn from the particle ensembles*. Specifically, we wish to infer the governing dynamics and

perhaps also the initial condition, using observations of an ensemble of particle at discrete time instants. We call an observation at a single time instant a “snapshot”, where part or all of the particle coordinates in the ensemble are recorded. Since the particles could be indistinguishable in observations, especially in a large system, we thus consider the case where the data are not labeled with particle indices, in other words, we cannot pair data across snapshots.

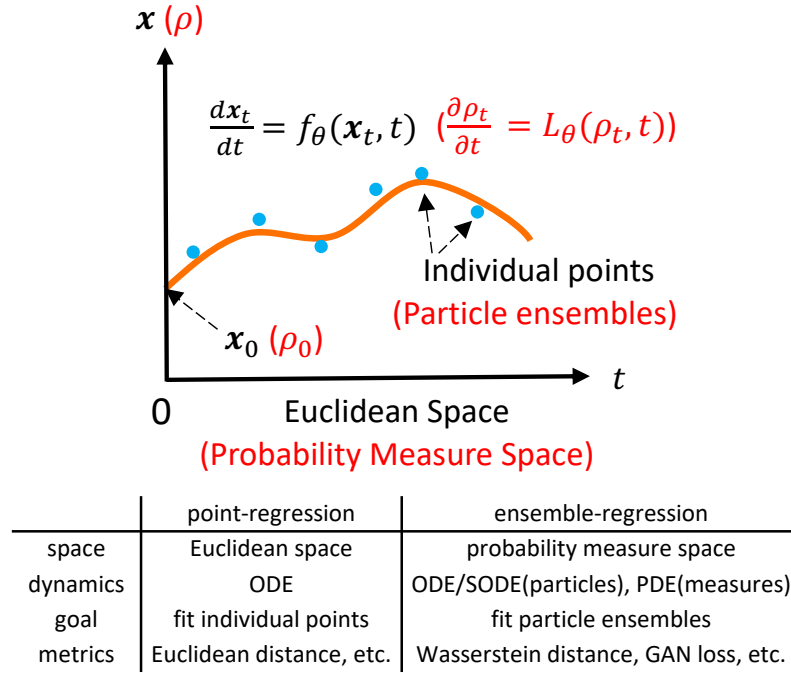


Figure 3.1: Schematic showing the two paradigms of point-regression and ensemble-regression for dynamic inference. In point-regression problems (black labels), we aim to fit the point coordinates (blue dots) from data with the inferred curve (orange curve), determined by the initial coordinate x_0 as well as the ODE $dx_t/dt = f_\theta(x_t, t)$, where f_θ is a function. As an analogue, in ensemble-regression problems (red labels), we aim to fit the distributions of the ensemble in the snapshots (blue dots) with the inferred curve (orange curve), determined by the initial distribution ρ_0 as well as the PDE $\partial\rho_t/\partial t = L_\theta(\rho_t, t)$, where L_θ is an operator.

We call this paradigm “ensemble-regression” in analogy to the “point-regression”. As illustrated in Figure 3.1, the initial condition and dynamics for particles would induce a curve $t \rightarrow \rho_t$ in the probability measure space, where ρ_t denotes the particle distributions at time t . Such ρ_t will be governed by a corresponding partial differential equation (PDE), e.g., the Fokker-Planck equation if the particles are

governed by SODEs of diffusion processes. We aim to optimize the dynamics and the initial distribution so that the inferred curve matches the distributions from the data, and the differences can be quantified with certain metrics.

Herein we propose a new method to perform ensemble-regression. We use a generative model with deep neural networks as build blocks, which will generate “fake” particle systems, to represent the inferred curve in the probability measure space, and then perform regression in the probability measure space with the inferred curve. We thus name our method as *generative ensemble-regression*. In this chapter we test the sliced Wasserstein (SW) distance [25] and the loss in generative adversarial networks (GANs) [37, 43] as two examples of metrics, the latter proved to be very effective when analyzing high dimensional data [28] in our problems.

The deep generative model is physics-informed in that our partial physical knowledge of the dynamics will be encoded into the architecture or the loss function. Such physical knowledge is sometimes essential for a correct dynamic inference, since the particle dynamic can be not unique even if the curve $t \rightarrow \rho_t$ and its governing PDE are fully given. For example, the following two particle dynamics with $\mathcal{N}(0, 1)$ as the initial distribution will lead to the same curve $\rho_t = \mathcal{N}(0, t + 1)$:

- Standard Brownian motion with no drift.
- $dx_t/dt = x_t/(2t + 2)$ with no diffusion, i.e., $x_t = x_0 \sqrt{t + 1}$.

However, if we know that the particle dynamic is in the form of $dx_t/dt = v_t(x, t)$, so that ρ_t is governed by the continuity equation $d\rho_t/dt = -\nabla \cdot (v_t \rho_t)$, and v_t is limited to the $L^2(\rho_t; \mathbb{R}^d)$ closure of $\{\nabla \varphi : \varphi \in C_c^\infty(\mathbb{R}^d)\}$, then the solution of v_t is unique, for any curve $t \rightarrow \rho_t$ absolutely continuous from $[a, b]$ to $\mathcal{P}_2(\mathbb{R}^d)$, where $\mathcal{P}_2(\mathbb{R}^d)$ is the

Wasserstein-2 space of probability measures with finite quadratic moments in $\mathbb{R}^d$ [2, 1].

In the generative model, we use discretized ODE or SODE with unknown terms parameterized as neural networks. This is referred as the “neural ODE” and “neural SDE” in the literature [19, 68, 48, 71, 127, 128], but in applications the observations are mainly a time series, i.e., observations of a single particle. The idea of employing a neural networks as velocity surrogates with physics-informed loss functions in particle systems was also used for solving mean field game/control problems [115]. There are other works using GANs to solve inverse problems, including [140, 143], where GANs were applied to learn parameters in time-independent stochastic differential equations, and [70] where GANs were applied to learn the random parameters from the (paired) observations of independent ODEs.

In this chapter, we tackle two typical types of particle systems: (1) independent particle systems governed by SODEs, and (2) interacting particle systems governed by nonlocal flocking dynamics. For inferring dynamics governed by SODE, most algorithms are based on observations of a sample path, and perform the inference by calculating or approximating the probability of observations conditioned on the system parameters, using Euler-Maruyama discretization [30, 31], Kalman filtering [119], variational Gaussian process-based approximation [3, 131], etc. There are other works inferring the stochastic dynamics with the (estimated) densities from the perspective of Fokker-Planck equations [5], but this approach is hard to scale to high dimensional problems. For flocking dynamic systems, other researchers infer the key parameters in the influence function by fitting the velocity field via Bayesian optimization [80], or fitting the density field by solving a system of transformed PDEs and optimizing the parameters [81]. But these

methods require the knowledge of the initial condition, including the distribution and velocity field.

3.2 Problem Setup

We start from a system consisting of an ensemble of particles, where the dynamics of the particles is independent of other particles. The most commonly used stochastic processes in physics and biology are diffusion processes and Lévy processes; in particular, the particle dynamics is governed by the stochastic differential equation:

$$d\mathbf{X}_t = \boldsymbol{\mu}_t dt + \boldsymbol{\sigma}_t d\mathbf{B}_t, \quad t \geq 0, \quad (3.1)$$

for diffusion processes, and

$$d\mathbf{X}_t = \boldsymbol{\mu}_t dt + \boldsymbol{\sigma}_t d\mathbf{L}_t^\alpha, \quad t \geq 0, \quad (3.2)$$

for Lévy processes, where $\mathbf{X}_t \in \mathbb{R}^d$ is the position of a particle at time t with $\mathbf{X}_0$ randomly drawn from the initial distribution ρ_0 , $\boldsymbol{\mu}_t \in \mathbb{R}^d$ is the deterministic drift, $\mathbf{B}_t$ and $\mathbf{L}_t^\alpha$ are the d -dimensional standard Brownian motion and the α -stable symmetric Lévy process, respectively, and $\boldsymbol{\sigma}_t \in \mathbb{R}^{d \times d}$ is the diffusion coefficient. In the mean field limit, the density of the particles would be governed by Fokker-Planck equations or fractional Fokker-Planck equations. For simplicity, in this work we assume that $\boldsymbol{\mu}_t$ is a function of $\mathbf{X}_t$ while $\boldsymbol{\sigma}_t$ is constant, but, in principle, our proposed method can also tackle the time-dependent case.

Apart from systems consisting of independent particles, we further consider systems where particles interact with each other and the particle distributions

exhibit more complicated behavior. As an example, we consider the Cucker-Smale particle model [22], which describes individuals in flocks with nonlocal interactions. The individual or particle motion is characterized by the following governing equations:

$$\begin{aligned} d\mathbf{X}_t^{(i)}/dt &= \boldsymbol{\mu}_t^{(i)}, \\ d\boldsymbol{\mu}_t^{(i)}/dt &= \frac{1}{N-1} \sum_{j \neq i} \phi(\|\mathbf{X}_t^{(i)} - \mathbf{X}_t^{(j)}\|)(\boldsymbol{\mu}_t^{(j)} - \boldsymbol{\mu}_t^{(i)}), \end{aligned} \quad (3.3)$$

where the superscripts denote the index of the particles, $N \gg 1$ is the number of particles in the system, and ϕ is the influence function. Here, we set

$$\phi(r) = c_{d,\alpha} |r|^{-(d+\alpha)}, \quad c_{d,\alpha} = \frac{\alpha \Gamma(\frac{d+\alpha}{2})}{2\pi^{\alpha+d/2} \Gamma(1-\alpha/2)}, \quad (3.4)$$

where Γ is the gamma function, d is the dimension of $\mathbf{X}_t^{(i)}$, and α is the factor that characterizes the decay rate of particle interactions as the distance r grows. In the mean field limit, as N grows the density of the particles is governed by the following fractional PDE:

$$\begin{aligned} \frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) &= 0, \\ \frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} &= [\mathcal{L}, \mathbf{u}](\rho), \\ [\mathcal{L}, \mathbf{u}](\rho)(\mathbf{x}) &= \text{p.v. } c_{d,\alpha} \int_{\mathbb{R}^d} \frac{\mathbf{u}(\mathbf{y}) - \mathbf{u}(\mathbf{x})}{\|\mathbf{x} - \mathbf{y}\|^{d+\alpha}} \rho(\mathbf{y}) d\mathbf{y}, \end{aligned} \quad (3.5)$$

where ρ is the density, $\mathbf{u}$ is the velocity field, and p.v. means the principle value. Note that α determines the fractional order.

We consider the scenario where the data available are the observations of the particles coordinates at different time instants $\{t_i\}_{i=1}^n$ with $0 \leq t_1 < t_2 \dots < t_n$, namely “snapshots”. In other words, the data $\mathcal{D}_i$ for time t_i will be a set of samples

drawn from ρ_{t_i} , the particle distributions at t_i . If $\{\mathcal{D}_i\}_{i=1}^n$ are observations of the same set of particles and we can distinguish these particles, we refer to these cases as the “paired” observations since we can pair the particles from different snapshots. In other cases, $\{D_i\}_{i=1}^n$ are observations of different sets of particles, or we cannot distinguish the particles. We refer to these cases as the “unpaired” observations. In this chapter we mainly focus on the unpaired cases, but we also present some work on the paired cases in Section 3.5, where we introduce how to reduce the effective dimensionality and we prove a theorem to support the introduced method. For independent particle systems, we assume that we are unaware of μ_t and σ_t , or we may only know the parametrized forms of these terms, and we aim to infer these terms directly or through a proper parametrization. For interacting particle systems, we assume we know the forms of the dynamics and the influence function, i.e., Equation 3.3 and 3.4, but need to infer the velocity field and the key parameter α .

3.3 Methodology

To perform ensemble regression, we will use a generative model with deep neural networks to represent a curve in the probability measure space. Note that the curve is determined by the initial condition ρ_0 and the governing equation, hence, the generative model consists of two parts. In the first part, we employ a feed-forward neural network G to represent ρ_0 . In particular, G takes samples from random noise $\mathcal{N}$, e.g., Gaussian noise, as input and the generated distribution $G_{\#}\mathcal{N}$ is intended to approximate ρ_0 , where $\#$ denotes the push forward operator. The second part of the generative model will generate “fake” particle trajectories with initial coordinates generated by G , and the marginal distribution $\tilde{\rho}_t$ at time

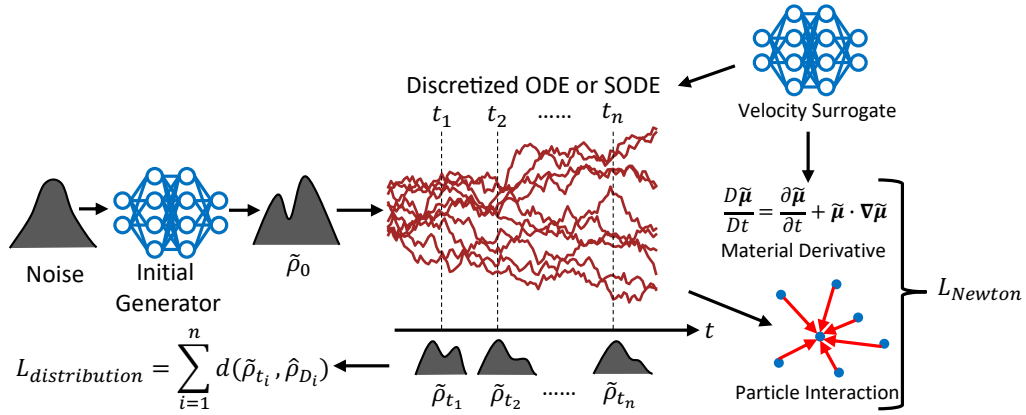


Figure 3.2: Schematic of the generative model for ensemble-regression. We first use a feed-forward neural network to map the input noise to the output $\tilde{X}_0 \in \mathbb{R}^d$, whose distribution $\tilde{\rho}_0$ is intended to approximate the initial distribution ρ_0 . Subsequently, we apply the discretized ODE or SODE with trainable parameters to generate particle trajectories $\tilde{X}_t$ for $t > 0$ with $\tilde{X}_0$ as the initial condition (brown curves). Differences between the distributions of $\tilde{X}_t$ and the snapshots from data are quantified as (a part of) our loss function. For interacting particle systems, a neural network is employed as the velocity surrogate to generate the particle trajectories. Another loss function term, namely the Newton loss L_{Newton} , is defined to quantify the consistency between the particle accelerations derived from the material derivatives, and the forces calculated from the particle interactions. With L_{Newton} , we enforce the inferred velocity to be consistent with our partial knowledge of the dynamics.

$t \geq 0$ will be used to represent ρ_t in the curve.

Our knowledge of the physics will be incorporated into the generative model in two ways. For non-interacting particle systems, our knowledge including the form of the drift and diffusion as well as the type of stochastic processes will be directly embedded into the architecture of the generative model in the second part. For interacting particle systems, while we have no direct knowledge of the velocity field, we will enforce the inferred velocity to be consistent with our knowledge of the dynamics with a physics-based soft penalty. In the following, we introduce the details of the learning algorithm for both types of systems separately. A schematic overview of the method is shown in Figure 3.2.

3.3.1 Non-Interacting Particle Systems

For non-interacting particle systems, generating particle trajectories is relatively straightforward by directly applying the discretization of governing SODE or ODE. For example, if the particle trajectories are diffusion processes or Lévy processes, we can use the following forward Euler scheme:

$$\begin{aligned}\tilde{X}_0 &= G(z), \quad z \sim \mathcal{N} \\ \tilde{X}_{(i+1)\Delta t} &= \tilde{X}_{i\Delta t} + \mu_t \Delta t + \sigma_t \sqrt{\Delta t} \xi_i, \quad i \geq 0, \\ \text{or } \tilde{X}_{(i+1)\Delta t} &= \tilde{X}_{i\Delta t} + \mu_t \Delta t + \sigma_t \Delta t^{1/\alpha} \zeta_{\alpha,i}, \quad i \geq 0,\end{aligned}\tag{3.6}$$

where Δt is the time step, ξ_i and $\zeta_{\alpha,i}$ are i.i.d. standard Gaussian random variables and α -stable random variables, respectively. We could represent μ_t and σ_t with neural networks if they are unknown, or represent the unknown parameters with trainable variables if we know their parameterized form.

Our target is to tune the trainable variables in the generative model, including the parameters in G and those for parameterizing μ_t and σ_t , so that the generated marginal distribution $\tilde{\rho}_{t_i}$ fits the data $\mathcal{D}_i$ for each i . We thus need to define a distance function $d(\cdot, \cdot)$ to measure the difference between the two input distributions, which can be estimated from samples drawn from the two distributions. Consequently the loss function in non-interacting particle systems is defined as:

$$L_{\text{distribution}} = \sum_{i=1}^n d(\tilde{\rho}_{t_i}, \hat{\rho}_{\mathcal{D}_i}),\tag{3.7}$$

where $\hat{\rho}_{\mathcal{D}_i}$ is the empirical distribution induced from the sample set $\mathcal{D}_i$. We will refer to it as the distribution loss.

There could be many ways to define $\mathbf{d}$, including Wasserstein distances, maximum mean discrepancy, etc. In this chapter we use two approaches to define $\mathbf{d}$.

First we choose the squared sliced Wasserstein-2 (SW) distance [117, 25] as the function $\mathbf{d}$:

$$\mathbf{d}(\mu, \nu) = SW_2^2(\mu, \nu) := \int_{\mathbb{S}^{d-1}} W_2^2(\pi_{e\#}\mu, \pi_{e\#}\nu) d\mathcal{H}^{d-1}(e), \quad (3.8)$$

where W_2 is the Wasserstein-2 distance, and $\pi_{e\#}\mu$ is the one dimensional distribution induced by projecting μ onto the direction e , defined by

$$(\pi_{e\#}\mu)(A) = \mu(\{x \in \mathbb{R}^d : e \cdot x \in A\}), \forall A \in \mathcal{B}(\mathbb{R}), \quad (3.9)$$

similarly for $\pi_{e\#}\nu$. $\mathcal{H}^{d-1}$ is the uniform Hausdorff measure on the sphere $\mathbb{S}^{d-1}$. In short, the squared sliced Wasserstein-2 distance is the expectation of the squared Wasserstein-2 distance between the two input measures projected onto uniformly random directions. The sliced Wasserstein-2 distance is exactly the Wasserstein-2 distance for one-dimensional distributions, but is easier to calculate for higher dimensional distributions. We present the details of the estimation in Section 3.6.1.

We also use GANs to obtain $\mathbf{d}$. The generative model we introduced above can generate “fake” samples $\tilde{\mathbf{X}}_{t_i}$ at $\{t_i\}_{i=1}^n$, and for each i , we use a discriminator D_i to discriminate generated samples $\tilde{\mathbf{X}}_{t_i}$ and real samples $\mathbf{X}_{t_i}$ from $\mathcal{D}_i$. The adversarial loss given by D_i can act as a metric of the difference between $\tilde{\rho}_{t_i}$ and $\hat{\rho}_{\mathcal{D}_i}$. In particular, we use WGAN-GP [43] as our version of GANs in our chapter, with

$$\mathbf{d}(\tilde{\rho}_{t_i}, \hat{\rho}_{\mathcal{D}_i}) = -\mathbb{E}_{\tilde{\mathbf{X}}_{t_i} \sim \tilde{\rho}_{t_i}} [D_i(\tilde{\mathbf{X}}_{t_i})] + \mathbb{E}_{\mathbf{X}_{t_i} \sim \hat{\rho}_{\mathcal{D}_i}} [D_i(\mathbf{X}_{t_i})], \quad (3.10)$$

and the loss function for each discriminator D_i is defined as

$$L_{D_i} = \mathbb{E}_{\tilde{\mathbf{X}}_{t_i} \sim \tilde{\rho}_{t_i}} [D_i(\tilde{\mathbf{X}}_{t_i})] - \mathbb{E}_{\mathbf{X}_{t_i} \sim \hat{\rho}_{\mathcal{D}_i}} [D_i(\mathbf{X}_{t_i})] + \lambda \mathbb{E}_{\hat{\mathbf{x}}_i \sim \rho_{\hat{\mathbf{x}}_i}} [(\|\nabla_{\hat{\mathbf{x}}_i} D_i(\hat{\mathbf{x}}_i)\|_2 - 1)^2], \text{ for } i = 1, 2, \dots, n, \quad (3.11)$$

where $\rho_{\hat{\mathbf{x}}_i}$ is the distribution generated by uniform sampling on straight lines between pairs of points sampled from $\tilde{\rho}_{t_i}$ and $\hat{\rho}_{\mathcal{D}_i}$, and $\lambda = 0.1$ is the gradient penalty coefficient. Here, $\mathbf{d}(\tilde{\rho}_{t_i}, \hat{\rho}_{\mathcal{D}_i})$ can be mathematically interpreted as the Wasserstein-1 distance between the two input distributions. WGAN-GP is computationally more expensive than the sliced Wasserstein distance since we need to train the generative model and the discriminators iteratively, but it is more scalable to high dimensional problems, for which we made a comparison in Section 3.6.1.

3.3.2 Interacting Particle Systems

The straightforward discretization of the governing Equations 3.3 cannot be directly applied to generate “fake” particle trajectories in interacting particle systems, especially for those with strong nonlocal interactions, since the computational cost for one time step would be $O(N^2)$, where N is the number of particles required so that the system is close to the mean field limit, which makes the learning almost intractable. Instead, we propose to first employ a neural network $\tilde{\mu}$ as a surrogate model of the velocity field μ_t in the spatial-temporal domain to generate trajectories, and then apply an additional penalty in the loss function to enforce the velocity field $\tilde{\mu}$ to be consistent with the Equations 3.3. In this chapter

we use the forward Euler scheme to generate trajectories:

$$\begin{aligned}\tilde{\mathbf{X}}_0 &= G(\mathbf{z}), \quad \mathbf{z} \sim \mathcal{N} \\ \tilde{\mathbf{X}}_{(i+1)\Delta t} &= \tilde{\mathbf{X}}_{i\Delta t} + \tilde{\boldsymbol{\mu}}(\tilde{\mathbf{X}}_{i\Delta t}, i\Delta t)\Delta t, \quad i \geq 0,\end{aligned}\tag{3.12}$$

where Δt is the time step. By employing the surrogate model for velocity, the computational cost for generating the particle trajectories is linear, instead of quadratic, with the number of particles.

To enforce $\tilde{\boldsymbol{\mu}}$ to be consistent with our knowledge of the dynamics of Equations 3.3, we first randomly generate K particle trajectories with Equation 3.12, denoted as $\{\{\tilde{\mathbf{X}}_{i\Delta t}^{(k)}\}_{i=0}^I\}_{k=1}^K$, and then calculate the “forces” applied on these k particles by another M randomly generated particles $\{\{\tilde{\mathbf{Y}}_{i\Delta t}^{(m)}\}_{i=0}^I\}_{m=1}^M$, using the formula of interactions in Equations 3.3, where we replace the unknown parameter α with a trainable variable. Meanwhile, from the velocity field $\tilde{\boldsymbol{\mu}}$ we can directly calculate the accelerations for these k particles using a material derivative. The forces and accelerations should be consistent with each other. We can thus define an L_2 loss:

$$\begin{aligned}L_{\text{Newton}} &= \frac{1}{I+1} \sum_{i=0}^I \frac{1}{K} \sum_{k=1}^K (\mathbf{F}_i^{(k)} - \mathbf{a}_i^{(k)})^2, \\ \mathbf{F}_i^{(k)} &= \frac{1}{M} \sum_{m=1}^M \phi(\|\tilde{\mathbf{X}}_{i\Delta t}^{(k)} - \tilde{\mathbf{Y}}_{i\Delta t}^{(m)}\|)(\tilde{\boldsymbol{\mu}}(\tilde{\mathbf{Y}}_{i\Delta t}^{(m)}, i\Delta t) - \tilde{\boldsymbol{\mu}}(\tilde{\mathbf{X}}_{i\Delta t}^{(k)}, i\Delta t)), \\ \mathbf{a}_i^{(k)} &= \frac{D\tilde{\boldsymbol{\mu}}(\tilde{\mathbf{X}}_{i\Delta t}^{(k)}, i\Delta t)}{Dt} = \frac{\partial \tilde{\boldsymbol{\mu}}(\tilde{\mathbf{X}}_{i\Delta t}^{(k)}, i\Delta t)}{\partial t} + \tilde{\boldsymbol{\mu}}(\tilde{\mathbf{X}}_{i\Delta t}^{(k)}, i\Delta t) \cdot \nabla \tilde{\boldsymbol{\mu}}(\tilde{\mathbf{X}}_{i\Delta t}^{(k)}, i\Delta t),\end{aligned}\tag{3.13}$$

and we name it as the Newton loss. The time span $[0, I\Delta t]$ should cover the time for the latest snapshot t_n , and in this chapter we simply set I as the ceiling of $t_n/\Delta t$. To reduce computational cost, the average over $I+1$ time steps in L_{Newton} can also be approximated by mini-batch, i.e., taking average over B random time steps in each training step.

For one time step, the computational cost is $O(KM)$ for calculating the force terms and $O(K)$ for the acceleration terms. Note that M should be larger than or equal to N , but K can be much less than N , thus the total computational cost for L_{Newton} can be much less than $O(N^2)$, and this makes the learning tractable.

In the end, the loss for interacting particle systems will be a combination of the Newton loss and the distribution loss:

$$L = \eta L_{\text{Newton}} + \frac{1}{n} \sum_{i=1}^n d(\tilde{\rho}_{t_i}, \hat{\rho}_{\mathcal{D}_i}), \quad (3.14)$$

where the weight η is set as 1 in this chapter.

3.3.3 Modifications to the Distributions

In order to provide flexibility and adapt the method to different problems, we can also modify the generated distribution and real data before feeding them to the distance function d and calculate the distribution loss. We present some examples here.

In some cases ρ_t may have heavy tails, e.g., when the particle trajectories correspond to a Lévy process. The heavy tails could spoil the training, since the rare outlier samples could dominate the loss function. We can choose a suitable bounded map $h : \mathbb{R}^d \rightarrow \mathbb{R}^d$ to preprocess both the generated samples and the real data so that the heavy tails are removed. If the observations of the particle coordinates are noisy, we can also perturb the generated samples to add artificial noise. By scaling the artificial noise with trainable variables, the size of the noise in observations can also be learned. If we can only make observations of particles

in a specific domain, i.e., the observations are truncated, we will also filter the generated samples using a corresponding mask so that the effective domain for the generated samples and observations are the same.

3.4 Computational Results

All the neural networks in this section are feed-forward neural networks with three hidden layers, each of width 128, except the discriminator for high dimensional problems which is a ResNet with 5 hidden layers, each of width 256, and shortcut connections skipping one hidden layer. We use the leaky ReLu [76] activation function with $\alpha = 0.2$ for the discriminator neural networks in WGAN-GP, while using the hyperbolic tangent activation function for other neural networks. The neural network weights are initialized with the uniform Xavier initializer, and the biases are initialized as 0. The variables for parameterizing the diffusion and noise size are initialized as 0 (before been activated by the softplus function). The drift parameters are initialized as 0 in 1D problems, -0.5 in high dimensional problems, and randomly initialized with standard Gaussian distribution in 2D problems. The batch size for the distribution loss is 1000 for non-interacting particle systems, except in the 2D problem with truncated observations, where we generate 10000 samples to compensate the loss of samples due to filtering. We use the Adam optimizer [54] with $lr = 0.0001, \beta_1 = 0.9, \beta_2 = 0.999$ for the cases using the sliced Wasserstein distance, while $lr = 0.0001, \beta_1 = 0.5, \beta_2 = 0.9$ for the cases with WGAN-GP. The time step in the generative model is set as $\Delta t = 0.01$.

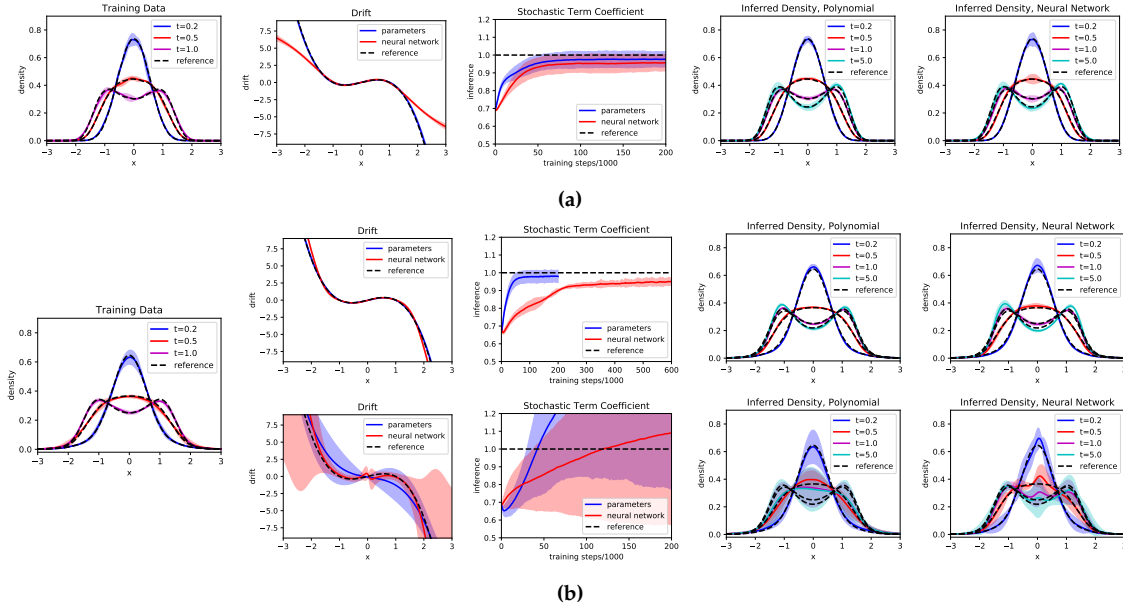


Figure 3.3: Results for 1D problems with (a) Brownian Motion and (b) Lévy process as the stochastic term. The two rows of (b) show the results with (top row) and without (bottom row) bounded map pre-processing, respectively. The first column visualizes densities estimated from training data. The second column shows the inferred drift functions in the end of training. The third column shows the inferred coefficient for the stochastic term during training. The fourth and fifth columns show the inferred densities in the two cases of using a cubic polynomial and a neural network to parameterize the drift, respectively. The solid lines and shaded areas refer to the mean and two standard deviations of three runs with different data.

3.4.1 1D Problems: Brownian Motion and Lévy Process

In this section, we test our method on the 1D problems using the sliced Wasserstein distance as d . We first consider the SODE with Brownian motion and then with the α -stable symmetric Lévy process as the stochastic term:

$$dX_t = (X_t - X_t^3)dt + dB_t, \quad (3.15)$$

$$\text{or } dX_t = (X_t - X_t^3)dt + dL_t^\alpha, \quad t \geq 0,$$

where $\alpha = 1.5$, with $\rho_0 = \mathcal{N}(0, 0.04)$.

Note that we have no knowledge of ρ_0 in learning. Also, we assume we know that the stochastic term has a constant coefficient but we need to infer it; here

the ground truth is 1.0. To represent the unknown coefficient, we use a trainable variable rectified by a softplus function $\text{softplus}(x) = \ln(1 + e^x)$, which ensures positivity. As for the drift $\mu(x) = x - x^3$, we consider the following two cases for both SODE problems. In case 1, we know that the drift $\mu(x)$ is a cubic polynomial of x . In this case, we use a cubic polynomial $a_0 + a_1x + a_2x^2 + a_3x^3$ to parameterize $\mu(x)$, and want to infer the four coefficients a_0, a_1, a_2 and a_3 . In case 2, we only know that the drift $\mu(x)$ is a function of x and hence we use a neural network to parameterize $\mu(x)$.

For the SODE problem with Brownian motion, we first prepare a pool of 10^5 sample paths, then independently draw 10,000 samples at $t = 0.2, 0.5, 1.0$ from the pool as our training data. The results for both cases of drift parameterization are illustrated in Figure 3.3a. In the figure, all the densities are estimated using Gaussian kernel density estimation with Scott's rule, and the inferred densities and the reference densities come from 10^6 samples produced by the generative model or simulation.

Both cases of drift parameterization provide a good inference of the diffusion coefficient, with an error less than 7% after 2×10^5 training steps, in all the runs. When using the cubic polynomial parameterization, the inferred drift fits well with the ground truth, with the relative L_2 error about 3% in $[-3, 3]$ averaged over three runs. The inferred drift using the neural network only fits the ground truth in the region between -1.5 and 1.5. This is reasonable since the particles mainly concentrate in this region, and we can hardly learn the drift outside this region, where the training data are sparse. However, we note that such an inference of drift is sufficiently good for an accurate time extrapolation of the particle distribution at $t = 5$.

One interesting observation is that in case 1 the inferred densities are more accurate than those estimated from the training data. This is because our knowledge of the governing SODE bridges the limited samples in three snapshots, and as we infer the density at, e.g., $t = 0.2$, we are not only utilizing the data at $t = 0.2$ but also implicitly leveraging the data at $t = 0.5$ and 1.0 . We can make an analogy in the context of linear regression: multiple noisy data are helpful for inferring the hidden ground truth, since they are bridged by the regressed linear function. We present a more detailed discussion on this topic in Section 3.6.2.

For the SODE problem with Lévy process, we prepare 1.5×10^5 sample paths, then independently draw 10,000 samples within the region $[-1000, 1000]$ at $t = 0.2, 0.5, 1.0$ from the pool as our training data. To prevent instability during the training, we apply double precision and clip the generated α -stable random variable in Equation 3.2 between -100 and 100 .

We first test our method without pre-processing the samples as in the problem with Brownian motion. As we can see in the second row of Figure 3.3b, the inferences are far away from the ground truth. This is due to the heavy tail of ρ_t in the Lévy process: some samples far away from 0, although rare, could dominate the loss function and spoil the training. To deal with this problem, we then apply a bounded map $h(x) = 2 \tanh(x/2)$ to all the generated and real samples before feeding them to d. The results are shown in the first row of Figure 3.3b where we can see the inferences are much improved. In case 2 where the drift is parameterized by neural networks, the inferred drift outside $[-1.5, 1.5]$ is better than that in the problem with Brownian motion. This is because the samples are more scattered in the Lévy case.

3.4.2 2D Problems: Various Scenarios of Observations

In this section, we test our method on 2D problems using the sliced Wasserstein distance as $\mathbf{d}$, with various scenarios of observations. We consider the following 2D SODE:

$$d\mathbf{X}_t = \boldsymbol{\mu}(\mathbf{X}_t)dt + \begin{bmatrix} s_0 & 0 \\ s_1 & s_2 \end{bmatrix} d\mathbf{B}_t \quad (3.16)$$

where

$$\boldsymbol{\mu}(\mathbf{x}) = \nabla_{\mathbf{x}}\varphi(\mathbf{x}), \quad (3.17)$$

and

$$\begin{aligned} \varphi(\mathbf{x}) = & -(x_1 + a_0)^2(x_2 + a_1)^2 \\ & - (x_1 + a_2)^2(x_2 + a_3)^2 \quad \text{for } \mathbf{x} = (x_1, x_2). \end{aligned} \quad (3.18)$$

The parameters are set as $a_0 = a_1 = s_0 = s_1 = s_2 = 1.0$, $a_2 = a_3 = -0.5$. We set the initial distribution as $\rho_0 = \mathcal{N}(0, \mathbf{I}_2)$. We assume we know that the diffusion coefficient is a constant lower triangular matrix but we need to infer the three unknown parameters s_0, s_1, s_2 . In particular, we use $(\text{softplus}(\tilde{s}_0), \tilde{s}_1, \text{softplus}(\tilde{s}_2))$ to approximate (s_0, s_1, s_2) , where $\tilde{s}_0, \tilde{s}_1, \tilde{s}_2$ are three trainable variables. Similar as in the 1D problems, we consider the following two cases for the drift. In case 1, we know the form of $\boldsymbol{\mu}$ and φ in Equation 3.17 and 3.18 but we need to infer a_0, a_1, a_2 and a_3 ((a_0, a_1) and (a_2, a_3) are exchangeable). In case 2, we only know that $\boldsymbol{\mu}$ is a gradient of φ in Equation 3.17, but we have no knowledge of φ . In this case, we use a neural network to parameterize φ .

For the training data, we prepare 10^5 sample paths, and consider the following various scenarios of observations at $t = 0, 0.1, 0.2, 0.3, 0.5, 0.7, 1.0$. The data are visualized in Section 3.6.3.

- Scenario 1: we assume that our observations are ideal, i.e., we make accurate observations of all the particle coordinates as the training data.
- Scenario 2: we assume that our observations are noisy. Specifically, we make observations of all the particle coordinates, but each coordinate is perturbed by an i.i.d. random noise $\mathcal{N}(0, e^2 \mathbf{I}_2)$, where e is set as 0.2 but also need to be inferred during learning.
- Scenario 3: we assume that our observations are truncated. Specifically, we make observations of the particle coordinates in $\Omega = (\infty, 0.5) \times \mathbb{R}$, with the particles outside of Ω dropped.

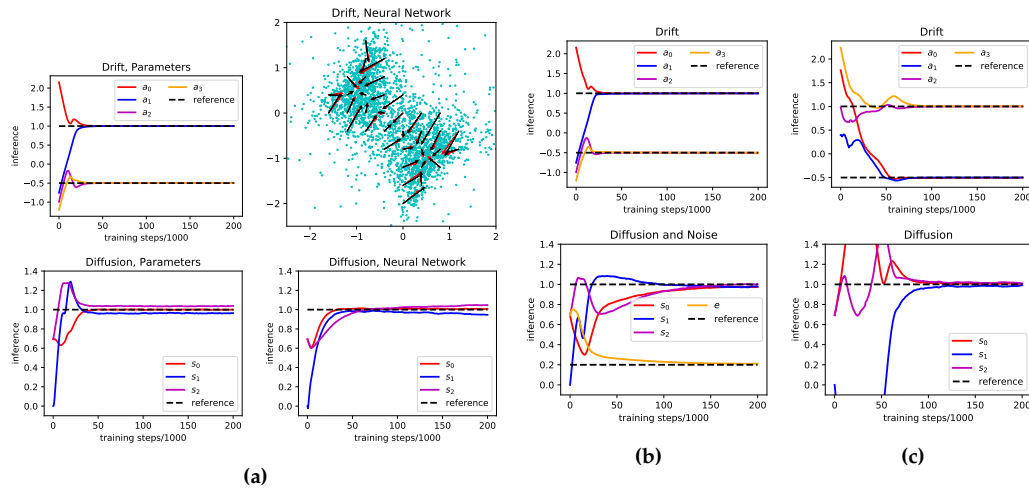


Figure 3.4: Inferred drift and diffusion in 2D problems, with (a) ideal observations, (b) noisy observations, and (c) truncated observations. In (a), the left column shows the results in case 1 where the drift is parameterized by four parameters, while the right column shows the results in case 2 where the drift is parameterized by a neural network. In the top right figure of (a), we visualize the drift field where the training data are dense. The red and black arrows represent the inferred drift and the exact drift, respectively. For each arrow, the length represents the norm of the drift, scaled by 0.1. The green dots are samples from the merged training data.

For the first scenario, the inferred drift and diffusion are illustrated in Figure 3.4a. In case 1, all the inferred drift and diffusion parameters approach the ground truth during the training, with relative error less than 0.3% for each a_i and less than 4% for each σ_i after 2×10^5 training steps. In case 2, the inferred diffusion

parameters are also close to the ground truth, with error comparable with that in case 1. In the region where the training data are dense (i.e., the density estimated from all the training data is no less than 0.05), the inferred drift field has a relative L_2 error about 13%. The inference is much worse in other regions, and the reason is the same as in the 1D problem: the neural network can hardly learn the drift field where the training data are sparse.

For the second and third scenarios, we apply the technique of perturbing and filtering the generated samples, respectively. In Figure 3.4b and Figure 3.4c we show the results of case 1 with the drift parameterized by four parameters. After 2×10^5 training steps, all the parameters converge to ground truth. For scenario 2, the error is less than 0.5% for each drift parameter a_i , less than 3% for each diffusion parameter s_i , and about 5% for the noise parameter e . For scenario 3, the error is less than 0.2% for each drift parameter a_i , less than 1% for each diffusion parameter s_i . We remark that we failed to learn the drift with neural network parameterization in the second and third scenario, suggesting a room for further improvements.

3.4.3 Higher Dimensional Problems

In this section, we test our method on higher dimensional non-interacting particle systems. Note that the sliced Wasserstein distance does not perform well in high dimensional problems, and we thus switch to the WGAN-GP to provide d. The comparison between the sliced Wasserstein distance and WGAN-GP in high dimensional problems is presented in Section 3.6.1.

We consider a d -dimensional SODE:

$$d\mathbf{X}_t = \boldsymbol{\mu}(\mathbf{X}_t)dt + \boldsymbol{\sigma}d\mathbf{B}_t \quad (3.19)$$

where

$$\mu^{(i)}(\mathbf{X}_t) = X_t^{(i)} - (X_t^{(i)})^3, i = 1, 2, \dots, d, \quad (3.20)$$

for $\mu^{(i)}$ as the i -th component of $\boldsymbol{\mu} \in \mathbb{R}^d$, and $X_t^{(i)}$ is the i -th component of $\mathbf{X}_t \in \mathbb{R}^d$.

We set the diffusion coefficient matrix as

$$\boldsymbol{\sigma} = \begin{bmatrix} s_1 & 0 & 0 & 0 & \cdots & 0 \\ s'_2 & s_2 & 0 & 0 & \cdots & 0 \\ 0 & s'_3 & s_3 & 0 & \cdots & 0 \\ 0 & 0 & s'_4 & s_4 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s'_d & s_d \end{bmatrix}, \quad (3.21)$$

where the $2d-1$ nonzero entries $\{s_i\}_{i=1}^d$ and $\{s'_i\}_{i=2}^d$ are set as 1. The initial distribution is $\rho_0 = \mathcal{N}(\mathbf{0}, 0.04\mathbf{I}_d)$. Due to the non-diagonal diffusion coefficient matrix, the motion in different dimensions are coupled.

We prepare 10^5 sample paths and observe all the particle positions at $t = 0.2, 0.5, 1.0$ as our training data. We use a cubic polynomial with four variables to parameterize $\mu^{(i)}$ as a function of $X_t^{(i)}$ for each i , while using $2d-1$ trainable variables to learn the nonzero entries in the diffusion coefficient matrix, with the diagonal entries rectified by a softplus function for positivity. In other words, we use $6d-1$ variables to parameterize the drift and diffusion for the d -dimensional SODE. The results are shown in Figure 3.5. Even for the 100-dimensional problem, after 6×10^5 training steps, the average error of the diffusion coefficients is about

0.04, and the average relative L_2 error of the drift in the interval $[-3, 3]$ in each dimension is about 14%.

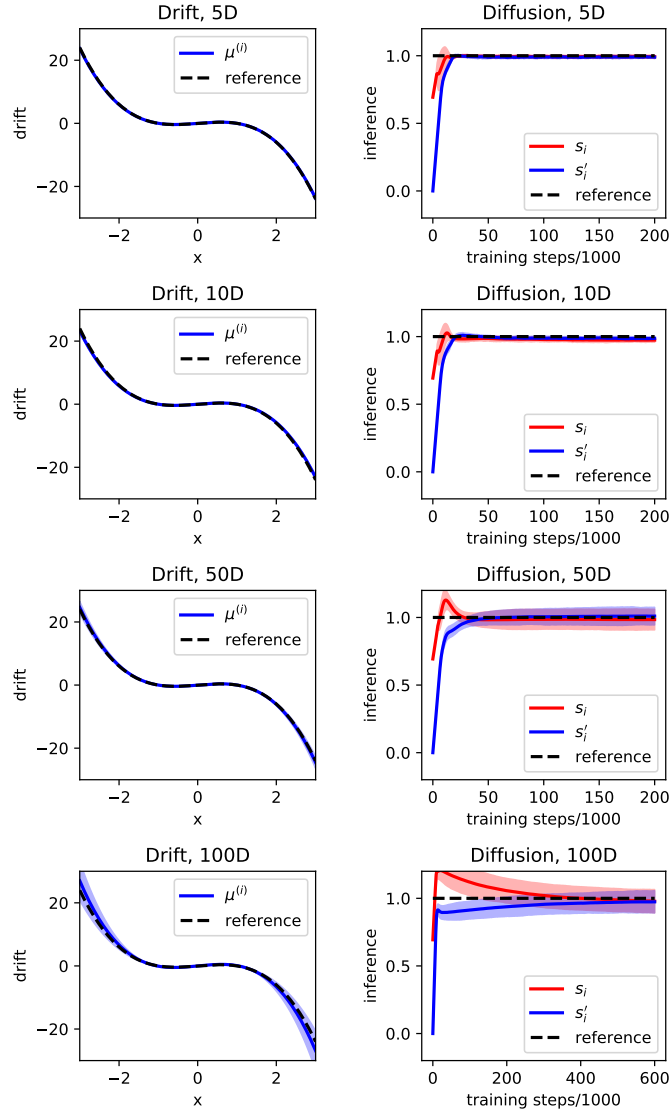


Figure 3.5: Inferred drift in the end of training and diffusion parameters during training, in 5D, 10D, 50D and 100D problems. The solid lines and shaded areas are the mean and two standard deviations over all i and three runs with different random seeds.

It may appear surprising that we can solve a 100-dimensional problem with only 10^5 samples since usually an exponentially large number of samples is required to describe the distribution. However, we remark that the difficulty of the learning task is significantly reduced since our partial knowledge, i.e., the form of

the SODE, is encoded into the generator.

3.4.4 Interacting Particle Systems

In this section we consider the 1D and 2D cases of the interacting particle system with governing equations 3.3. To prepare the training data, we follow [80] and set the initial condition, including the density and velocity as

$$\begin{aligned}\rho_0(x) &= \mathbb{1}_{[-0.75, 0.75]} \frac{\pi}{3} \cos\left(\frac{2\pi x}{3}\right), \\ \mu_0(x) &= -\frac{1}{2} \sin\left(\frac{2\pi x}{3}\right),\end{aligned}\tag{3.22}$$

for the 1D case and

$$\begin{aligned}\rho_0(x_1, x_2) &= \mathbb{1}_{[-0.75, 0.75]^2} \left(\frac{\pi}{3}\right)^2 \cos\left(\frac{2\pi x_1}{3}\right) \cos\left(\frac{2\pi x_2}{3}\right), \\ \mu_0(x_1, x_2) &= \left(-\frac{1}{2\sqrt{2}} \sin\left(\frac{2\pi x_1}{3}\right), -\frac{1}{2\sqrt{2}} \sin\left(\frac{2\pi x_2}{3}\right)\right)\end{aligned}\tag{3.23}$$

for the 2D case, where $\mathbb{1}$ is the indicator function. Using the Velocity-Verlet method with time step 0.01, we perform simulations with 1024 and 9976 particles for the 1D and 2D cases, respectively, and generate data at $t = 0.5, 0.6, \dots, 2.0$, i.e., 16 snapshots in total. The input radius r of the influence function $\phi(r)$ is clipped to be at least $r_{\min} = 0.01$, both in simulation and learning, to avoid the singularity. While using the same number of particles and make observations at the same time instants as in [80], our method does not require knowledge of the initial condition or the velocity field in the data snapshots, as opposed to their method based on Bayesian optimization to infer α .

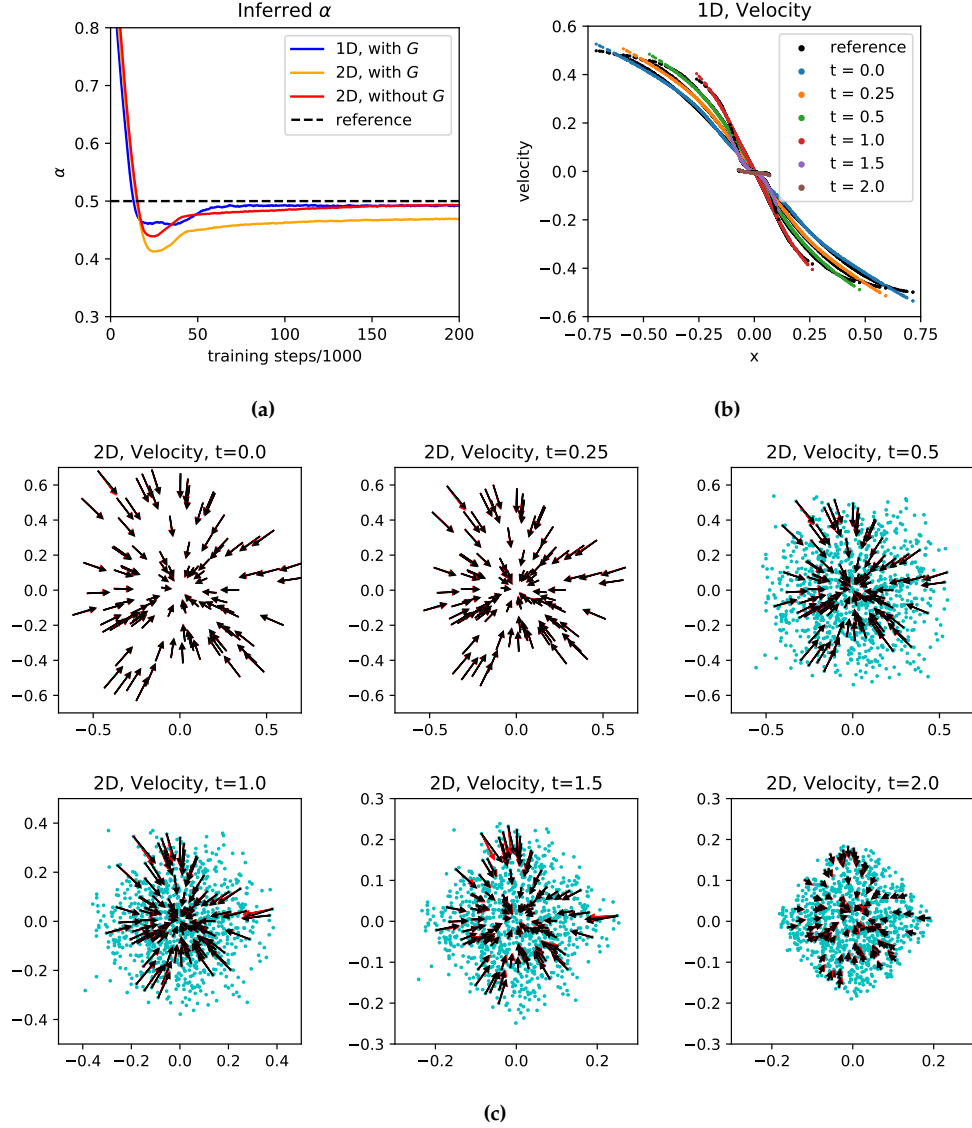


Figure 3.6: Results for the 1D and 2D problems of interacting particle systems. (a): The inferred alpha in 1D and 2D problems. (b): The inferred velocity field against the reference velocity from simulation in the 1D problem. The dots also show the distribution of training data. (c): The inferred velocity field (red arrows) against the reference velocity from simulation (black arrows) in the 2D problem. For each arrow, the length represents the norm of the velocity scaled by 0.3. The green dots are random samples from training data.

We apply the sliced Wasserstein distance for the distribution loss, with the batch size equal to the number of particles in training data. When calculating the distance at each time instant, the generated and real distributions are normalized with the mean and standard deviation of the real distribution. For the Newton

loss, we set $K = 16$, $M = 10000$ and $B = 10$. We use $2 \times \text{sigmoid}(\beta)$ to represent the inferred α so that the inference is bounded by 0 and 2, and the variable β is initialized as 0.

The results are visualized in Figure 3.6. For the 1D case, the inferred α is 0.492 with ground truth 0.5 in the end of training. As a comparison, the inference is 0.480 for the 1D case in [80]. Our inferred velocity field also matches well with the reference ground truth at different time instants, even at $t = 0$ and 0.25 when we have no observations at all. This is because we incorporated the knowledge of dynamics in the learning system. For the 2D case, while the inferred velocity field also matches well with the ground truth, the inferred α is 0.469 with ground truth 0.5 in the end of training, i.e., about 6% error, much larger than in the 1D case. We attribute the error to the fact that the singularity problem for $\phi(r)$, where the order for r is dimension-dependent, is more severe in the 2D case. In Section 3.6.4, we show that as we increase $r_{\min}$ from 0.01 to 0.1, the error is reduced from 6% to 2%. Alternatively, in the 2D case we also try to directly use the data at $t = 0.5$ as the starting coordinates to generate trajectories for $t \geq 0.5$ (M is also reduced to the data size 9976), so that the initial generator G is removed from the generative model. By doing so, the number of trainable variables is reduced and the learning becomes easier. The inferred α is 0.493 in the end of training. As a comparison, the inference is 0.513 for the 2D case in [80]. However, we remark that while this strategy helps to infer α without extra data, we cannot infer the velocity or density for $0 \leq t < 0.5$.

While unpaired observations of particle positions are sufficient for the inference of velocity field and α in this problem, we remark that for other problems, it might be difficult or even impossible to infer particle dynamics with these data. For such cases, additional information of the velocity field or paired observations may be

helpful.

3.5 Paired Observations

We should note that the convergence of the marginal distribution in each snapshot does not necessarily lead to the convergence of the joint distribution of coordinate tuples $(\mathbf{X}_{t_1}, \mathbf{X}_{t_2}, \dots, \mathbf{X}_{t_n})$. While the joint distribution is not available for unpaired observations that we are focused on so far, in other cases where the observed particle coordinates can be paired across snapshots, it is possible to improve the inference by fitting the joint distribution of coordinate tuples with the corresponding generated joint distribution.

In [26], the authors pointed out that Wasserstein convergence of the distribution of $(\mathbf{X}_{t_1}, \mathbf{X}_{t_2}, \dots, \mathbf{X}_{t_n})$ is equivalent to Wasserstein convergence of the distributions of $(\mathbf{X}_{t_i}, \mathbf{X}_{t_{i+1}})$ for all $i = 1, 2, \dots, n - 1$, if $(\mathbf{X}_{t_1}, \mathbf{X}_{t_2}, \dots, \mathbf{X}_{t_n})$ is a Markov chain. However, the sample spaces for $\{\mathbf{X}_{t_i}\}_{i=1}^n$ are limited to be finite and discrete in [26], thus the result doesn't apply to dynamic systems in continuous spaces. Here, we present a new theorem for continuous sample spaces:

Theorem 1. *Let $(X_1, X_2, \dots, X_T)$ be a Markov chain of length $T \geq 3$ and we use $X_{i:j}$ to denote the nodes $(X_i, X_{i+1}, \dots, X_j)$, for $i \leq j$. Suppose the domain D_t for X_t is a compact subset of $\mathbb{R}^{d_t}$ for $t = 1, 2, \dots, T$. We use the l_q ($q \geq 1$) Euclidean metric for all the Euclidean spaces with different dimensions.*

Let $\{P_n^{X_{i:j}}\}_{n=1}^\infty$ and $P^{X_{i:j}}$ be probability measures of $X_{i:j}$ for $i \leq j$, $P_n^{X_i|X_j}$ and $P^{X_i|X_j}$ be the corresponding probability transition kernels. If $P_n^{X_{t:t+1}}$ converges to $P^{X_{t:t+1}}$ in Wasserstein- p ($p \geq 1$) metric for all $t = 1, 2, \dots, T - 1$, $P_n^{X_t|X_{t+1}}$ and $P^{X_{t+2}|X_{t+1}}$ as functions of X_{t+1} are C-

Lipschitz continuous in Wasserstein- p metric for all $t = 1, 2, \dots, T - 2$ and n , where C is a constant, then $P_n^{X_{1:T}}$ converges to $P^{X_{1:T}}$ in Wasserstein- p metric.

We present the proof for Theorem 1 in Section 3.6.5. The assumption on the continuity of the probability transition kernels is not required for finite discrete sample spaces in [26], but the theorem does not hold without it for continuous sample space. We provide a counter-example in Section 3.6.6.

The theorem states that under certain conditions, we can set our goal as fitting the distributions of $(\mathbf{X}_{t_i}, \mathbf{X}_{t_{i+1}})$, i.e., coordinate pairs from adjacent snapshots. This should be easier compared with directly fitting the distribution of $(\mathbf{X}_{t_1}, \mathbf{X}_{t_2}, \dots, \mathbf{X}_{t_n})$, since the effective dimensionality is reduced. We still can view this approach as “ensemble-regression”, except that instead of a curve, we try to fit the data with a 2D surface $(t_i, t_j) \rightarrow \rho_{t_i, t_j}$ in the probability measure space, where ρ_{t_i, t_j} denotes the joint distributions of $(\mathbf{X}_{t_i}, \mathbf{X}_{t_j})$.

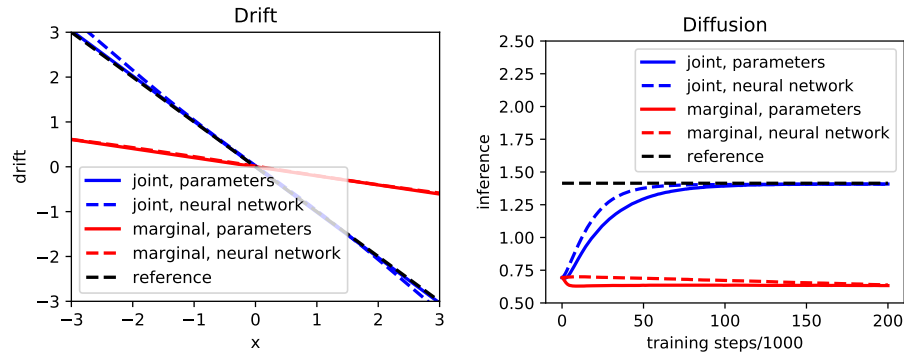


Figure 3.7: Inferred drift function in the end of training and diffusion coefficient during training for the OU process problem, using a linear function or a neural network to parameterize the drift function.

As an illustration, we study the 1D Ornstein–Uhlenbeck (OU) process:

$$dX_t = -X_t dt + \sqrt{2} dW_t, \quad (3.24)$$

with $\rho_0 = \mathcal{N}(0, 1)$, so that $\rho_t = \mathcal{N}(0, 1)$ for any $t > 0$. This is a special example as the governing SODE is not unique given ρ_t . We make observations of 100 particles at $t = 0.1, 0.2, 0.3, 0.4, 0.5, 0.75, 1.0$ as data, and compare the inferences by fitting the marginal distributions of individual coordinates or the joint distributions of adjacent coordinate pairs using the SW distance. The drift function is parameterized by a linear function $y(x) = ax$ or a neural network, while the diffusion coefficient is represented by a trainable variable rectified by a softplus function. We present the results in Figure 3.7, where we can clearly see the failure in the cases of fitting the marginal distributions, while fitting the joint distributions works very well.

3.6 Supplementary Material

3.6.1 Sliced Wasserstein Distance and Comparison with WGAN-GP

Recall that the squared sliced Wasserstein-2 distance is the expectation of the squared Wasserstein-2 distance between the two input measures projected onto uniformly random directions. To estimate $SW_2(\mu, \nu)$ from samples of μ and ν , we use the following process introduced in [25].

- Draw samples independently from μ and ν with batch size b , denote as $\mathcal{U}$ and $\mathcal{V}$.
- Uniformly sample m projection directions $\{e_j\}_{j=1}^m$ in $\mathbb{R}^d$. In this chapter we set $m = 1000$.

- For each random direction e_j , project and sort the samples in $\mathcal{U}$ and $\mathcal{V}$ in the direction of e_j , getting $\{u_{i,j}\}_{i=1}^b$ and $\{v_{i,j}\}_{i=1}^b$, where $u_{i,j} \leq u_{i+1,j}$ and $v_{i,j} \leq v_{i+1,j}$ for $i = 1, 2, \dots, b-1$. Calculate $L_j = \sum_{i=1}^b (u_{i,j} - v_{i,j})^2 / b$.
- Calculate $L = \sum_{j=1}^m L_j / m$ as the estimation of squared $SW_2(\mu, \nu)$.

Compared with GANs, the sliced Wasserstein distance does not need discriminators, and is more robust than WGAN-GP in low dimensional problems. Take the following 1D problem as an example. We consider the SODE:

$$dX_t = adt + bdB_t, \quad t \geq 0, \quad (3.25)$$

with $\rho_0 = \mathcal{N}(-0.5, 0.5)$. We set $a = b = 1$ so that the exact solution is

$$\rho_t = \mathcal{N}(t - 0.5, t + 0.5). \quad (3.26)$$

We have 10,000 samples at $t = 0.5$ and 1.5 , respectively, as the training data, and wish to infer the constant drift and diffusion coefficients. The input noise to the generator is uniform noise from -1 to 1. The results are shown in Figure 3.8.

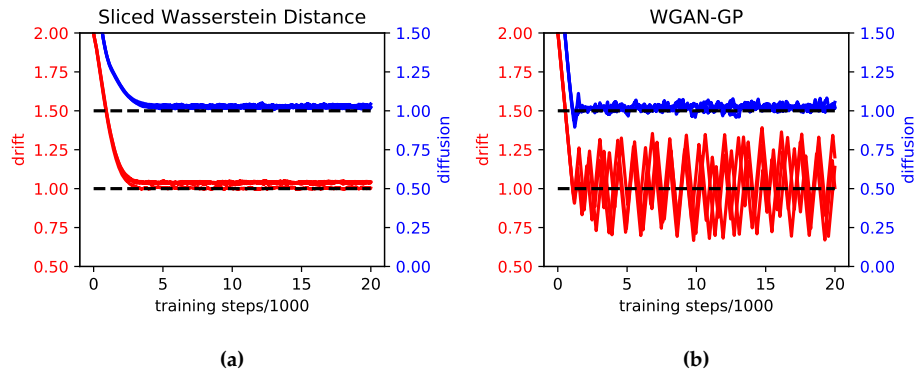


Figure 3.8: Inferred drift and diffusion coefficient during the training, using (a) sliced Wasserstein distance and (b) WGAN-GP. The black dashed lines represent the ground truth, while the multiple colored lines represent the results in three independent runs.

We can clearly see oscillations of the inferred drift coefficient for WGAN-

GP. This can be attributed to the two-player game between the generator and discriminator, which was also reported in [23].

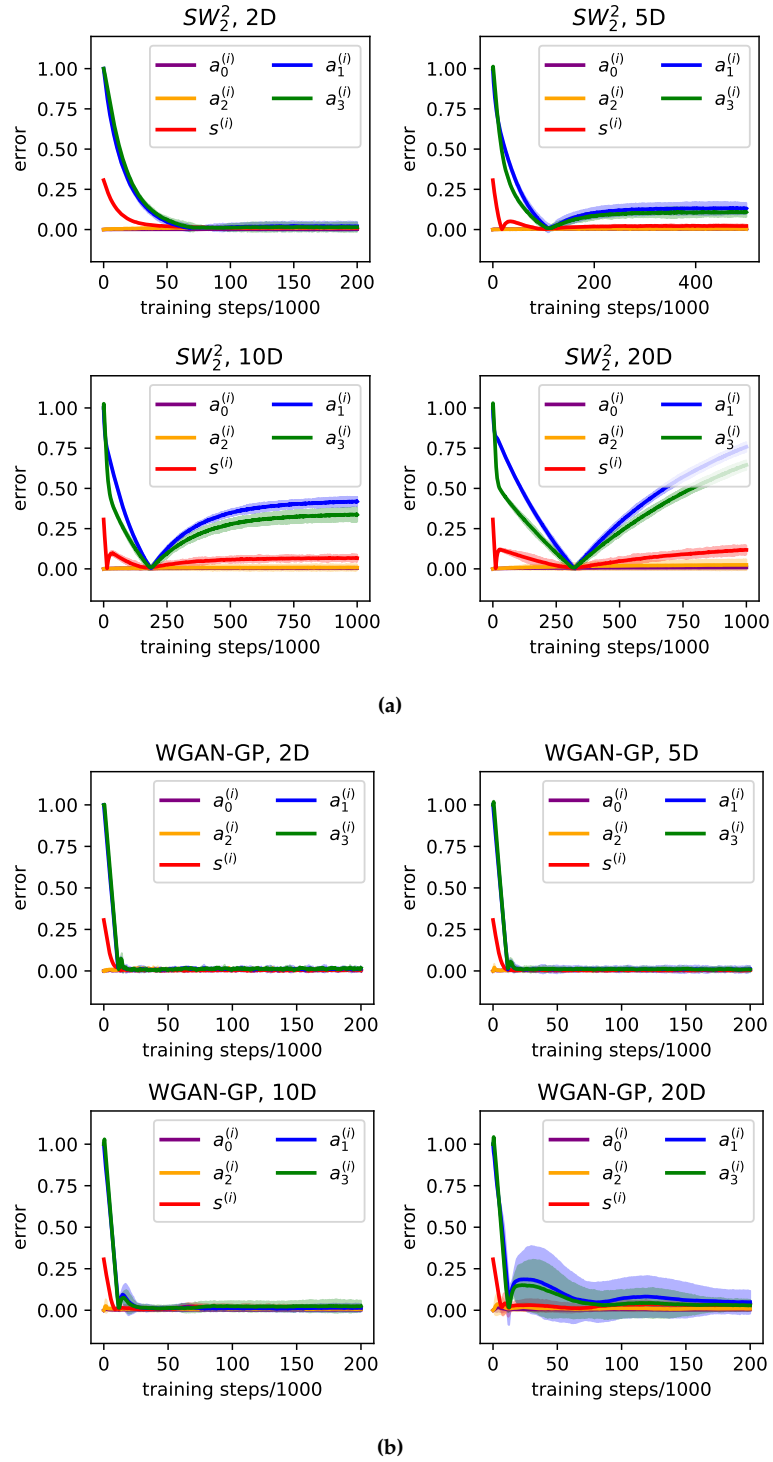


Figure 3.9: Results of 2D, 5D, 10D and 20D problems, using (a) the squared sliced Wasserstein-2 distance or (b) WGAN-GP for d . The solid lines and shaded areas represent the mean and two standard deviations of the errors over all dimensions.

We did not observe significant oscillations using WGAN-GP in higher dimensional problems. Indeed, WGAN-GP outperforms the sliced Wasserstein distance in high dimensional problems. Taking the following simple SODE problem as an example, where the motions are uncoupled between dimensions:

$$dX_t^{(i)} = (X_t^{(i)} - (X_t^{(i)})^3)dt + dB_t^{(i)}, i = 1, 2, \dots, d \quad (3.27)$$

where $X_t^{(i)}$ is the i -th component of $\mathbf{X}_t \in \mathbb{R}^d$, with $\rho_0 = \mathcal{N}(0, 0.04\mathbf{I}_d)$. We prepare 10^5 sample paths and observe all the particle positions at $t = 0.2, 0.5, 1.0$ as our training data. We use a cubic polynomial $a_0^{(i)} + a_1^{(i)}x + a_2^{(i)}x^2 + a_3^{(i)}x^3$ to parameterize the i -th component of the drift. We use another trainable variable rectified by a softplus function to represent the diffusion coefficient $s^{(i)} = 1$ in each direction. In total, $5d$ variables are used to parameterize the d -dimensional SODE.

The results are shown in Figure 3.9. While the SW distance works for 2D problems, it does not scale well to high dimensional problems, for which WGAN-GP gives much better inferences.

3.6.2 A Study on the Density Estimation

In this section, we wish to show that our method can make use of the SODE and multiple snapshots to reduce the error of density estimation from limited samples. We consider the one-dimensional SODE:

$$dX_t = (4X_t - X_t^3)dt + 0.4dB_t, \quad t \geq 0, \quad (3.28)$$

with $\rho_0 = 0.5\mathcal{N}(-0.5, 0.3^2) + 0.5\mathcal{N}(0.5, 0.3^2)$.

We test the following two cases of training data sets. In both cases we have 1000 samples at $t = 0.05, 0.1, 0.15, 0.2, 0.25$, but the particle trajectories are different.

- Case 1: We make observations from different sets of trajectories at different time instants.
- Case 2: We make observations from the same set of trajectories at different time instants.

We assume that we know Equation 3.28, but have no knowledge of ρ_0 . We could make the analogy of performing linear regression, where we know the slope but do not know the intercept. In Figure 3.10 we show the comparison between the inferred densities and the densities estimated directly from data. All the densities are estimated using Gaussian kernel density estimation. The inferred densities and the ground truth come from 10^5 samples produced by the generative model or simulation, with bandwidth decided by the Scott's rule. To remove the effect of bandwidth selection in the density estimation directly from data, we perform a grid search of the optimal bandwidth factor via the L_2 error against ground truth, from 0.01 to 0.5 with grid size 0.01 (Scott's rule suggests about 0.25).

The grid search strategy is actually infeasible in practice since we don't know the ground truth, but it should perform better than any bandwidth selector. Despite that, in Figure 3.10a we can clearly see that in case 1 our inferred densities with naive Scott's rule significantly outperform the ones estimated directly from training data with grid search. This cannot be attributed to the number of samples, since in case 2 the inferred densities with our method cannot perform better, as shown in Figure 3.10b.

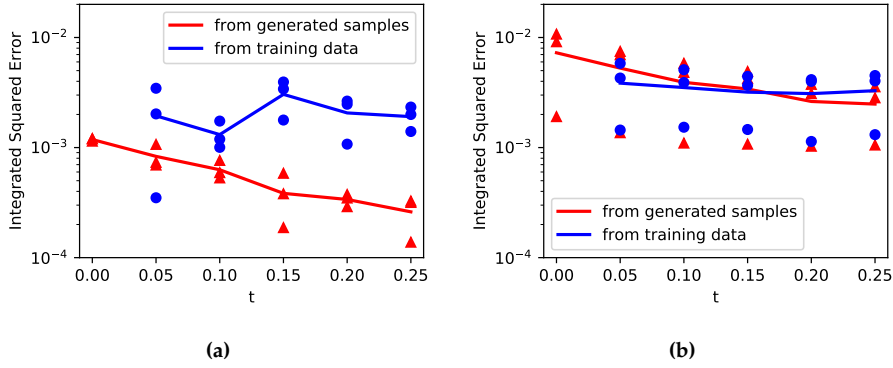


Figure 3.10: Integrated squared errors (ISE) of the inferred density functions at different time instants in (a) case 1 and (b) case 2. The squared errors are integrated from -4 to 4. The colored lines show the ISE averaged over three independent runs, while the markers with the corresponding colors show the ISE in each run.

The different performances in two cases are reasonable. In case 1, our method can utilize the SODE and the observations at multiple time instants to reduce the error from limited samples. As we infer the density at $t = 0.05$, we are not only utilizing the data at $t = 0.05$ but also implicitly taking advantage of the data at later time instants. In case 2, since the observations come from the same sample paths, the observations at later time instants cannot provide additional information considering that the SODE is already known. Let us again make an analogy in the context of linear regression: if the observations have independent noise, multiple observations will be more helpful than a single observation. However, if the observations have the same noise, multiple observations cannot help us more than a single observation, if we already know the slope.

3.6.3 Training data in 2D Problems with Various Scenarios of Observations

In Figure 3.11 we visualize the training data in 2D problems with various scenarios of observations.

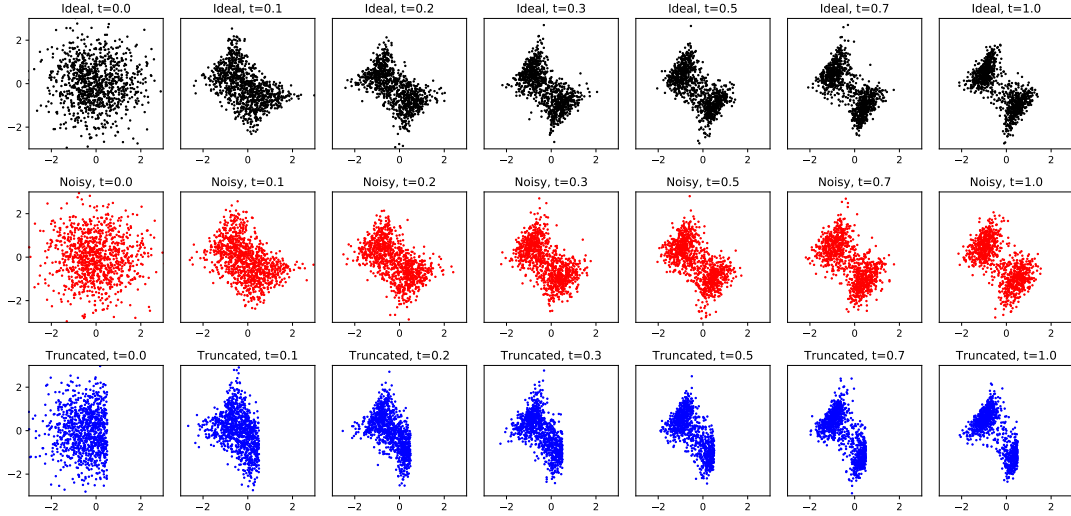


Figure 3.11: Samples from the training data in 2D problems with ideal, noisy, or truncated observations.

3.6.4 Effect of Clipping Radius in the Interacting Particle System

In this section we study the effect of various clipping radius $r_{\min}$ from 0.01 to 0.1, both in the simulation for data generation and the learning algorithm. We remark that changing $r_{\min}$ in this range would not make a huge difference to the system behavior, since the dynamics are dominated by non-local interactions. In Figure 3.12 we show the inferred α during the training. As $r_{\min}$ increase from 0.01 to 0.1, the inferred α in the end of training increases from 0.469 to 0.490, with the ground truth 0.5. This supports that the error mainly comes from the singularity

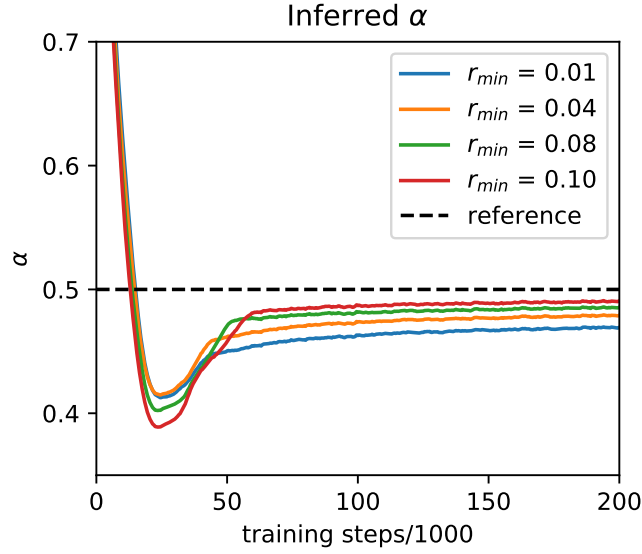


Figure 3.12: Inferred α in the 2D interacting particle system during training, with different clipping radius $r_{\min}$.

problem of $\phi(r)$ when r is small.

3.6.5 Proof for Theorem 1

The proof is based on the weak convergence: convergence of a sequence of probability measures $\{P_n\}_{n=1}^{\infty}$ to a probability measure P in the Wasserstein- p metric is equivalent to the weak convergence plus the convergence of p -th moments of $\{P_n\}_{n=1}^{\infty}$ to P [130]. Here, the weak convergence means $E_{P_n}[f(x)]$ converges to $E_P[f(x)]$ for each $f \in \mathcal{F}$, where $\mathcal{F} = \{\text{all bounded and continuous functions}\}$ or $\mathcal{F} = \{\text{all bounded and Lipschitz functions}\}$, which are equivalent[57]. Note that in Theorem 1 we assume that the domain for X_t is a compact subset of Euclidean space for each t , thus $\|X_{1:T}\|^p$ is a bounded and continuous function of $X_{1:T}$. The convergence of p -th moment then directly comes from the weak convergence, i.e., we only need to show that $\{P_n\}_{n=1}^{\infty}$ converges to P weakly. This is proved in Lemma 4 below.

The notations are inherited from Theorem 1. For simplicity, we will sometimes use X, Y, Z to represent the nodes in sequence, and use P^{XY} to represent the probability measure of (X, Y) , etc.

Lemma 1. *Let P_y mapping from the Euclidean space to the probability measure space be C -Lipschitz in Wasserstein- p metric ($p \geq 1$). For any $f(x, y, z) : \mathbb{R}^{d_x} \times \mathbb{R}^{d_y} \times \mathbb{R}^{d_z} \rightarrow \mathbb{R}$ K -Lipschitz,*

$$g(x, y) = \int f(x, y, z) dP_y(z)$$

is $(C + 1)K$ -Lipschitz.

Proof. Note that

$$\begin{aligned} & |g(x + \epsilon, y + \xi) - g(x, y)| \\ &= \left| \int f(x + \epsilon, y + \xi, z) dP_{y+\xi}(z) - \int f(x, y, z) dP_y(z) \right| \\ &\leq \left| \int f(x + \epsilon, y + \xi, z) dP_{y+\xi}(z) - \int f(x + \epsilon, y + \xi, z) dP_y(z) \right| + \\ &\quad \left| \int f(x + \epsilon, y + \xi, z) dP_y(z) - \int f(x, y, z) dP_y(z) \right|. \end{aligned} \tag{3.29}$$

Firstly

$$\begin{aligned} & \left| \int f(x + \epsilon, y + \xi, z) dP_{y+\xi}(z) - \int f(x + \epsilon, y + \xi, z) dP_y(z) \right| \\ &\leq KW_1(P_y, P_{y+\xi}) \\ &\leq KW_p(P_y, P_{y+\xi}) \\ &\leq CK\|\epsilon\|, \end{aligned} \tag{3.30}$$

where the first inequality comes from the Kantorovich–Rubinstein formula [130] and that $f(x + \epsilon, y + \xi, z)$ is K -Lipschitz. The last inequality comes from that P_y is C -Lipschitz in Wasserstein- p sense.

Secondly,

$$\begin{aligned}
& \left| \int f(x + \epsilon, y + \xi, z) dP_y(z) - \int f(x, y, z) dP_y(z) \right| \\
& \leq \int |f(x + \epsilon, y + \xi, z) - f(x, y, z)| dP_y(z) \\
& \leq \int K \|(\epsilon, \xi)\| dP_y(z) \\
& = K \|(\epsilon, \xi)\|.
\end{aligned} \tag{3.31}$$

We conclude that

$$|g(x + \epsilon, y + \xi) - g(x, y)| \leq (C + 1)K \|(\epsilon, \xi)\|, \tag{3.32}$$

i.e. $g(x, y)$ is $(C + 1)K$ -Lipschitz □

Lemma 2. *If P_n^{XY} converge to P^{XY} weakly, then P_n^X converge to P^X weakly.*

Proof. For any bounded and continuous function $f(x)$

$$\begin{aligned}
& \int f(x) P_n^X(x) - \int f(x) P^X(x) \\
& = \iint f(x) P_n^{XY}(x, y) - \iint f(x) P^{XY}(x, y) \rightarrow 0.
\end{aligned} \tag{3.33}$$

The convergence comes from the weak convergence of P_n^{XY} and that $f(x)$ is bounded and continuous as a function of x and y . □

Lemma 3. *Suppose P_n^{YZ} converge to P^{YZ} weakly, $P^{Z|Y=y}$ is Lipschitz in Wasserstein- p metric. For any $g(y, z) : \mathbb{R}^{d_y} \times \mathbb{R}^{d_z} \rightarrow \mathbb{R}$ bounded and Lipschitz, we have*

$$\begin{aligned}
& \iint g(y, z) dP_n^{Z|Y=y}(z) dP_n^Y(y) \\
& - \iint g(y, z) dP^{Z|Y=y}(z) dP_n^Y(y) \rightarrow 0.
\end{aligned} \tag{3.34}$$

Proof. Since $g(y, z)$ is bounded and continuous, and P_n^{YZ} converge to P^{YZ} weakly,

$$\iint g(y, z) dP_n^{YZ}(y, z) - \iint g(y, z) dP^{YZ}(y, z) \rightarrow 0, \quad (3.35)$$

i.e.

$$\begin{aligned} & \iint g(y, z) dP_n^{Z|Y=y}(z) dP_n^Y(y) \\ & - \iint g(y, z) dP^{Z|Y=y}(z) dP^Y(y) \rightarrow 0. \end{aligned} \quad (3.36)$$

Since $g(y, z)$ is bounded and Lipschitz, $P^{Z|Y=y}$ is Lipschitz, we have

$$\int g(y, z) dP^{Z|Y=y}(z) \quad (3.37)$$

is Lipschitz. Further since P_n^Y converge to P^Y weakly (from Lemma 2):

$$\begin{aligned} & \iint g(y, z) dP_n^{Z|Y=y}(z) dP_n^Y(y) \\ & - \iint g(y, z) dP^{Z|Y=y}(z) dP^Y(y) \rightarrow 0. \end{aligned} \quad (3.38)$$

Combining 3.36 and 3.38, we have

$$\begin{aligned} & \iint g(y, z) dP_n^{Z|Y=y}(z) dP_n^Y(y) \\ & - \iint g(y, z) dP^{Z|Y=y}(z) dP_n^Y(y) \rightarrow 0. \end{aligned} \quad (3.39)$$

□

Lemma 4. Let $(X_1, X_2, \dots, X_T)$ be a Markov chain of length $T \geq 3$ and we use $X_{i:j}$ to denote the nodes $(X_i, X_{i+1}, \dots, X_j)$, for $i \leq j$. Suppose the domain D_t for X_t is a compact subset of $\mathbb{R}^{d_t}$ for $t = 1, 2, \dots, T$. We use the l_q ($q \geq 1$) Euclidean metric for all the Euclidean spaces with different dimensions.

Let $\{P_n^{X_{i:j}}\}_{n=1}^\infty$ and $P^{X_{i:j}}$ be probability measures of $X_{i:j}$ for $i \leq j$, $P_n^{X_i|X_j}$ and $P^{X_i|X_j}$ be the corresponding probability transition kernels. If $P_n^{X_{t:t+1}}$ converges to $P^{X_{t:t+1}}$ weakly for all $t = 1, 2, \dots, T-1$, $P_n^{X_t|X_{t+1}}$ and $P^{X_{t+2}|X_{t+1}}$ are C -Lipschitz continuous in Wasserstein- p metric for all $t = 1, 2, \dots, T-2$ and n , where C is a constant, then $P_n^{X_{1:T}}$ converges to $P^{X_{1:T}}$ weakly.

Proof. We start from the case $T = 3$. For simplicity, we use X, Y, Z to denote the nodes in sequence.

For any $f(x, y, z)$ K -Lipschitz and bounded, we have

$$\begin{aligned}
 & \int f(x, y, z) dP_n^{XYZ}(x, y, z) \\
 &= \int \left(\int f(x, y, z) dP_n^{Z|Y=y}(z) \right) dP_n^{XY}(x, y) \\
 &= \int (g_n(x, y) - g(x, y)) dP_n^{XY}(x, y) \\
 &+ \int g(x, y) dP_n^{XY}(x, y),
 \end{aligned} \tag{3.40}$$

where

$$\begin{aligned}
 g(x, y) &= \int f(x, y, z) dP^{Z|Y=y}(z) \\
 g_n(x, y) &= \int f(x, y, z) dP_n^{Z|Y=y}(z).
 \end{aligned} \tag{3.41}$$

From Lemma 1, since $f(x, y, z)$ is Lipschitz, $P^{Z|Y=y}$ is Lipschitz in Wasserstein- p sense, we have $g(x, y)$ is Lipschitz. $g(x, y)$ is also bounded since $f(x, y, z)$ is bounded. So we have

$$\int g(x, y) dP_n^{XY}(x, y) \rightarrow \int g(x, y) dP^{XY}(x, y), \tag{3.42}$$

since P_n^{XY} converge to P^{XY} weakly.

We then need to show $\int (g_n(x, y) - g(x, y))dP_n^{XY}(x, y)$ converges to 0. We prove by contradiction. Suppose it does not converge, then there exists $\epsilon > 0$ and a subsequence of n (denote as i) such that

$$\left| \int (g_i(x, y) - g(x, y))dP_i^{XY}(x, y) \right| \geq \epsilon, \forall i. \quad (3.43)$$

Without loss of generality, we assume that

$$\int (g_i(x, y) - g(x, y))dP_i^{XY}(x, y) \geq \epsilon > 0, \forall i, \quad (3.44)$$

i.e.

$$\begin{aligned} & \iiint f(x, y, z)dP_i^{Z|Y=y}(z)dP_i^{X|Y=y}(x)dP_i^Y(y) \\ & - \iiint f(x, y, z)dP^{Z|Y=y}(z)dP^{X|Y=y}(x)dP^Y(y) \geq \epsilon > 0, \forall i. \end{aligned} \quad (3.45)$$

Let

$$h_i(y, z) = \int f(x, y, z)dP_i^{X|Y=y}(x), \quad (3.46)$$

then

$$\begin{aligned} & \iint h_i(y, z)dP_i^{Z|Y=y}(z)dP_i^Y(y) \\ & - \iint h_i(y, z)dP^{Z|Y=y}(z)dP^Y(y) \geq \epsilon > 0, \forall i. \end{aligned} \quad (3.47)$$

Note that f is K -Lipschitz, $P_i^{X|Y=y}$ is K -Lipschitz, we have h_i are all $K(C + 1)$ -Lipschitz, thus uniformly equicontinuous. Also $h_i(y, z)$ are uniformly bounded by the bound of f , and the domain for (Y, Z) is compact (since the domain D_t for X_t is compact for all t). By the Arzela-Ascoli theorem, there exists a subsequence j such

that $h_j \rightarrow h$ uniformly, where h is $K(C + 1)$ -Lipschitz and bounded. Therefore,

$$\begin{aligned} & \iint (h_j(y, z) - h(y, z)) dP_j^{Z|Y=y}(z) dP_j^Y(y) \\ & - \iint (h_j(y, z) - h(y, z)) dP^{Z|Y=y}(z) dP^Y(y) \rightarrow 0. \end{aligned} \quad (3.48)$$

From Lemma 3

$$\begin{aligned} & \iint h(y, z) dP_j^{Z|Y=y}(z) dP_j^Y(y) \\ & - \iint h(y, z) dP^{Z|Y=y}(z) dP^Y(y) \rightarrow 0. \end{aligned} \quad (3.49)$$

We then have

$$\begin{aligned} & \iint h_j(y, z) dP_j^{Z|Y=y}(z) dP_j^Y(y) \\ & - \iint h_j(y, z) dP^{Z|Y=y}(z) dP_j^Y(y) \rightarrow 0. \end{aligned} \quad (3.50)$$

We have a contradiction between Equation 3.47 and 3.50. We finish the proof for $T = 3$.

We then prove the case for general T by induction. Suppose it holds for $T = N$, we now prove it for $T = N + 1$.

From the conditions for the case $T = N + 1$ and that the theorem holds for $T = N$, we have $P_n^{X_{1:N}}$ converges to $P^{X_{1:N}}$ weakly and $P_n^{X_{2:N+1}}$ converges to $P^{X_{2:N+1}}$ weakly. We now view X_1 as X , view the Cartesian product of $X_{2:N}$ as Y , and view X_{N+1} as Z , so we have P_n^{XY} converges to P^{XY} weakly and P_n^{YZ} converges to P^{YZ} weakly. Also $P_n^{X|Y}$ and $P^{Z|Y}$ are C -Lipschitz continuous and Wasserstein- p metric for all n , since $P_n^{X|Y} = P_n^{X_1|X_2}$ and $P^{Z|Y} = P^{X_{N+1}|X_N}$ from the Markovian property, and that $\|X_2\| \leq \|X_{2:N}\|$, $\|X_N\| \leq \|X_{2:N}\|$. Therefore, from the theorem for $T = 3$ case we have that P_n^{XYZ} converges to P^{XYZ} weakly, i.e., $P_n^{X_{1:N+1}}$ converges to $P^{X_{1:N+1}}$ weakly.

□

3.6.6 A Counterexample in Continuous Sample Space

As a direct implementation of Corollary 7 in [26], for the Markov chain $X_{1:T}$ in a finite discrete sample space, $P_n^{X_{t:t+1}}$ converges to $P^{X_{t:t+1}}$ in Wasserstein- p sense for each $t = 1, 2, \dots, T-1$ implies that $P_n^{X_{1:T}}$ converges to $P^{X_{1:T}}$ in Wasserstein- p sense. However, if the Markov chains are defined in the continuous sample space, the implementation is, in general, not correct without further assumptions, e.g., the assumption of continuity for probability transition kernels. In this section, we provide a counterexample to show that.

We consider the Markov chain with $T = 3$ and use X, Y, Z to denote the nodes in sequence. We define P_n^{XYZ} as follows:

$$\begin{aligned} P_n^{XYZ}(0, 0, 0) &= \frac{1}{2}, \\ P_n^{XYZ}(1, \frac{1}{n}, 1) &= \frac{1}{2}, \end{aligned} \tag{3.51}$$

and P^{XYZ} as following:

$$\begin{aligned} P^{XYZ}(0, 0, 0) &= \frac{1}{4}, \\ P^{XYZ}(0, 0, 1) &= \frac{1}{4}, \\ P^{XYZ}(1, 0, 0) &= \frac{1}{4}, \\ P^{XYZ}(1, 0, 1) &= \frac{1}{4}. \end{aligned} \tag{3.52}$$

We can easily check that P_n^{XY} converges to P^{XY} in Wasserstein- p metric, since

$$\begin{aligned} P_n^{XY}(0, 0) &= \frac{1}{2}, \\ P_n^{XY}(1, \frac{1}{n}) &= \frac{1}{2}, \end{aligned} \tag{3.53}$$

and

$$\begin{aligned} P^{XY}(0,0) &= \frac{1}{2}, \\ P^{XY}(1,0) &= \frac{1}{2}, \end{aligned} \tag{3.54}$$

Similarly P_n^{YZ} converges to P^{YZ} in Wasserstein- p metric. However, P_n^{XYZ} does not converge to P^{XYZ} in Wasserstein- p metric: the support of P_n^{XYZ} and $(0,0,1)$ always have a distance larger than 1.

3.7 Summary of Chapter

In this chapter, we introduce a new method for inferring the governing dynamics of particles from *unpaired observations* of their coordinates at multiple time instants, namely “snapshots”. We fitted the observed particle ensemble distribution with a physics-informed generative model, which can be viewed as performing regression in the probability measure space. We refer to this approach as generative “ensemble-regression”, in analogy to the classic “point-regression”, where we infer the dynamics by performing regression in the Euclidean space.

We first applied the method to particle systems governed by independent stochastic ordinary differential equations (SODE) with Brownian or Lévy noises, where we inferred the drift and the diffusion terms from a small number of snapshots. In the Lévy noise case, we demonstrated that the heavy tails in the distributions could spoil the training, but we addressed this issue by applying a preprocessing map to both the generated and target distributions. In scenarios with noisy or truncated training data, we modified the generated distributions accordingly by perturbing or filtering the generated samples. We then addressed high-dimensional SODE problems using the adversarial loss in GANs. We also

managed to learn the parameters for particle interactions in a nonlocal flocking systems. Finally, we investigate scenarios of paired observations and discuss how to reduce the dimensionality in such cases by proving a convergence theorem that provides theoretical support.

CHAPTER FOUR

Potential Flow Generator with L_2 Optimal Transport Regularity for GANs

4.1 Introduction

A big family of generative models, for example, generative adversarial networks (GANs) [37, 4, 116] and normalizing flow models [111, 55, 19], can be employed to find a generator that could map the input distribution to the target distribution. In general, given the input and target distributions, a plethora of such maps can be found to accomplish this task. In many cases, especially when the input distributions are purely noise, the specific maps between input and output are of little importance as long as the generated distributions match the target ones. However, in other cases like the image-to-image translation, where both input and target distributions are distributions of images, the generators are required to have additional regularity such that the input individuals are mapped to the “corresponding” outputs in some sense. If paired input-output samples are provided, L_p penalty could be hybridized into generators loss functions to encourage the output individuals to fit the ground truth `refisola2017image`. For the cases without paired data, a popular approach is to introduce another generator and encourage the two generators to be the inverse maps of each other, as in CycleGAN [156], DualGAN [145] and DiscoGAN [53], etc. However, such a pair of generators is not unique and lacks a clear mathematical interpretation of its effectiveness.

In this chapter we introduce a special generator, i.e., the potential flow generator, with L_2 optimal transport regularity. By applying such a generator, not only are we trying to find a map from the input distribution to the target one, but we also aim to find the **optimal transport map** that minimizes the total squared Euclidean transport distance. In Figure 4.1 we provide a schematic comparison between generators with and without optimal transport regularity. While both generators provide a scheme to map from the input distribution to the output dis-

tribution, the total squared transport distance in the left generator is larger than that in the right generator. Note that the generator with optimal transport regularity has the characteristic of “proximity” in that the inputs tend to be mapped to “nearby” outputs. As we will show later, this “proximity” characteristic of L_2 optimal transport regularity can be utilized in image translation tasks. Compared with other approaches like CycleGAN, the L_2 optimal transport regularity has a much clearer mathematical interpretation.

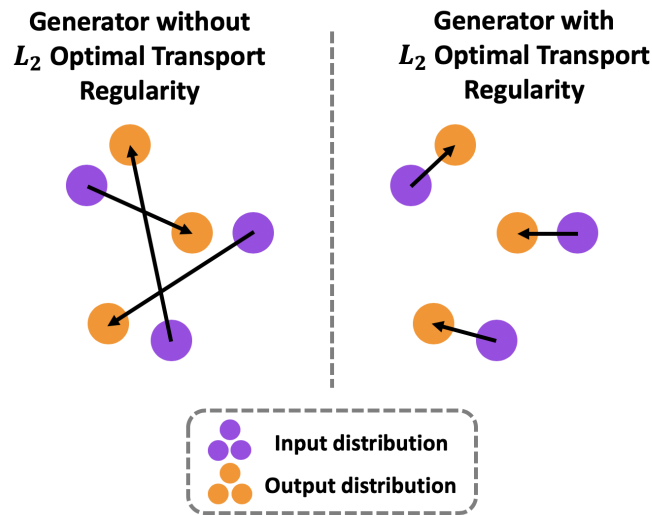


Figure 4.1: Schematic of generator without and with L_2 optimal transport regularity.

We want to clarify that the L_2 optimal transport regularity here is different from the L_2 regularization commonly used in machine learning; that is, we do not penalize the norm of parameters, and the L_2 optimal transport regularity is not related to overfitting. It also serves different objectives compared with the optimal transport theory in the well-known Wasserstein GANs. On the one hand, the optimal transport theory used in Wasserstein GANs is used to provide a metric (Wasserstein distance) between the output distribution and the target one, but it does not help to regularize the generator map itself. On the other hand, the L_2 optimal transport regularity is only for the generator map itself, and is not related to the metric between the output distribution and the target one.

There have been other approaches to learn the optimal transport map or its variants in generative models. For example, [122] proposed to first learn the regularized optimal transport plan and then the optimal transport map, based on the dual form of regularized optimal transport problem. Also, [137] proposed to learn the unbalanced optimal transport plan in an adversarial way derived from a convex conjugate representation of divergences. In the W2GAN model proposed by [65], the discriminator’s objective is the 2-Wasserstein metric so that the generator is supposed to recover the L_2 optimal transport map. In [122] and [137] the authors also take advantage of the characteristic of “proximity” for optimal transport in tasks like image translation, domain adaption, etc.

All the above approaches need to introduce, and are limited to, specific loss functions to train the generators. Our proposed potential flow generator takes a different approach in that with only a slight augmentation to the original generator loss functions, our generator can be integrated into a wide range of generative models with various generator loss functions, including different versions of GANs and normalizing flow models. This simple modification makes our method easy to adopt on various tasks considering the existing rich literature and the future developments of generative models.

In Section 4.2 we present a formal definition of optimal transport map and the motivation to apply L_2 optimal transport regularity to generators. In Section 4.3 we give a detailed formulation of the potential flow generator and the augmentation to the original loss functions. Results are then provided in Section 4.4. We include the discussion and conclusions in Section 4.5.

4.2 Generative Models and Optimal Transport Map

First, we introduce the concept of push-forward, which will be used extensively in the chapter.

Definition 1. *Given two Polish spaces $\mathbb{X}$ and $\mathbb{Y}$, $\mathcal{B}(\mathbb{X})$ and $\mathcal{B}(\mathbb{Y})$ the Borel σ -algebra on $\mathbb{X}$ and $\mathbb{Y}$, and $\mathcal{P}(\mathbb{X}), \mathcal{P}(\mathbb{Y})$ the set of probability measures on $\mathcal{B}(\mathbb{X})$ and $\mathcal{B}(\mathbb{Y})$. Let $f : \mathbb{X} \rightarrow \mathbb{Y}$ be a Borel map, and $\mu \in \mathcal{P}(\mathbb{X})$. We define $f_{\#}\mu \in \mathcal{P}(\mathbb{Y})$, the push-forward of μ through f , by*

$$f_{\#}\mu(\mathbb{A}) = \mu(f^{-1}(\mathbb{A})), \forall \mathbb{A} \in \mathcal{B}(\mathbb{Y}). \quad (4.1)$$

With the concept of push-forward, we can formulate the goal of GANs and normalizing flow models as to train the generator G such that $G_{\#}\mu$ is equal to or at least close to ν in some sense, where μ and ν are the input and target distributions, respectively. Usually, the loss functions for training the generators are metrics of closeness that vary for different models. For example, in continuous normalizing flows [19], such metric of closeness is $D_{\text{KL}}(G_{\#}\mu||\nu)$ or $D_{\text{KL}}(\nu||G_{\#}\mu)$. In Wasserstein GANs (WGANs) [4, 43], the metric of closeness is the Wasserstein-1 distance between $G_{\#}\mu$ and ν , which is estimated in a variational form with the discriminator neural network. More specifically, in GANs we define a two-player zero-sum game between the generator $G(\cdot)$ and the discriminator $D(\cdot)$. The discriminator takes a sample as input and aims to find if it is generated by the generator, i.e., sampled from $G_{\#}\mu$, or from a real dataset, i.e. sampled from ν . Meanwhile, the generator tries to “deceive” the discriminator by approximating the target distribution ν . The two-player zero-sum game in WGANs is defined as follows:

$$\min_G \max_{D \text{ is 1-Lipschitz}} \mathbb{E}_{y \sim \nu}[D(y)] - \mathbb{E}_{x \sim \mu}[D(G(x))], \quad (4.2)$$

where $\mathbb{E}_{y \sim \nu}[\cdot]$ represents the expectation with $\mathbf{y}$ sampled from ν . The Lipschitz constraint for D could be imposed via the gradient penalty [43], spectral normalization [86], etc. For example, in WGANs with gradient penalty (WGAN-GP), the loss functions for the generator and discriminator are defined as

$$\begin{aligned} L_g &= -\mathbb{E}_{x \sim \mu}[D(G(x))], \\ L_d &= \mathbb{E}_{x \sim \mu}[D(G(x))] - \mathbb{E}_{y \sim \nu}[D(y)] + \\ &\quad \eta \mathbb{E}_{\hat{x} \sim P_{\hat{x}}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2], \end{aligned} \tag{4.3}$$

respectively, where $P_{\hat{x}}$ is the distribution generated by uniform sampling on interpolation lines between pairs of points sampled from ν and $G_{\#}\mu$, and η is the gradient penalty coefficient.

Now we introduce the concept of optimal transport map as follows:

Definition 2. Given a cost function $c : \mathbb{X} \times \mathbb{Y} \rightarrow \mathbb{R}$, and $\mu \in \mathcal{P}(\mathbb{X})$, $\nu \in \mathcal{P}(\mathbb{Y})$, we let $\mathbb{T}$ be the set of all transport maps from μ to ν , i.e., $\mathbb{T} := \{f : f_{\#}\mu = \nu\}$. Monge's optimal transport problem is to minimize the cost functional $C(f)$ among $\mathbb{T}$, where

$$C(f) = \mathbb{E}_{x \sim \mu} c(x, f(x)) \tag{4.4}$$

and the minimizer $f^* \in \mathbb{T}$ is called the optimal transport map.

In this chapter, we are concerned mostly with the case where $\mathbb{X} = \mathbb{Y} = \mathbb{R}^d$ with L_2 transport cost, i.e., the transport $c(x, \mathbf{y}) = \|x - \mathbf{y}\|^2$. We assume that μ and ν are absolute continuous w.r.t. Lebesgue measure, i.e. they have probability density functions. In general, Monge's problem could be ill-posed in that $\mathbb{T}$ could be empty set or there is no minimizer in $\mathbb{T}$. Also, the optimal transport map could be non-unique. However, for the special case we consider, there exists a unique

solution to Monge's problem [15, 36].

Informally speaking, with L_2 transport cost the optimal transport map has the characteristic of "proximity", i.e. the inputs tend to be mapped to nearby outputs. In image translation tasks, such "proximity" characteristic would be helpful if we could properly embed the images into Euclidean space such that our preferred input-output pairs are close to each other. A similar idea is also proposed in [137] for unbalanced optimal transport. Apart from image translations, the L_2 optimal transport problem is important in many other aspects. For example, it is closely related to gradient flow [2], Fokker-Planck equations [117], flow in porous media [93], etc.

4.3 Methodology

4.3.1 Potential Flow Formulation of Optimal Transport Map

We assume that μ and ν have probability density ρ_μ and ρ_ν , respectively, and consider all smooth enough density fields $\rho(t, \mathbf{x})$ and velocity fields $\mathbf{v}(t, \mathbf{x})$, where $t \in [0, T]$, subject to the continuity equation as well as initial and final conditions

$$\begin{aligned} \partial_t \rho + \nabla \cdot (\rho \mathbf{v}) &= 0, \\ \rho(0, \cdot) &= \rho_\mu, \quad \rho(T, \cdot) = \rho_\nu. \end{aligned} \tag{4.5}$$

The above equation states that such velocity field will induce a transport map: we can construct an ordinary differential equation (ODE)

$$\frac{d\mathbf{u}}{dt} = \mathbf{v}(t, \mathbf{u}), \quad (4.6)$$

and the map from the initial point to the final point gives the transport map from μ to ν . The time dependent density fields $\rho(t, \mathbf{x})$ is then induced from $\mathbf{v}$.

In general the solution of $(\rho, \mathbf{v})$ in Equation 4.5 is not unique, since the transport map from μ to ν is not unique. As is proposed by [9], for the transport cost function $c(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|^2$, the minimal transport cost is equal to the infimum of

$$T \int_{\mathbb{R}^d} \int_0^T \rho(t, \mathbf{x}) |\mathbf{v}(t, \mathbf{x})|^2 d\mathbf{x} dt \quad (4.7)$$

among all $(\rho, \mathbf{v})$ satisfying equation (4.5). The optimality condition for $\mathbf{v}$ is given by

$$\mathbf{v}(t, \mathbf{x}) = \nabla \phi(t, \mathbf{x}) \quad (4.8)$$

$$\partial_t \phi + \frac{1}{2} |\nabla \phi|^2 = 0. \quad (4.9)$$

In other words, the optimal velocity field is actually induced from a flow with time-dependent potential $\phi(t, \mathbf{x})$, which satisfies the Hamilton-Jacobi(HJ) equation 4.9. In this chapter we integrate this formulation in the deep generative models. Instead of solving Monge's problem and find the exact L_2 optimal transport map, which is unrealistic due to the errors arising from training the neural networks, our goal is to regularize the generators in a wide range of generative models, so that the generator maps could approximate the L_2 optimal transport map at least in low dimensional problems. The maps would also be endowed with the

characteristics of “proximity” so that we can apply them and design interpretable learning machines for engineering problems.

4.3.2 Potential Flow Generator

The potential $\phi(t, \mathbf{x})$ is the key function to estimate, since the velocity field could be obtained by taking the gradient of the potential and consequently the transport map could be obtained from Equation 4.6. There are two strategies to use neural networks to represent ϕ . One can take advantage of the fact that the time-dependent potential field ϕ is actually uniquely determined by its initial condition from Equation 4.9, and use a neural network to represent the initial condition of ϕ , i.e. $\phi(0, \mathbf{x})$, while approximating $\phi(t, \mathbf{x})$ via time discretization schemes. Alternatively, one can use a neural network to represent $\phi(t, \mathbf{x})$ directly and later apply the PDE regularity for $\phi(t, \mathbf{x})$ in Equation 4.9. We name the generators defined in the above two approaches as **discrete** potential flow generator and **continuous** potential flow generator, respectively, and give a detailed formulation as follows.

Discrete Potential Flow Generator

In the discrete potential flow generator, we use the neural network $\tilde{\phi}_0(\mathbf{x}) : \mathbb{R}^d \rightarrow \mathbb{R}$ to represent the initial condition of $\phi(t, \mathbf{x})$, i.e. $\phi(0, \mathbf{x})$. The potential field $\phi(t, \mathbf{x})$ as well as the velocity field $\mathbf{v}(t, \mathbf{x})$ could then be approximated by different time discretization schemes. As an example, here we use the first-order forward Euler scheme for the simplicity of implementation. To be specific, suppose the time discretization step is Δt and the number of total steps is n with $n\Delta t = T$, then for

$i = 0, 1 \dots n$, $\phi(i\Delta t, \mathbf{x})$ could be represented by $\tilde{\phi}_i(\mathbf{x})$, where

$$\tilde{\phi}_{i+1}(\mathbf{x}) = \tilde{\phi}_i(\mathbf{x}) - \frac{\Delta t}{2} |\nabla \tilde{\phi}_i(\mathbf{x})|^2, \quad \text{for } i = 0, 1, 2 \dots, n-1. \quad (4.10)$$

Consequently, the velocity field $\mathbf{v}(i\Delta t, \mathbf{x})$ could be represented by $\tilde{\mathbf{v}}_i(\mathbf{x})$, where

$$\tilde{\mathbf{v}}_i(\mathbf{x}) = \nabla \tilde{\phi}_i(\mathbf{x}), \quad \text{for } i = 0, 1 \dots n. \quad (4.11)$$

Finally, we can build the transport map from Equation 4.6:

$$\begin{aligned} \tilde{\mathbf{u}}_0(\mathbf{x}) &= \mathbf{x}, \\ \tilde{\mathbf{u}}_{i+1}(\mathbf{x}) &= \tilde{\mathbf{u}}_i(\mathbf{x}) + \Delta t \tilde{\mathbf{v}}_i(\tilde{\mathbf{u}}_i(\mathbf{x})), \text{ for } i = 0, 1, 2 \dots n-1, \end{aligned} \quad (4.12)$$

with $G(\cdot) = \tilde{\mathbf{u}}_n(\cdot)$ be our transport map.

The discrete potential flow generator has built-in optimal transport regularity since the optimal condition (Equation 4.9) is encoded in the time discretization (Equation 4.10). However, such discretization also introduces nested gradients, which dramatically increases the computational cost when the number of total steps n is increased. In our tests, we found that even $n = 5$ is almost intractable.

Continuous Potential Flow Generator

In the continuous potential flow generator, we use the neural network $\tilde{\phi}(t, \mathbf{x}) : \mathbb{R} \times \mathbb{R}^d \rightarrow \mathbb{R}$ to represent $\phi(t, \mathbf{x})$. Consequently, the velocity field $\mathbf{v}(t, \mathbf{x})$ can be represented by $\tilde{\mathbf{v}}(t, \mathbf{x})$, where

$$\tilde{\mathbf{v}}(t, \mathbf{x}) = \nabla \tilde{\phi}(t, \mathbf{x}). \quad (4.13)$$

With the velocity field we could estimate the transport map by solving the ODE (Equation 4.6) using any numerical ODE solver. As an example, we can use the first-order forward Euler scheme, i.e.

$$\begin{aligned}\tilde{\mathbf{u}}(0, \mathbf{x}) &= \mathbf{x}, \\ \tilde{\mathbf{u}}((i+1)\Delta t, \mathbf{x}) &= \tilde{\mathbf{u}}(i\Delta t, \mathbf{x}) + \Delta t \tilde{\mathbf{v}}(i\Delta t, \tilde{\mathbf{u}}(i\Delta t, \mathbf{x})), \\ &\text{for } i = 0, 1, 2 \dots n-1,\end{aligned}\tag{4.14}$$

with

$$G(\cdot) = \tilde{\mathbf{u}}(T, \cdot)\tag{4.15}$$

be the transport map, i.e. the generator in generative models, where Δt is the time discretization step and n is the number of total steps with $n\Delta t = T$.

In the continuous potential flow generator, increasing the number of total steps will not introduce high order differentiation, therefore we can have large n for better accuracy of the ODE solver. Different from the discrete potential flow generator, the optimal condition (Equation 4.9) is not encoded in the continuous potential flow generator, therefore we need to penalize Equation 4.9 in the loss function, as we will discuss in the next subsection.

One may come up with another strategy of imposing the L_2 optimal transport regularity: to use a vanilla generator, which is a neural network directly mapping from inputs to outputs, and penalize the L_2 transport cost, i.e., the loss function is

$$L_{\text{vanilla}} = L_{\text{original}} + \alpha \mathbb{E}_{\mathbf{x} \sim \mu} \|\mathbf{G}(\mathbf{x}) - \mathbf{x}\|^2,\tag{4.16}$$

where L_{original} is the original loss function for the generator, and α is the weight for the transport penalty. We emphasize that such a strategy is much inferior to

penalizing Equation 4.9 in the continuous potential flow generator. When training the vanilla generator with L_2 transport penalty, no matter how we weight the L_2 transport cost penalty, we always have to make a trade-off between “matching the generated distribution with the target one” and “reducing the transport cost” since there is always a conflict between them, and consequently $G_{\#}\mu$ will be biased towards μ . On the other hand, there is no conflict between matching the distributions and penalizing Equation 4.9 in the continuous potential flow generator. As a consequence, the continuous potential flow generator is robust with respect to different weights for the PDE penalty. We will show this in Section 4.4.

4.3.3 Training the Potential Flow Generator

While the optimal condition (Equation 4.9) has been considered in the above two generators, the constraints of initial and final conditions have so far been neglected. However, the constraint of initial and final conditions provides the principle to train the neural network: we need to tune the parameter in the neural network $\tilde{\phi}$ so that $G_{\#}\mu$ matches ν . This could be done using either GANs or normalizing flow models.

We note that the potential flow generator can be integrated into various generative models; we only need to replace the generator in the original model with the discrete or continuous potential flow generator we introduced above, and train the potential flow generator with the loss function defined as follows. For the discrete one, the loss function is identical to the original generator loss. For the continuous one, as we will see in Equation 4.18 and 4.21, the loss function is the original generator loss augmented with an additional penalty for PDE residual. We do not need to change the other parts of the generative models, e.g., the loss

functions for the discriminators in GANs, the training processes, etc.

Loss in GAN Models

For the discrete potential flow generator, since the optimal transport regularity is already built-in, the loss for training G is simply the original GAN loss for the generator, i.e.

$$L_{\text{D-PFG}} = L_{\text{original}}, \quad (4.17)$$

where L_{original} actually depends on the specific version of GANs. For example, if we use WGAN, then $L_{\text{original}} = -\mathbb{E}_{x \sim \mu}[D(G(x))]$, where D is the discriminator neural network.

For the continuous potential flow generator, as mentioned above, we also need to make $\tilde{\phi}$ satisfy Equation 4.9 for the optimal transport regularity. Inspired by the applications of neural networks in solving forward and backward problems of PDEs [58, 110, 124], we penalize the squared residual of the PDE on the so-called “residual” points, i.e., the random points in the domain where we evaluate the residual of the PDE. In particular, the loss for continuous potential flow generator would be

$$L_{\text{C-PFG}} = L_{\text{original}} + \lambda \mathbb{E}_{(t,x) \sim \mathcal{B}} (\partial_t \tilde{\phi}(t, x) + \frac{1}{2} |\nabla \tilde{\phi}(t, x)|^2)^2, \quad (4.18)$$

where λ is the weight for PDE residual penalty, $\mathcal{B}$ is the distribution for the residual points (t, x) .

Theoretically, Equation 4.9 should be satisfied in the entire spatial-temporal domain. Note that we actually only care about ϕ in the support of ρ in Equation 4.5, which determines the velocity and thus the transport map for the input samples.

Therefore, we sample the residual points in the following way: we first collect the points on the “trajectories” of input samples, i.e.,

$$\mathbb{B} = \bigcup_{i=0}^n \bigcup_{x_j \in I} \{(i\Delta t, \tilde{\mathbf{u}}(i\Delta t, \mathbf{x}_j))\}, \quad (4.19)$$

where I is the set of samples from μ in batch. For each point in $\mathbb{B}$, we then perturb the temporal coordinate t with uniform noise $\mathcal{U}(-\Delta t/2, \Delta t/2)$ (except for the points with $t = 0$ or $t = T$, where the uniform noise is replaced by $\mathcal{U}(0, \Delta t/2)$ and $\mathcal{U}(-\Delta t/2, 0)$, respectively), and perturb the spatial coordinate $\mathbf{x}$ with Gaussian noise $\mathcal{N}(0, \mathbf{I})$, where $\mathbf{I}$ is the identity matrix. In this way, in principle we make the support of $\mathcal{B}$ cover the entire spatial-temporal domain, while also adaptively putting more weight on the support of ρ .

As an example of applying the continuous potential flow generator, here we give a detailed algorithm for WGAN-GP with a continuous potential flow generator in Algorithm 3. We use θ_d to represent the parameters in the discriminator neural network $D(\cdot)$, while we use θ_g to represent the parameters in the potential neural network $\tilde{\phi}$ as well as the generator $G(\cdot)$ defined by Equation 4.13, 4.14 and 4.15. Compared with the vanilla WGAN-GP, we only replace the generator, and in line 19 augment the original generator loss L_{original} with the PDE residual penalty, for which we need to sample residual points from line 14 to line 18, to get the new generator loss L_g .

Loss in Normalizing Flow Models

Our continuous potential flow generator could be viewed as a further development of Neural ODEs [19] applied to generative models, i.e. the continuous normalizing

Algorithm 3 WGAN-GP with continuous potential flow generator.

Input: The PDE residual penalty coefficient λ , the gradient penalty coefficient η , the number of discriminator iterations per generator iteration n_d , the batch size m , Adam optimizer hyperparameters.

- 1: *Initialisation:* discriminator parameters θ_d , generator parameters θ_g .
- 2: **while** θ_g has not converged **do**
- 3: **for** $s = 1, 2, \dots, n_d$ **do**
- 4: Sample m data $\{x_j\}_{j=1}^m$ from the training dataset of the input distribution μ .
- 5: Sample m data $\{y_j\}_{j=1}^m$ from the training dataset of the target distribution ν .
- 6: Sample m random numbers $\{\epsilon_j\}_{j=1}^m$ uniformly from 0 to 1.
- 7: **for** $j = 1, 2, \dots, m$ **do**
- 8: $\hat{y}_j \leftarrow \epsilon_j y_j + (1 - \epsilon_j) G(x_j)$
- 9: **end for**
- 10: $L_d \leftarrow \frac{1}{m} \sum_{j=1}^m [D(G(x_j)) - D(y_j) +$
- 11: $\eta(\|\nabla_{\hat{y}_j} D(\hat{y}_j)\|_2 - 1)^2]$
- 12: $\theta_d \leftarrow \text{Adam}(\nabla_{\theta_d} L_d, \theta_d)$
- 13: **end for**
- 14: Sample m data $\{x_j\}_{j=1}^m$ from the training dataset of the input distribution μ .
- 15: $L_{\text{original}} \leftarrow -\frac{1}{m} \sum_{j=1}^m D(G(x_j))$
- 16: **for** $i = 0, 1, \dots, n$ **do**
- 17: **for** $j = 1, 2, \dots, m$ **do**
- 18: $(t'_{i,j}, \mathbf{u}'_{i,j}) \leftarrow (i\Delta t, \tilde{\mathbf{u}}(i\Delta t, x_j)) + \xi$, where ξ is the perturbation noise described after Equation 4.19.
- 19: **end for**
- 20: **end for**
- 21: $L_g \leftarrow L_{\text{original}} + \frac{\lambda}{(n+1)m} \sum_{i=0}^n \sum_{j=1}^m (\partial_t \tilde{\phi}(t'_{i,j}, \mathbf{u}'_{i,j}) + \frac{1}{2} |\nabla \tilde{\phi}(t'_{i,j}, \mathbf{u}'_{i,j})|^2)^2$
- 22: $\theta_g \leftarrow \text{Adam}(\nabla_{\theta_g} L_g, \theta_g)$
- 23: **end while**

flow. The difference mainly lies in that we set the velocity as the gradient of a time-dependent potential function, and an augmented PDE loss is required for the generator. We refer the readers to [152] and [114] for more relevant works on Neural ODEs and optimal transport. Just as in Neural ODEs, for the continuous potential flow generator G , we can estimate the inverse map $\mathbf{x} = G^{-1}(\mathbf{y})$ by solving the ODE

$$\frac{d\mathbf{w}(t)}{dt} = -\tilde{v}(T-t, \mathbf{w}(t)) = -\nabla\tilde{\phi}(T-t, \mathbf{w}(t)) \quad (4.20)$$

with initial condition $\mathbf{w}(0)$ as $\mathbf{y}$ and $\mathbf{w}(T)$ as the corresponding $\mathbf{x} = G^{-1}(\mathbf{y})$.

While both density matching and maximum likelihood training could be applied, here we take the latter as an example: we assume that the density of μ and samples from ν are available, then in the vanilla normalizing flow model, we will maximize $\mathbb{E}_{\mathbf{y} \sim \nu}[\log p_{G_{\#}\mu}(\mathbf{y})]$, where $p_{G_{\#}\mu}$ is the density of $G_{\#}\mu$. Note that this is equivalent to minimizing $D_{\text{KL}}(\nu \| G_{\#}\mu)$. Consequently, the augmented loss for the continuous potential flow generator is:

$$L_{\text{C-PFG}} = -\mathbb{E}_{\mathbf{y} \sim \nu}[\log p_{G_{\#}\mu}(\mathbf{y})] + \lambda \mathbb{E}_{(t, \mathbf{x}) \sim \mathcal{B}}(\partial_t \tilde{\phi}(t, \mathbf{x}) + \frac{1}{2} |\nabla \tilde{\phi}(t, \mathbf{x})|^2)^2, \quad (4.21)$$

where as in the GAN model, $\mathcal{B}$ is the distribution for the residual points $(t, \mathbf{x})$ for estimating the residual of PDE (Equation 4.9), and λ is the weight for the PDE penalty.

To estimate $\mathbb{E}_{\mathbf{y} \sim \nu}[\log p_{G_{\#}\mu}(\mathbf{y})]$, we have the ODE that connects the probability density at inputs and outputs of the generator:

$$\begin{aligned} \frac{d}{dt} \log(p(\tilde{\mathbf{u}}(t, \mathbf{x}))) &= -\nabla \cdot \tilde{v}(t, \tilde{\mathbf{u}}(t, \mathbf{x})) \\ &= -\Delta \tilde{\phi}(t, \tilde{\mathbf{u}}(t, \mathbf{x})), \end{aligned} \quad (4.22)$$

where the density $p(\tilde{\mathbf{u}}(0, \mathbf{x})) = p_\mu(\mathbf{x})$ is the density of μ at input $\mathbf{x}$, while $p(\tilde{\mathbf{u}}(T, \mathbf{x})) = p_{G_\# \mu}(G(\mathbf{x}))$ is the density of $G_\# \mu$ at output $G(\mathbf{x})$. In practice, for each $\mathbf{y}$, we solve the following equation system induced from Equation 4.20 and 4.22:

$$\begin{aligned} \frac{d\mathbf{w}(t)}{dt} &= -\nabla \tilde{\phi}(T - t, \mathbf{w}(t)) \\ \frac{dq(t)}{dt} &= \Delta \tilde{\phi}(T - t, \mathbf{w}(t)), \end{aligned} \tag{4.23}$$

with initial condition $\mathbf{w}(0) = \mathbf{y}$ and $q(0) = 0$. We can then calculate $\log p_{G_\# \mu}(\mathbf{y}) = \log p_\mu(\mathbf{w}(T)) - q(T)$, and then use samples from ν in training data to estimate $\mathbb{E}_{\mathbf{y} \sim \nu}[\log p_{G_\# \mu}(\mathbf{y})]$. In Section 4.4.1 we will use the first-order Euler scheme and second-order midpoint scheme to solve the system of equations 4.23, and we can see both work well.

Note that when applying maximum likelihood training, the density of μ could be unnormalized, since multiplications with p_μ would merely lead to a constant difference in the loss function. Finally, we remark that the discrete potential flow cannot be trivially applied in normalizing flow models since we found that the time step size is too large to calculate the density accurately.

4.4 Computational Results

Except in the regularized discrete optimal transport solver and official CycleGAN, all the neural networks are feedforward neural networks of 5 hidden layers, each of width 128 for 2D problems, or 256 in image translation tasks. For potential flow generator, the activation function is tanh for the smoothness of $\tilde{\phi}$. The activation function in vanilla generator in Figure 4.2 is also tanh. All the other activation

functions are leaky ReLU [77] with negative slope 0.2, except in the regularized discrete optimal transport solver and official CycleGAN.

The batch size is set as 1000 for all the cases except in Table 4.2 and in official CycleGAN. We use 1000 random projection directions to estimate the sliced Wasserstein distances. In the WGAN-GP model the coefficient for gradient penalty is 0.1, and we do 5 discriminator updates per generator update.

In potential flow generators, the time span T is 1.0. We set the number of total time steps $n = 4$ in discrete potential flow generators, while $n = 100$ in continuous potential flow generators for 2D problems and $n = 10$ in image translation tasks. The PDE penalty weight λ for continuous potential flow generator is set as 1.0 by default, except those in the 2D problems where we compare different generators.

We use the Adam optimizer [54] for all the problems. In the 2D problems, the learning rate is set as $l = 10^{-5}$, $\beta_1 = 0.5$, $\beta_2 = 0.999$ in sliced Wasserstein generator, while set as $l = 10^{-5}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$ in WGAN-GP, and we train the generators for 100,000 iterations in both models. In the normalizing flow model we set $l = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and train the generator for 10,000 iterations. In image translation tasks we set $l = 10^{-4}$, $\beta_1 = 0.5$, $\beta_2 = 0.9$, and train the generator for 100,000 iterations for our method and CycleGANs with FNN, while 200,000 iterations for vanilla GAN.

4.4.1 2D Problems

In this subsection, we apply the potential flow generators to several two-dimensional problems. We mainly study the following two problems where we know analyt-

ical solutions for the optimal transport maps. In problem 1 we assume that both μ and ν are Gaussian distributions with $\mu = \mathcal{N}([0;0], [0.25, 0; 0, 1])$ and $\nu = \mathcal{N}([0;0], [1, 0; 0, 0.25])$. In this case the optimal transport map is $f((x, y)) = (2x, 0.5y)$. In problem 2 we assume that μ and ν are concentrated on concentric rings. In polar coordinates, suppose μ has (r, θ) uniformly distributed on $[0.5, 1] \times [0, 2\pi)$, while ν has (r, θ) uniformly distributed on $[2, 2.5] \times [0, 2\pi)$, where r and θ are radius and angle, respectively. In this case the optimal transport map is $f((r, \theta)) = (r + 1.5, \theta)$ in polar coordinates. Samples from μ and ν as well as the optimal transport map in both problems are illustrated in Figure 4.2. We prepared 40,000 samples for the input distribution and target distribution in each problem as training data for Section 4.4.1 and 4.4.1, and 1000 samples for Section 4.4.1.

Vanilla Generator Versus Potential Flow Generator

For the above two problems we compare the following methods: (a) vanilla generator with L_2 transport cost penalty, i.e., using the loss function in Equation 4.16 with GAN loss as L_{original} , (b) discrete potential flow generator, and (c) continuous potential flow generator with PDE penalty. For the vanilla generator and the continuous potential flow generator, we test different weights for the penalty in order to compare the influence of penalty weights in both generators. As for the GAN loss for generators, we use the sliced Wasserstein distance (SWG), due to its relatively low computational cost, robustness, and clear mathematical interpretation in low dimensional problems [25].

In Figure 4.2 we illustrate the maps of different generators. A more systematic and quantitative comparison for each case is provided in Table 4.1, where we show the mean and standard deviation of the statistics from three independent

runs. The best results are marked as bold. We also provide the statistics of 40,000 samples from the target distribution as reference for the errors induced from the limited samples in training data.

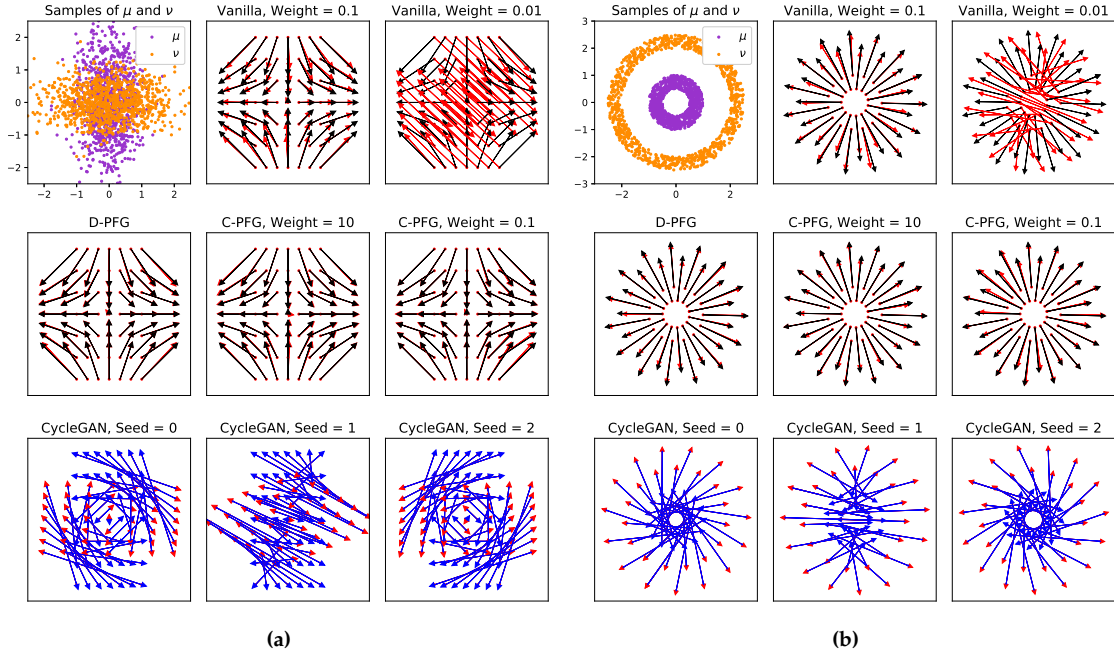


Figure 4.2: Comparison of different methods: vanilla generator with L_2 transport cost penalty, discrete potential flow generator (D-PFG), continuous potential flow generator (C-PFG), and CycleGAN in (a) problem 1 and (b) problem 2. The red and blue arrows represent the map of generators, the black arrows represent the analytical optimal transport map.

Table 4.1: Comparison between different generators on two problems

	Problem 1			Problem 2	
	Std in x-axis	Std in y-axis	Error of map	Mean of norm	Error of map
Exact	1.000	0.500		2.250	
40,000 samples	1.000±0.004	0.500±0.002		2.250±0.001	
Vanilla ($\alpha=0.1$)	0.919±0.004	0.592±0.003	0.108±0.002	2.107±0.002	0.146±0.003
Vanilla ($\alpha=0.01$)	0.985±0.005	0.499±0.006	0.439±0.587	2.227±0.004	0.973±1.319
Vanilla ($\alpha=0.001$)	0.992±0.009	0.493±0.001	0.462±0.593	2.243±0.002	1.000±1.311
D-PFG	0.993±0.001	0.498±0.002	0.018±0.006	Fail sometimes	
C-PFG ($\lambda=10.0$)	0.997±0.007	0.502±0.001	0.018±0.005	2.241±0.002	0.034±0.002
C-PFG ($\lambda=1.0$)	0.994±0.002	0.502±0.002	0.018±0.005	2.244±0.001	0.045±0.003
C-PFG ($\lambda=0.1$)	0.994±0.002	0.500±0.003	0.017±0.001	2.243±0.001	0.049±0.011

As we already mentioned, vanilla generators with L_2 transport penalty would make $G_{\#}\mu$ biased towards μ to reduce the transport cost from μ to $G_{\#}\mu$. This is

clearly shown in both problems with the penalty weight $\alpha = 0.1$. In fact, we observed more significant biases with larger penalty weights. For the cases with smaller penalty weights $\alpha = 0.01, 0.001$, in some of the runs, while $G_{\#}\mu$ are close to ν , the maps of generators are far from the optimal ones, which shows that the L_2 transport penalty cannot provide sufficient regularity if the penalty weight is too small. These numerical results are consistent with our earlier discussion about the intrinsic limitation of the L_2 transport penalty.

On the other hand, the potential flow generators give better matching between $G_{\#}\mu$ and ν , as well as smaller errors between the estimated transport maps and the analytical optimal transport maps. Notably, in both problems, the continuous potential flow generators give good results with a wide range of PDE penalty weights ranging from 0.1 to 10, which shows the superiority of the PDE penalty in the continuous potential flow generators compared with the transport penalty in vanilla generators. We also report that while in the first problem the discrete potential flow generator achieves a comparable result with the continuous potential flow generators, in the second problem we encountered “NAN” problems during the training of the discrete potential flow generator in some of the runs. This indicates that the discrete potential flow generator is not as robust as the continuous one, which could be attributed to the high order differentiation and small total time steps n in the discrete potential flow generators.

CycleGAN Versus Potential Flow Generator

We also apply CycleGAN on the above two problems, with different random seeds. Here, we use feedforward networks as generators and discriminators, with WGAN-GP [43] for the GAN loss function, and L_1 loss for cycle-consistency

loss with weight 5. In Figure 4.2 we illustrate the maps of G (red arrows) and F (blue arrows), i.e. the two generators in CycleGAN. The red and blue arrows overlap with opposite directions, which indicates that G and F are approximately the inverse map of each other, as we expected from the cycle-consistency loss. However, the maps are totally different in the three runs with different random seeds, which agrees with our discussion in Section 4.1 that the generator pair in CycleGAN is not unique. Moreover, the generator maps are less “regular” than the maps from the potential flow generator. Specifically, we can hardly interpret the generator maps given by CycleGAN.

Discrete Regularized Optimal Transport Solver Versus Potential Flow Generator

Finally, we compare the continuous potential flow generator in SWG and WGAN-GP with the discrete regularized optimal transport (RegOT) solver¹ introduced by [122]. We still use the two problems as in Section 4.4.1 and 4.4.1, i.e., the transport map between two Gaussian distributions and between concentric rings, except that we decrease the size of the training datasets to 1000 for all the methods compared here, which is of the same magnitude as the original training dataset in the test code of the RegOT solver. All the setups of their method are kept as default, except for the training data and batch size, which would be specified later. The results of the output distributions as well as the errors between estimated transport maps and the analytical optimal transport maps for each case are shown in Table 4.2, where we show the mean and standard deviation of the statistics from three independent runs. The best results are marked as bold. We also provide the

¹We used the code from <https://github.com/vivienseguy/Large-Scale-OT>, with the author’s permission. Their approach consists of two steps: (1) computing a regularized optimal transport plan, and (2) learning an optimal map by approximating the barycentric projection of the obtained optimal transport plan.

statistics of 1000 samples from the target distribution as reference for the errors induced from the limited samples in training data.

Table 4.2: Comparison between discrete regularized OT solver and C-PFG on two problems

	Problem 1			Problem 2	
	Std in x-axis	Std in y-axis	Error of map	Mean of norm	Error of map
Exact	1.000	0.500		2.250	
1000 samples	1.000±0.022	0.500±0.011		2.250±0.005	
RegOT, bs=50	0.925±0.030	0.456±0.008	0.102±0.011	2.195±0.007	0.234±0.057
C-PFG, bs=50, SWG	0.960±0.020	0.489±0.009	0.084±0.013	2.143±0.008	0.245±0.048
C-PFG, bs=50, WGAN	1.001±0.004	0.505±0.015	0.141±0.035	2.250±0.013	0.293±0.115
RegOT, bs=1000	All fail			All fail	
C-PFG, bs=1000, SWG	1.023±0.023	0.507±0.013	0.089±0.004	2.245±0.008	0.193±0.057
C-PFG, bs=1000, WGAN	1.027±0.027	0.516±0.015	0.118±0.010	2.244±0.009	0.201±0.058

We first set the batch size as 50, which is the default batch size of the discrete regularized optimal transport solver. In this case, the errors of maps are similar for all three methods, while the output distributions of the continuous potential flow generator in WGAN-GP match the best with the target ones. As is well known, the gradients of the sample Wasserstein loss are biased [7], thus the output distributions are biased for SWG, especially when the batch size is small. Therefore, we increased the batch size to 1000. In this case, we encountered “NAN” problems for all the runs for the discrete RegOT solver. SWG and WGAN-GP with continuous potential flow generators are stable, and we can see an improvement in the output distributions for the SWG, as we expected.

More Benchmark Problems in GAN and Normalizing Flow Models

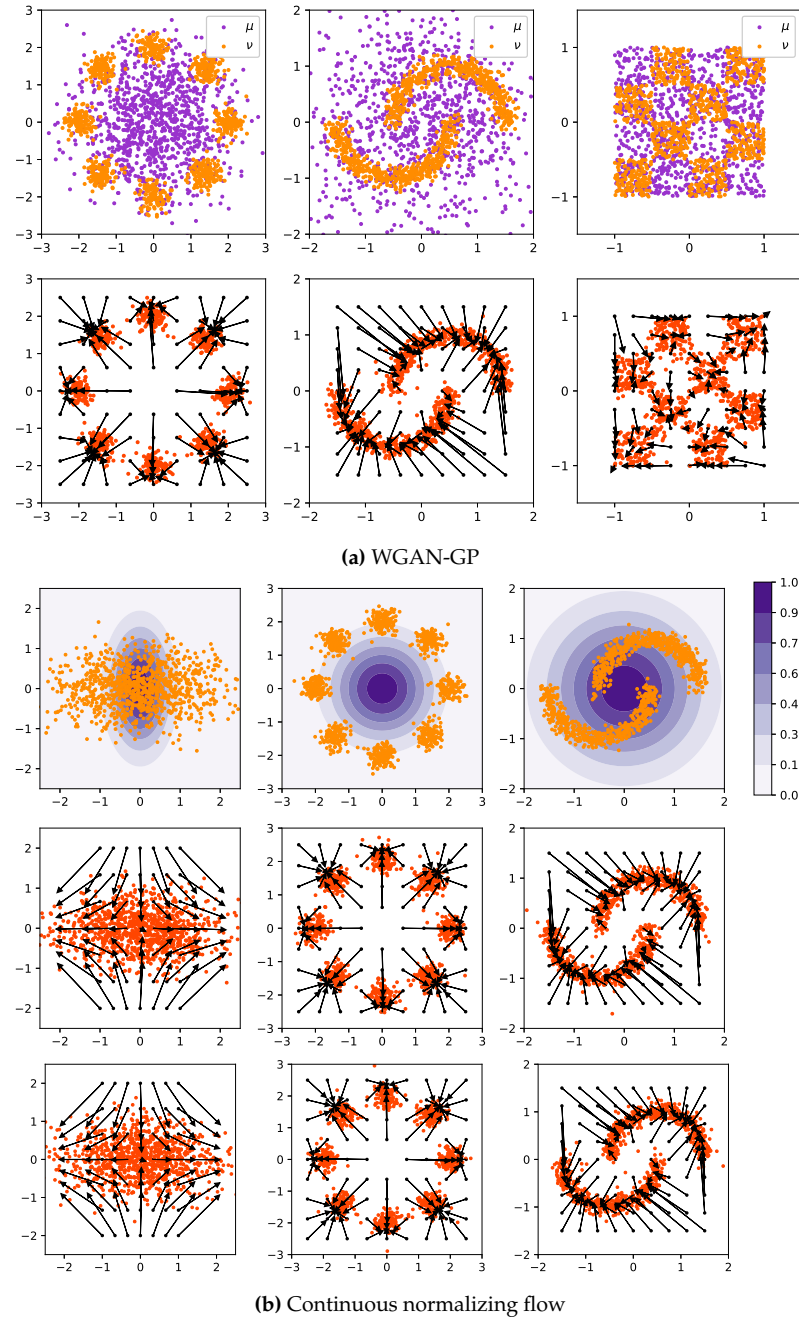


Figure 4.3: Potential flow generator in (a) WGAN-GP and (b) continuous normalizing flow for different problems with (middle row) first-order Euler scheme and (bottom row) second-order midpoint scheme. Each column shows the setup and results of one problem. The top row shows the samples or the unnormalized density functions of μ (purple) and ν (orange), the other rows show the map estimated by potential flow generator G and samples of $G_{\#}\mu$.

Apart from the previous two problems, we also apply WGAN-GP and continuous normalizing flow models with continuous potential flow generators to several more complicated distributions. The results are illustrated in Figure 4.3.

We can see the match between $G_{\#}\mu$ and ν in each of the problems, as well as that the samples of μ tend to be mapped to nearby positions. This shows the effectiveness of the continuous potential flow generator in various generative models, as well as the characteristics of “proximity” in the potential flow generator maps due to the L_2 optimal transport regularity.

4.4.2 Image Translation Tasks

In this section, we aim to show the capability of continuous potential flow generator in dealing with high dimensional problems, and also to show its advantage in image translations tasks with unpaired training data. We use WGAN-GP for the GAN loss. Before feeding the images into the generators, we embed them into a Euclidean space, where the L_2 distances between embedding vectors should quantify the similarities between images. In this chapter we apply the principal component analysis (PCA) [49], a simple but highly interpretable approach to conduct the image embedding.

The MNIST Dataset

We first test the problem of translation between the MNIST images [62]. We divide the MNIST training dataset into two clusters: (a) images of digits 0 to 4, and (b) images of digits 5 to 9. We view the two clusters of images as samples of μ and ν ,

respectively, i.e., we want to find the optimal transport map from images of digits 0 to 4 to images of digits 5 to 9. The dimensionality of Euclidean space, i.e. the total components in PCA, is set as 100.

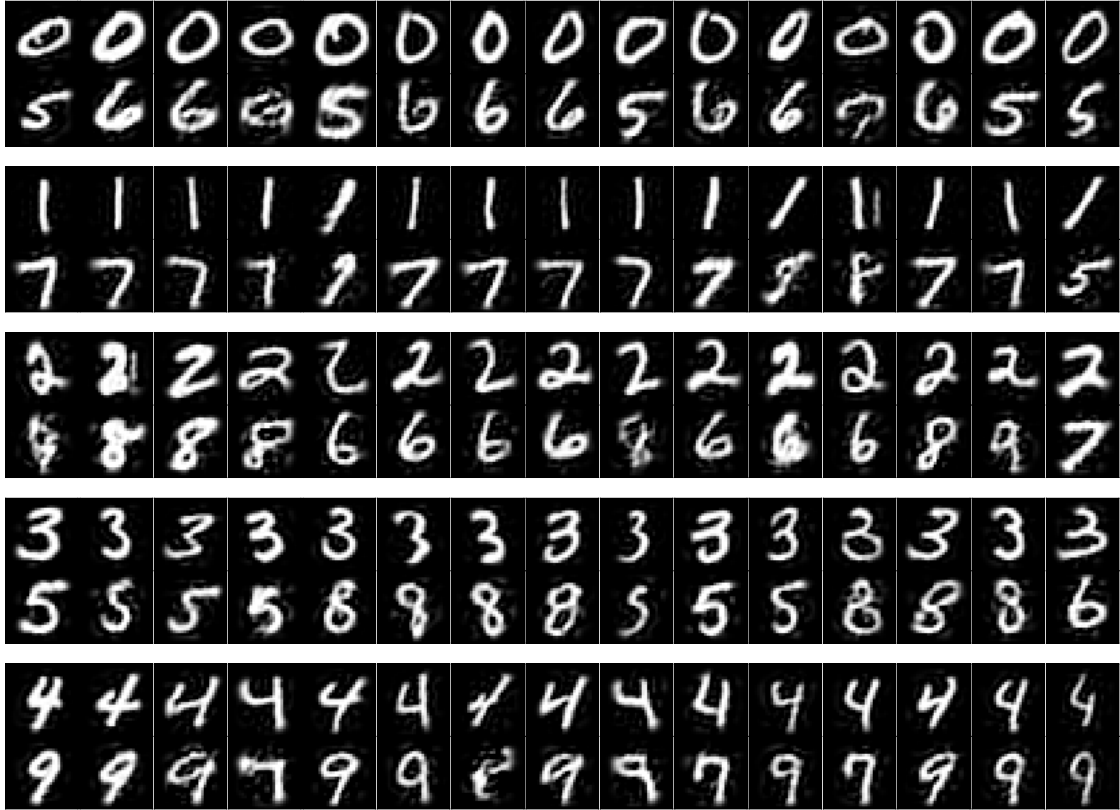
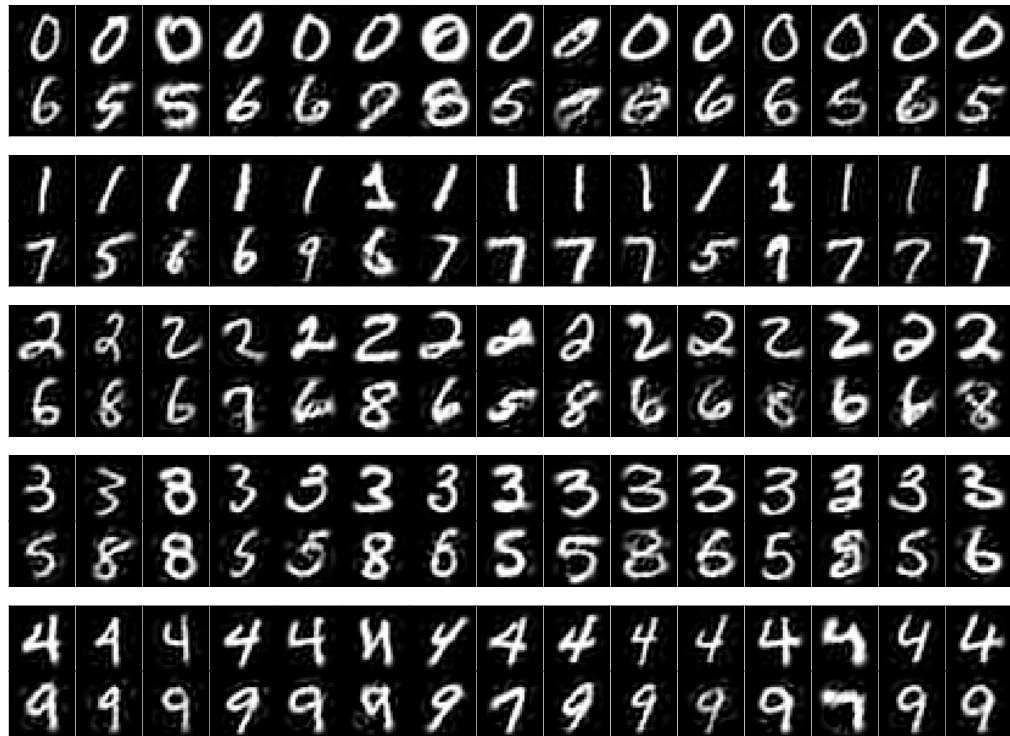
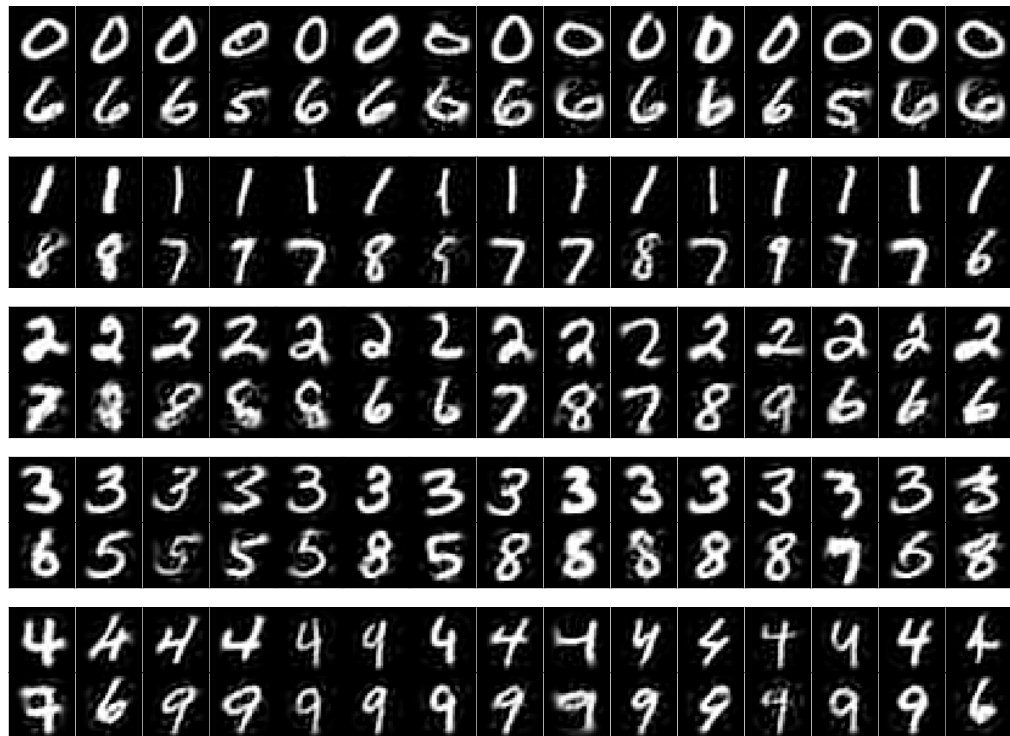


Figure 4.4: Potential flow generator on the MNIST. In each row, the top images are the inputs, while the bottom images are the corresponding outputs.

In Figure 4.4 we randomly pick images from the test dataset and show the corresponding inputs and outputs (more images in Figure 4.5). As we can see, the potential flow generator tends to translate images of digit 0 to digit 6, digit 1 to digit 7, digit 3 to digit 5 or 8, and digit 4 to digit 9. This is consistent with our previous discussion about the characteristics of “proximity” in that the input digits and output digits “look similar”, and the corresponding embedding vectors should be close in the L_2 distance.



(a) Training dataset



(b) Testing dataset

Figure 4.5: Potential flow generator on the MNIST (a) training and (b) testing datasets. In each row, the top images are reconstructed from the input vectors, while the bottom images are reconstructed from the corresponding output vectors.

The CelebA Dataset



Figure 4.6: Comparison between our method, vanilla WGAN-GP, CycleGAN with WGAN-GP as GAN loss and L_1 or L_2 loss as cycle-consistency loss, as well as official CycleGAN. The first two rows are the original images and their projections on the 700-dimensional Euclidean space induced by PCA, which are similar. The next five rows are the corresponding outputs. The last three rows are the reconstructed images from different CycleGANs.

In this section, we test the translation task between the CelebA images [72]. We randomly pick 60000 images from the CelebA training dataset and divide them into two clusters: (a) images with attribute “smiling” labeled as false, and (b) images with attribute “smiling” labeled as true. The images are cropped so that only faces remain on the images. We view the two clusters as samples of μ and ν , respectively. The total number of components in PCA is set to 700. Note that our goal is to generate images of smiling faces belonging to the same people as in input images. The difficulty lies in that the training data are unpaired and actually we do not have the ground truth as a reference.

We compare our method against vanilla WGAN-GP, and CycleGAN with

WGAN-GP for GAN loss. In CycleGAN we use L_1 or L_2 loss as cycle-consistency loss, with weight = 10 as in [156]. To make a fair comparison, for each generator and discriminator, we use a feedforward neural network (FNN) with the same size of hidden layers (5×256). We also test the official CycleGAN on this problem, where we feed the cropped face images (with resizing but without random jitter cropping) instead of embedding vectors into the model.² The results of images randomly picked from the test dataset are shown in Figure 4.6. More figures for our method are presented in Figure 4.7.

For most of the images, our method could successfully translate the no-smiling faces to smiling faces belonging to the same people. Some of the output images are blurred since it is difficult to learn the high order modes of PCA with FNN. Vanilla WGAN-GP and CycleGAN with FNN totally failed, in that the input and output images come from different people. This comparison clearly showed the necessity of additional regularity for the generators in translation tasks with only unpaired data, and that GAN loss + cycle-consistency loss cannot provide sufficient regularity.

The official CycleGAN is decent in performance, generating images less blurred than our method, but failed to change the countenance for some images. Note that the number of parameters in official CycleGAN is about 30 times more than that in our method, and the total training time is more than two times of our method on a single NVIDIA Tesla V100 GPU.

²We used the code from <https://www.tensorflow.org/tutorials/generative/cyclegan>. The size of the training dataset is reduced to 1000 for official CycleGAN, which is similar to the horse-to-zebra dataset. The number of total epochs is set back to 200 as in the original CycleGAN paper.



(a) Training dataset



(b) Testing dataset

Figure 4.7: Potential flow generator on the CelebA (a) training and (b) testing datasets. In each row, the top images are reconstructed from the input vectors, while the bottom images are reconstructed from the corresponding output vectors.

4.5 Summary of Chapter

In this chapter we propose potential flow generators with L_2 optimal transport regularity as plug-and-play generator modules that could be easily integrated into a wide range of generative models. In particular, we propose two versions: the discrete one and the continuous one. For the discrete version, the L_2 optimal transport regularity is directly encoded in, while for the continuous version we only need a slight augmentation to the original generator loss functions to impose the L_2 optimal transport regularity.

We firstly show that the potential flow generators are able to approximate the L_2 optimal transport maps in 2D problems. The continuous potential flow generator outperforms the discrete one as well as the discrete regularized optimal transport solver [122] in robustness. The continuous potential flow generator is also applied to WGAN-GP and continuous normalizing flow models to show its extensive application in generative models, where we also illustrate the characteristic of “proximity” for the potential flow generator due to the L_2 optimal transport regularity. Subsequently, we show the effectiveness of our method in image translation tasks using unpaired training data from the MNIST dataset and the CelebA dataset. Our method can yield decent results with only PCA, a linear embedding method, and without convolutional neural networks. We can also see that our method significantly outperforms the vanilla WGAN-GP and CycleGAN using feedforward neural networks as generators.

Apart from image-to-image translations, it is also possible to apply the potential flow generator to other translation tasks, if the translation objects could be properly embedded into Euclidean space. This paves the way to mathemati-

cally interpretable learning machines. Moreover, as we discussed in Section 4.3.3, the potential flow generator naturally provides the inverse map, and thus could be directly used to infer a latent representation for a given data sample in GAN models [21].

In this chapter, the linear embedding method in translation tasks actually leads to the blurred patterns in the output images, which is also the case (even more severe) for vanilla WGAN-GP and CycleGAN using the same embedding method. A possible improvement to our method is to combine the potential flow generator with other embedding methods, e.g. autoencoders with convolutional neural networks.

CHAPTER FIVE

Measure-Conditional Discriminator with Stationary Optimum for GANs and Statistical Distance Surrogates

5.1 Introduction

Generative adversarial networks (GANs) [37] have proven to be successful in training generative models to fit the target distributions. Apart from tasks of image generation [16, 156], text generation [154, 32], etc., GANs have also been used in physical problems to infer unknown parameters in stochastic systems [140, 143, 138]. Due to the variety of the generative models, GAN loss functions, and the need for high accuracy inferences, such tasks usually set a strict requirement to the robustness of GANs, as well as the similarity between the generated and target distributions in various metrics.

The optimum for the discriminator is, in general, non-stationary, i.e., it varies during the training, since it depends on the generated distributions. Such issue could lead to instability or oscillation in the training. Here, we propose a simple but effective modification to the discriminator as a plug-and-play module for different GANs, including vanilla GANs [37], Wasserstein GANs with gradient penalty (WGAN-GP) [43], etc. The main idea is to make the discriminator conditioned on the generated distributions, so that its optimum is stationary during the training.

The neural network architecture of the measure-conditional discriminator is adapted from DeepSets neural network [147], which is widely used in point cloud related tasks. It is also used in GANs [66] where each sample corresponds to a point cloud, while we target on more general tasks where each sample corresponds to a particle, an image, etc. In [74], the discriminator takes the mixture of real and generated distributions (instead of individual samples) as input, but it also has a non-stationary target optimum, and performs worse than our discriminators in experiments. We also emphasize the difference between the measure-conditional

discriminator and conditional GANs [85]. Conditioned on a vector featuring the target distributions, the conditional GANs still have a non-stationary target optimum for the discriminator and are limited to the scenarios where the samples can be categorized. Moreover, the measure-conditional discriminator can also be applied in conditional GANs, by making the original discriminator further conditioned on the generated distributions.

In Section 5.2 we discuss why we need stationary target optimum for the discriminator. In Section 5.3 we give a detailed description of the proposed discriminator neural networks and how to apply them in GANs. In Section 5.4 we extend the application of measure-conditional discriminators as surrogate models of statistical distances. The experimental results are shown in Section 5.5 with supplementary information in Section 5.6. We conclude in Section 5.7.

5.2 Why Stationary Target Optimum for the Discriminator

In general, there are two mathematical perspectives for GANs. The first perspective is to view GANs as a two-player zero-sum game between the generator G and the discriminator D . The hope is that the iterative adversarial training of G and D will lead to the Nash equilibrium of this zero-sum game, where the generated distribution will be identical to the target distribution of real data. The second perspective is that the discriminator gives the distance between the generated distribution and the target distribution in a variational form. For example, vanilla

GANs can be formulated as:

$$\begin{aligned}
& \min_G \max_D V(G_{\#}\mathcal{N}, D), \\
V(G_{\#}\mathcal{N}, D) &= \mathbb{E}_{x \sim G_{\#}\mathcal{N}} \log(1 - D(x)) + \mathbb{E}_{x \sim Q} \log(D(x)) \\
&= \mathbb{E}_{z \sim \mathcal{N}} \log(1 - D(G(z))) + \mathbb{E}_{x \sim Q} \log(D(x)),
\end{aligned} \tag{5.1}$$

while Wasserstein GANs (WGANs) can be formulated as:

$$\begin{aligned}
& \min_G \max_{D \text{ is 1-Lipschitz}} V(G_{\#}\mathcal{N}, D), \\
V(G_{\#}\mathcal{N}, D) &= -\mathbb{E}_{x \sim G_{\#}\mathcal{N}} D(x) + \mathbb{E}_{x \sim Q} D(x) \\
&= -\mathbb{E}_{z \sim \mathcal{N}} D(G(z)) + \mathbb{E}_{x \sim Q} D(x),
\end{aligned} \tag{5.2}$$

where Q represents the target distribution, $\mathcal{N}$ represents input noise, $\#$ is the push-forward operator, thus $G_{\#}\mathcal{N}$ represents the generated distribution. In vanilla GANs and WGANs, $\max_D V(G_{\#}\mathcal{N}, D)$ and $\max_{D \text{ is 1-Lipschitz}} V(G_{\#}\mathcal{N}, D)$ are nothing but the Jensen-Shannon (JS) divergence and the Wasserstein-1 distance up to constants between $G_{\#}\mathcal{N}$ and Q , respectively.

5.2.1 Non-Stationary Target Optimum Hurts: An Illustrative Example

From both two perspectives of GANs, the discriminator will approach its optimum D^* in each iteration. However, we will use the following illustrative example to demonstrate that D^* could be totally different as we perturb the generator, and such issue would lead to the oscillation of both generator and discriminator during training. This illustrative problem is adapted from [24] with different analysis. We first consider a linear discriminator as well as a translation function as the

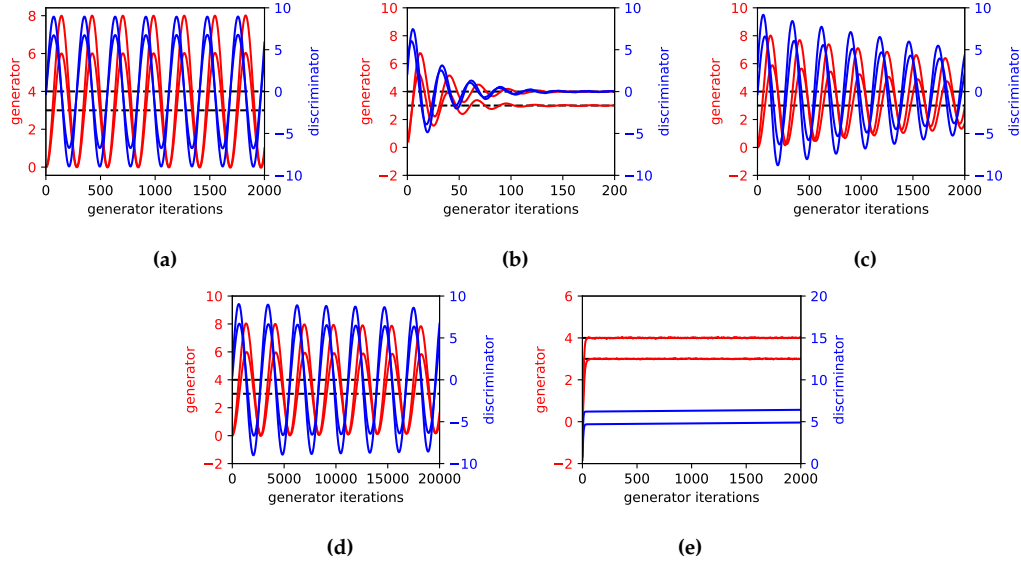


Figure 5.1: Results for the illustrative example of oscillation. (a): vanilla discriminator using gradient descent with learning rate 0.01. (b-d): vanilla discriminator using optimistic mirror descent, with learning rate 0.1, 0.01, and 0.001, respectively. Note that the multiplications of learning rate and training iterations are kept the same. (e): measure-conditional discriminator using gradient descent with learning rate 0.01. The red and blue lines show the generator and discriminator parameters in both two dimensions, while the black horizontal lines represent the ground truth for the generator parameters.

generator, i.e.,

$$D_w(x) = \langle w, x \rangle = \sum_{i=1}^d w_i x_i, \quad (5.3)$$

$$G_\theta(z) = z + \theta, z \sim \mathcal{N}(0, I) =: \mathcal{N}$$

with the target real distribution $Q = \mathcal{N}(v, I)$. The goal is to learn θ with ground truth $\theta^* = v$.

The WGAN with weight-clipping is formulated as

$$\min_{\theta} \max_{|w_i| \leq c} -\mathbb{E}_{z \sim \mathcal{N}} D(G(z)) + \mathbb{E}_{x \sim Q} D(x), \quad (5.4)$$

where $c > 0$ is the weight-clipping bound. In practice, we will use the empirical distributions to calculate the expectations, but if we calculate it analytically, we

have the following min-max formulation:

$$\min_{\theta} \max_{|w_i| \leq c} \langle w, v - \theta \rangle. \quad (5.5)$$

This two-player game has a unique equilibrium at $\theta = v, w = 0$, which appears to be satisfactory. However, if we set $\theta = v + \epsilon$, where $\epsilon \neq 0$ is the inevitable small fluctuation vector due to the randomness of the training data, moments in the optimizer, etc., then w would achieve the corresponding optimum at $-\text{sgn}(\epsilon)c$, where “sgn” denotes the component-wise sign function. In other words, the optimal w would jump between c and $-c$ for each entry as θ fluctuates around the ground truth. Such issue of jumping optimum will lead to the oscillation of both the generator and discriminator during training, as is illustrated in Figure 5.1a, where we test on a 2D problem with ground truth $v = (3, 4)$ and $c = 10$. Note that even if we set the discriminator as a general 1-Lipschitz function as in Equation 5.2, the corresponding optimum will be $D^*(x) = -\epsilon \cdot x/|\epsilon| + d$, which is still sensitive to the small fluctuation ϵ , where d is an arbitrary constant.

To remove the oscillation, [24] proposed to replace the gradient descent (GD) for the min-max formulation (5.5)

$$\begin{aligned} w_{t+1} - w_t &= \eta(v - \theta_t) =: \eta \dot{w}_t, \\ \theta_{t+1} - \theta_t &= \eta w_t =: \eta \dot{\theta}_t, \end{aligned} \quad (5.6)$$

with the optimistic mirror descent (OMD)

$$\begin{aligned} w_{t+1} - w_t &= 2\eta(v - \theta_t) - \eta(v - \theta_{t-1}) = \eta \dot{w}_t - \eta^2 \dot{\theta}_{t-1}, \\ \theta_{t+1} - \theta_t &= 2\eta w_t - \eta w_{t-1} = \eta \dot{\theta}_t + \eta^2 \dot{w}_{t-1}, \end{aligned} \quad (5.7)$$

where η is the learning rate. However, we report that the oscillation decay with

OMD could be very slow when a small learning rate is used, as is illustrated in Figures 5.1b, 5.1c, 5.1d. This is because the differences between the OMD and GD update, $-\eta^2\dot{\theta}_{t-1}$ and $\eta^2\dot{w}_{t-1}$, are second order w.r.t. η , i.e., one order higher than the GD update, $\eta\dot{w}_t$ and $\eta\dot{\theta}_t$. The difference between the GD and OMD dynamics thus vanishes as η goes to zero.

In the following, we will propose a much simpler and more effective strategy to remove these oscillations.

5.2.2 Benefits of Stationary Target Optimum

Since the aforementioned problem is due to the fact that the target optimum for the discriminator is non-stationary, to remove the oscillation, we propose to modify the discriminator architecture so that its target optimum is stationary during the training.

While keeping the generator and min-max formulation unchanged as in Equations 5.3 and 5.4, we set the discriminator

$$D_w(x) = \sum_{i=1}^d w_i x_i (\mathbb{E}_{x \sim Q}(x) - \mathbb{E}_{z \sim \mathcal{N}}(G_\theta(z)))_i \quad (5.8)$$

where $(\cdot)_i$ denotes the i -th component. The only differences between Equations 5.3 and 5.8 are the weights for $w_i x_i$. If we calculate the expectations in the min-max formulation 5.4 analytically, we will have

$$\min_{\theta} \max_{|w_i| \leq c} \sum_{i=1}^d w_i (v_i - \theta_i)^2. \quad (5.9)$$

For this min-max problem, any w with non-negative entries and $\theta = v$ is a Nash equilibrium. If we set $\theta = v + \epsilon$ with $\epsilon \neq 0$, then w would achieve the corresponding optimum at c for each entry, i.e., the target optimum for the discriminator is stationary. As shown in Figure 5.1e, the oscillation is totally removed. Each entry of w is heading to the optimum c in the early stage of training, while the change becomes negligible after θ converges to v , indicating that the Nash equilibrium is achieved.

The magic of the above solution lies in the fact that by designing the discriminator properly, we have a stationary target optimum for the discriminator during the training. Is this possible for more general GAN tasks where generators and discriminators are neural networks, and the target distributions are more flexible?

Note that the discriminator in Equation 5.8 can be interpreted as a discriminator conditioned on the generated and target distribution, so for more general GAN tasks we can simply design the discriminator as

$$D_{mc} = D_{mc}(x, G_{\#}\mathcal{N}), \quad (5.10)$$

where $G_{\#}\mathcal{N}$ is the generated distribution. The target distribution Q is omitted in the input since it is usually fixed in a GAN task, but we will revisit this in Section 5.4. We name the discriminator in Equation 5.10 as a “*measure-conditional discriminator*” since it is conditioned on the probability measure corresponding to the generated distribution. The proposed measure-conditional discriminator can be a plug-and-play module in a variety of GANs. We only need to replace the original discriminator $D(\cdot)$ with $D_{mc}(\cdot, G_{\#}\mathcal{N})$, while the generator and the min-max formulation of GANs will be kept unchanged. A more detailed introduction of the measure-conditional discriminator in GANs will be presented in Section 5.3.

We can see that by taking $G_{\#}\mathcal{N}$ as part of the input, the measure-conditional discriminator will have a stationary target optimum during the training process. Indeed, for a general GAN problem originally formulated as

$$\min_{G \in \mathcal{G}} \max_{D \in \mathcal{D}} V(G_{\#}\mathcal{N}, D), \quad (5.11)$$

with two examples given in Equation 5.1 and Equation 5.2, the target optimum for the measure-conditional discriminator is

$$D_{mc}^*(x, G_{\#}\mathcal{N}) = \left(\arg \max_{D \in \mathcal{D}} V(G_{\#}\mathcal{N}, D) \right)(x). \quad (5.12)$$

Although $G_{\#}\mathcal{N}$ varies during the training, D_{mc}^* is a function of $G_{\#}\mathcal{N}$ and x is stationary.

From the perspective of statistical distances, the target optimum D_{mc}^* is exactly a surrogate model for the distance between $G_{\#}\mathcal{N}$ and Q . For example, in vanilla GANs,

$$\mathbb{E}_{z \sim \mathcal{N}} \log(1 - D_{mc}^*(G(z), G_{\#}\mathcal{N})) + \mathbb{E}_{x \sim Q} \log(D_{mc}^*(x, G_{\#}\mathcal{N})) \quad (5.13)$$

represents the JS divergence between $G_{\#}\mathcal{N}$ and Q up to constants, while in WGANs,

$$-\mathbb{E}_{z \sim \mathcal{N}} D_{mc}^*(G(z), G_{\#}\mathcal{N}) + \mathbb{E}_{x \sim Q} D_{mc}^*(x, G_{\#}\mathcal{N}) \quad (5.14)$$

represents the Wasserstein-1 distance between $G_{\#}\mathcal{N}$ and Q up to constants.

It is hard to attain the target optimum D_{mc}^* , considering that it is a function of measures. However, we note that D_{mc} does not need to attain D_{mc}^* for the convergence of GANs. Indeed, we only require D_{mc} to approximate the optimum for $G_{\#}\mathcal{N}$, instead of the whole space of probability measures.

The vanilla discriminator only utilizes the result of the previous *one* iteration to provide the initialization. If the optimum is sensitive to the generated distribution as in the above illustrative example, in each iteration, the vanilla discriminator need to “forget the wrong optimum” inherited from the previous iteration and head for the new one in a few discriminator updates. In contrast, the measure-conditional discriminator progressively head for the stationary target optimum in *all* the iterations. In fact, even outdated generated distributions can be used to train the measure-conditional discriminator. If the optimum is sensitive to the generated distribution, the measure-conditional discriminator does not need to forget the inheritances from previous iterations, but only need to learn the sensitivity w.r.t. the input measure.

To some extent, the generator and the measure-conditional discriminator are trained in a *collaborative* way, in that the generator adaptively produces new distributions as training data to help the discriminator approximate D_{mc}^* , while the discriminator provides statistical distances to help the generator approach the target distribution. This concept is actually similar to reinforcement learning in the actor-critic framework [40], with parallelism between the generator and actor, as well as between the discriminator and critic.

5.3 Measure-Conditional Discriminator in GANs

Proposed in [147], the DeepSets neural network having the form of $H(X) = g(\sum_{x_i \in X} f(x_i))$ is widely used to represent a function of a point cloud X . The summation can be replaced by averaging to represent a function of probability measure P , i.e., $H(P) = g(\frac{1}{n} \sum_{i=1}^n f(x_i))$ where $\{x_i\}_{i=1}^n$ are samples from P .

In order to take a probability measure and an individual sample simultaneously as the discriminator input, we adapt the neural network architecture above to get

$$D_{mc}(x, P) = h(\mathbb{E}_{y \sim P}[f(y)], g(x)) \approx h\left(\frac{1}{n} \sum_{i=1}^n f(y_i), g(x)\right) \quad (5.15)$$

where f , g and h are neural networks, and $\{y_i\}_{i=1}^n$ are samples from P .

The measure-conditional discriminator is a plug-and-play module in a various GANs. The only modification is to replace $D(\cdot)$ with $D_{mc}(\cdot, G_{\#}\mathcal{N})$. For example, in vanilla GANs, the loss functions for the generator and the discriminator are

$$\begin{aligned} L_g &= \mathbb{E}_{z \sim \mathcal{N}} \log(1 - D_{mc}(G(z), G_{\#}\mathcal{N})) \\ &\quad + \mathbb{E}_{x \sim Q} \log(D_{mc}(x, G_{\#}\mathcal{N})), \\ L_d &= -\mathbb{E}_{z \sim \mathcal{N}} \log(1 - D_{mc}(G(z), G_{\#}\mathcal{N})) \\ &\quad - \mathbb{E}_{x \sim Q} \log(D_{mc}(x, G_{\#}\mathcal{N})), \end{aligned} \quad (5.16)$$

respectively. In WGAN with gradient penalty (WGAN-GP), the loss functions are

$$\begin{aligned} L_g &= -\mathbb{E}_{z \sim \mathcal{N}} D_{mc}(G(z), G_{\#}\mathcal{N}) + \mathbb{E}_{x \sim Q} D_{mc}(x, G_{\#}\mathcal{N}), \\ L_d &= \mathbb{E}_{z \sim \mathcal{N}} D_{mc}(G(z), G_{\#}\mathcal{N}) - \mathbb{E}_{x \sim Q} D_{mc}(x, G_{\#}\mathcal{N}) \\ &\quad + \lambda \mathbb{E}_{\hat{x} \sim \rho_{\hat{x}}} (\|\nabla_{\hat{x}} D_{mc}(\hat{x}, G_{\#}\mathcal{N})\|_2 - 1)^2, \end{aligned} \quad (5.17)$$

respectively, where λ is the weight for gradient penalty, and $\rho_{\hat{x}}$ is the distribution generated by sampling uniformly on interpolation lines between pairs of points sampled from real distributions and generated distributions. Note that the expectation over the real distribution Q cannot be removed from the generator loss, since this term depends on the generator now.

5.4 Measure-Conditional Discriminator for Statistical Distances Surrogate

The target distribution is usually fixed in GANs, thus omitted in the input of D_{mc} . Taking one step further, we will build a measure-conditional discriminator D_{sr} conditioned on two probability measures P and Q , which can act as a surrogate model to approximate the statistical distances between P and Q . Specifically, the neural network D_{sr} is formulated as

$$\begin{aligned} D_{sr}(x, P, Q) &= h(\mathbb{E}_{y \sim P}[f_1(y)], \mathbb{E}_{y \sim Q}[f_2(y)], g(x)) \\ &\approx h\left(\frac{1}{n_1} \sum_{i=1}^{n_1} f_1(y_i^P), \frac{1}{n_2} \sum_{i=1}^{n_2} f_2(y_i^Q), g(x)\right) \end{aligned} \quad (5.18)$$

where f_1, f_2, g and h are neural networks, and $\{y_i^P\}_{i=1}^{n_1}$ and $\{y_i^Q\}_{i=1}^{n_2}$ are samples from P and Q , respectively.

5.4.1 Unsupervised Training with Variational Formula

We will train D_{sr} using the variational form of the statistical distances, in the same spirit as in GANs. Here, we take the KL divergence as an example, which has the following variational formula [91]:

$$\begin{aligned} D_{KL}(P||Q) &= \sup_{g>0} \left(\mathbb{E}_{x \sim P}[\log(g(x))] - \mathbb{E}_{x \sim Q}[g(x)] + 1 \right), \\ &= \sup \left(\mathbb{E}_{x \sim P}[g(x)] - \mathbb{E}_{x \sim Q}[\exp(g(x))] + 1 \right). \end{aligned} \quad (5.19)$$

Thus, the loss function for D_{sr} can be written as

$$\begin{aligned} L_{KL} &= \mathbb{E}_{(P,Q) \sim \mu} [-l_{KL}(P, Q)] \\ l_{KL}(P, Q) &= \mathbb{E}_{x \sim P} [D_{sr}(x, P, Q)] - \mathbb{E}_{x \sim Q} [\exp(D_{sr}(x, P, Q))] + 1, \end{aligned} \quad (5.20)$$

where μ represents the distribution for the probability measure pairs (P, Q) in the training. Ideally, $l_{KL}(P, Q)$ will approximate $D_{KL}(P, Q)$ if D_{sr} achieves optimum. Similar loss functions can be constructed for many other statistical distances like JS divergence, total variation etc., provided with variational forms as in Equation 5.19. We also give an example of a surrogate model with results for the optimal transport map in Section 5.6.5.

Note that after the optimization of D_{sr} (which can be offline), via a forward propagation of D_{sr} , we can estimate $l_{KL}(P, Q)$ as an approximation of $D_{KL}(P, Q)$ for various (P, Q) pairs sampled from μ , and even for (P, Q) pairs that are never seen in the training procedure (thanks to the generalization of neural networks). The computational cost for the forward propagation grows linearly w.r.t. the sample size. More importantly, no labels are required to train D_{sr} . Instead, we only need to prepare samples of P and Q as training data.

Here, μ can, of course, be prescribed by the users. It can also be decided actively during the training, depending on specific tasks. In the context of GANs, with Q being different target distributions and P being the corresponding generated distributions, (P, Q) samples can be induced from a family of GAN tasks. (P, Q) samples can also be induced from a single GAN task, if we need to fit multiple target distributions simultaneously, e.g., the distributions at multiple time instants in time-dependent problems. We will demonstrate this with an example in Section 5.5.

5.4.2 Transfer Learning with Statistical Distance Surrogate

As a surrogate model of statistical distances between distributions, it is possible that a well-trained D_{sr} can be transferred to GANs and act as a discriminator without any update. For example, if D_{sr} is pretrained with Equation 5.20, the generator can be trained with the loss function $L_g = l_{KL}(G_{\#}\mathcal{N}, Q)$, with l_{KL} from Equation 5.20. However, training G with a frozen D_{sr} requires that $(G_{\#}\mathcal{N}, Q)$ is not an outlier of μ . This typically means that the degree of freedom for G is limited.

Alternatively, the pretrained D_{sr} can be employed as an initialization of the discriminator and be fine-tuned in GANs. With Q fixed as the target distribution, D_{sr} is reduced to a function of P and x , just as D_{mc} . We can then train it iteratively with the generator as in Section 5.3. Note that the loss function in GANs should coincide with that in the pretraining of D_{sr} . For example, if D_{sr} is pretrained with Equation 5.20, then the loss functions for the generator G is given by $L_g = l_{KL}(G_{\#}\mathcal{N}, Q)$, while the loss functions for D_{sr} is $-L_g$.

5.5 Computational Results

We show some results for the experimental comparisons in this section. The detailed neural network architectures are given in Section 5.6.1. We emphasize that although measure-conditional discriminators D_{mc} have more inputs than vanilla ones, the neural networks for both are designed to have almost the same number of parameters for the same problem. For each set-up in Section 5.5.1 and 5.5.2 we run the code with three different random seeds; the colored lines and shaded areas in the figures represent the mean and standard deviation.

5.5.1 2D Distributions

We first compare the vanilla discriminator and D_{mc} for different GAN set-ups on 2D problems. In particular, we test the vanilla GANs and WGAN-GP with different discriminator/generator iteration ratios, and (β_1, β_2) in the Adam optimizer (the initial learning rate is set as 0.0001).

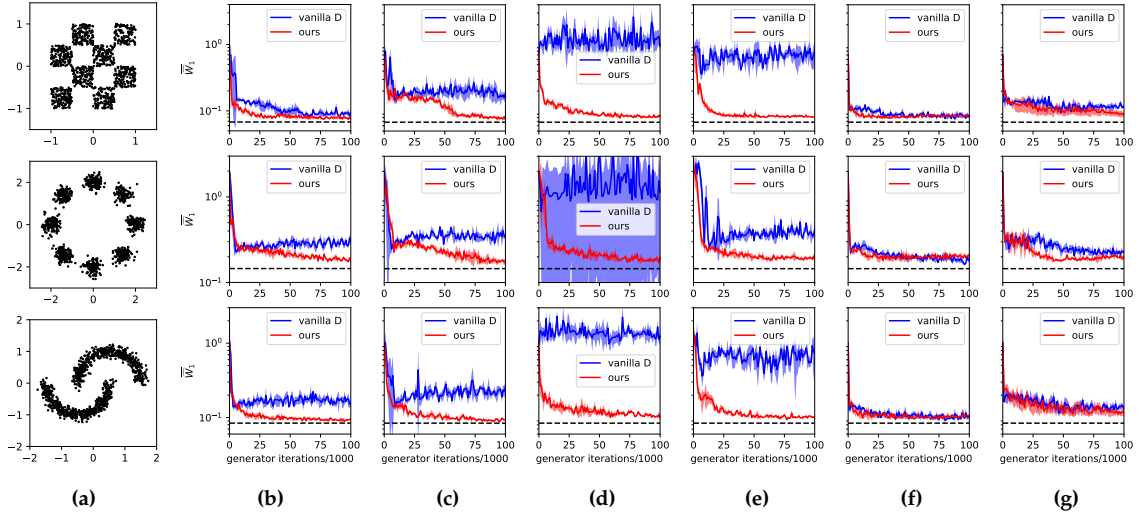


Figure 5.2: Comparison between the vanilla discriminator and measure-conditional discriminator D_{mc} (ours) in three 2D problems. Each row represents the results for one problem. (a): The three target distributions. (b-g): $\overline{W}_1(P, Q)$ against generator iterations, using different versions of GANs, discriminator/generator iteration ratios, and (β_1, β_2) in Adam optimizer. (b): Vanilla GAN, 1:1, (0.5, 0.9), (c): Vanilla GAN, 1:1, (0.9, 0.999), (d): WGAN-GP, 1:1, (0.5, 0.9), (e): WGAN-GP, 1:1, (0.9, 0.999), (f): WGAN-GP, 5:1, (0.5, 0.9), (g): WGAN-GP, 5:1, (0.9, 0.999). The y-axes are shared for each row, and the black dashed lines represent $\overline{W}_1(Q, Q)$.

To evaluate the generated distribution $P = G_{\#}\mathcal{N}$, we take the expectation of empirical Wasserstein-1 distance, i.e., $\overline{W}_1(P, Q) := \mathbb{E}_{\hat{P}_n, \hat{Q}_n}[W_1(\hat{P}_n, \hat{Q}_n)]$ as an approximation of $W_1(P, Q)$, where $\hat{P}_n$ is the (random) empirical measure of P with $n = 1000$ samples, similarly for $\hat{Q}_n$. We average over 100 empirical Wasserstein distances, which can be calculated via linear programming, to calculate the expectation. The target distributions and results are shown in Figure 5.2 with more results in Section 5.6.2. It is clear that for all set-ups except WGAN-GP with iteration ratio 5:1 and $(\beta_1, \beta_2) = (0.5, 0.9)$, the measure-conditional discriminator significantly out-

performs the vanilla discriminator in achieving smaller Wasserstein distances or converging faster. In fact, the measure-conditional discriminator is very robust w.r.t. the versions of GANs, the iteration ratio, and the optimizer hyperparameters, achieving approximately the same performance in different set-ups, in contrast to the vanilla discriminator.

5.5.2 Stochastic Dynamic Inference

To further show the advantage of measure-conditional discriminator, here we compare it with the vanilla discriminator on the problem of inferring stochastic dynamics from observations of particle ensembles, following the framework in [138]. Specifically, we consider a particle system whose distributions at $t > 0$, denoted as ρ_t , are determined by the initial distribution $\rho_0 = \mathcal{N}(0, 0.2^2)$ and the dynamics for each particle, which is governed by the stochastic ordinary differential equation:

$$dx = (a_0 + a_1x + a_2x^2 + a_3x^3)dt + \sigma dW_t, \quad (5.21)$$

where $a_0 = 0, a_1 = 1, a_2 = 0, a_3 = -1, \sigma = 1$, and W_t is the standard Brownian motion. We consider the scenario where we do not know $\rho_0, \{a_i\}_{i=0}^3$ and σ , but have observations of 10^5 indistinguishable particles at $t = 0.2, 0.5, 1.0$, which can be viewed as samples from $\rho_{0.2}, \rho_{0.5}$ and $\rho_{1.0}$. Our goal is to infer $\{a_i\}_{i=0}^3$ and σ from these observations.

Taking the standard Gaussian noise as input, the generator G is a feedforward neural network whose output distribution aims to approximate ρ_0 , followed by a first-order numerical discretization of Equation 5.21 with $\{a_i\}_{i=0}^3$ and σ replaced by trainable variables (the variable for σ is activated by a softplus function to guaran-

tee positivity), so that the particle distributions at any $t > 0$ can be generated. Note that we need to tune the feedforward neural network as well as the five trainable variables to fit the target distributions $\rho_{0.2}$, $\rho_{0.5}$ and $\rho_{1.0}$ simultaneously.

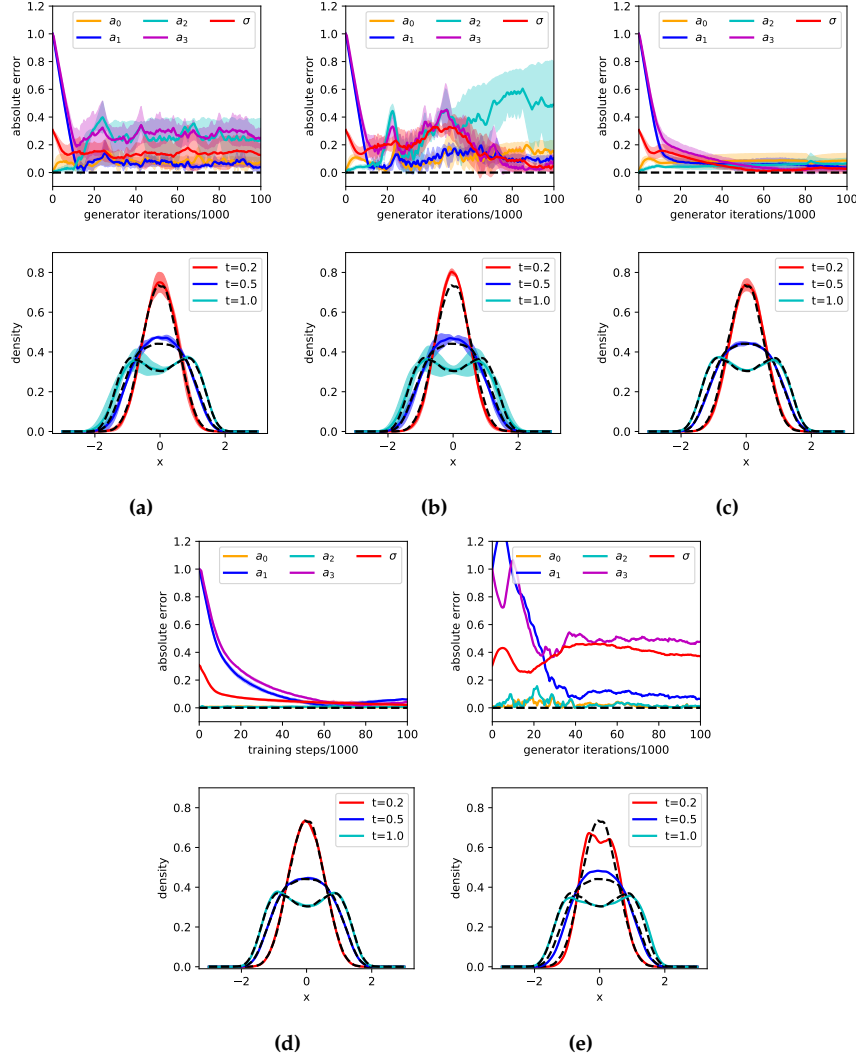


Figure 5.3: Comparison between different set-ups in the task of stochastic dynamic inference. (a): WGAN-GP, vanilla discriminator, Adam optimizer, (b): WGAN-GP, vanilla discriminator, Optimistic Adam optimizer, (c): WGAN-GP, D_{mc} , Adam optimizer, (d): WGAN-GP, D_{sr} , Adam optimizer, (e): BGAN with Adam optimizer. First row: the absolute error of the inferred dynamic parameters $\{a_i\}_{i=0}^3$ and σ against generator iterations. Second row: the generated distributions at $t = 0.2, 0.5, 1.0$, in the end of the training, with the dashed black lines showing the ground truth.

We compare the following set-ups in WGAN-GP: (a) vanilla discriminators with Adam optimizer, (b) vanilla discriminators with Optimistic Adam optimizer [24], which is the combination of Adam and optimistic mirror descent, (c) D_{mc}

in Equation 5.15 with Adam optimizer, (d) D_{sr} in Equation 5.18 with Adam optimizer. We emphasize that the discriminator/generator iteration ratio is set as 5:1 and $(\beta_1, \beta_2) = (0.5, 0.9)$, for which the measure-conditional discriminator does not outperform the vanilla one in Section 5.5.1. We also compare with another version of GAN, i.e., (e) BGAN [74], where the discriminator also takes a distribution (instead of individual samples) as input. In short, the BGAN discriminator takes the mixture of real and generated samples as input and aims to tell the ratio of real samples.

For set-up (a,b,c,e) we have to use three discriminators, denoted as $D_{0.2}, D_{0.5}, D_{1.0}$, to handle $\rho_{0.2}, \rho_{0.5}, \rho_{1.0}$, respectively. The losses for the discriminators and the generator are

$$\begin{aligned} L_{D_t} &= L_d(G, D_t, \rho_t), t = 0.2, 0.5, 1.0 \\ L_G &= \sum_{t \in S} L_g(G, D_t, \rho_t), S = \{0.2, 0.5, 1.0\} \end{aligned} \quad (5.22)$$

where $L_d(G, D_t, \rho_t)$ and $L_g(G, D_t, \rho_t)$ are discriminator and generator loss functions, given generator G , discriminator D_t , and a single target distribution ρ_t . We only need one D_{sr} in set-up (d) since D_{sr} can also take various ρ_t as input. In particular, the discriminator and generator loss functions for D_{sr} are $L_{D_{sr}} = \sum_{t \in S} L_d(G, D_{sr}, \rho_t)$ and $L_G = \sum_{t \in S} L_g(G, D_{sr}, \rho_t)$, respectively.

In Figure 5.3 we visualize the results for the inferred dynamic parameters $\{a_i\}_{i=0}^3$ and σ , as well as the generated distributions at $t = 0.2, 0.5, 1.0$ for each set-up. More results are presented in Section 5.6.3. Note that D_{mc} significantly outperforms the vanilla discriminator with the Adam or Optimistic Adam optimizer, even with a 5:1 iteration ratio. D_{sr} achieves results as good as if not better than D_{mc} , and the performance is almost independent of the random seed. Moreover, since only one discriminator is involved, set-up (d) has less than half discriminator parameters

compared with other setups. Such a difference in the model size will be even larger for problems with more time instants. As for BGAN, two out of three runs encountered the “NAN” issue, while the rest one did not outperform WGAN-GP with a measure-conditional discriminator.

5.5.3 Surrogate Model for KL Divergence

We consider the problem of approximating $D_{KL}(P||Q)$ by using the surrogate model D_{sr} . We set μ_{train} and μ_{test} as probability measures on d -dimensional Gaussian distributions, denoted as $\mathcal{N}(m, \Sigma)$. We set $m_i \sim U([0.15, 0.5] \cup [-0.5, -0.15])$ for μ_{train} , while $m_i \sim U([-0.15, 0.15])$ for μ_{test} , where the subscripts represent the component indices. For both μ_{train} and μ_{test} we set $\sqrt{\Sigma_{i,i}} \sim U([0.5, 1.0])$, and the correlation coefficient $\Sigma_{i,j} / \sqrt{\Sigma_{i,i}\Sigma_{j,j}} \sim U([-0.5, 0.5])$ for $i \neq j$.

The surrogate model is trained with P and Q i.i.d sampled from μ_{train} , with the batch size set as 100 for (P, Q) pairs, and 1000 samples for each P and Q , i.e., $n_1 = n_2 = 1000$ in Equation 5.18. The surrogate model is then tested on three cases: (a): $P \sim \mu_{train}, Q \sim \mu_{train}$, (b): $P \sim \mu_{train}, Q \sim \mu_{test}$, (c): $P \sim \mu_{test}, Q \sim \mu_{test}$. Note that there are $d(d+3)$ degrees of freedom for each (P, Q) pair. For such cases there exists an analytical formula for $D_{KL}(P||Q)$ as a ground truth: $D_{KL}(P||Q) = \frac{1}{2}[\log \frac{|\Sigma_q|}{|\Sigma_p|} - d + \text{tr}(\Sigma_q^{-1}\Sigma_p) + (m_p - m_q)^T \Sigma_q^{-1}(m_p - m_q)]$.

We compare the surrogate model against direct calculation via $D_{KL}(P, Q) = \mathbb{E}_{x \sim P}[\log(P(x)) - \log(Q(x))]$, where the densities are estimated via the kernel density estimation with Gaussian kernel and Scott’s rule [121], for dimensionality $d = 2$. In Figure 5.4 we quantify the accuracy against floating-point operations (FLOPs). Note that the computational cost grows linearly w.r.t. the sample size in the

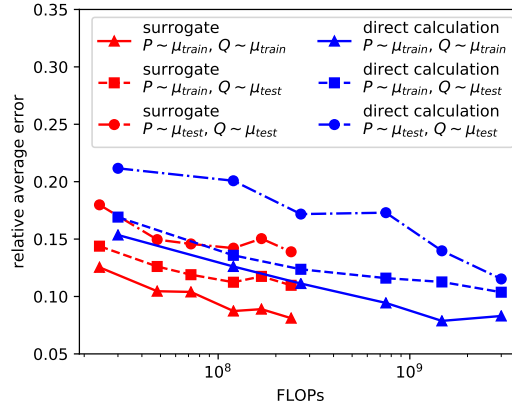


Figure 5.4: Comparison between the surrogate model and direct calculation for the KL divergence. For each line, the sample size, i.e., n_1 and n_2 in Equation 5.18, increases from 1000 to 10000, to improve accuracy in the cost of FLOPs.

surrogate model, while quadratically w.r.t. the sample size in the direct calculation. The surrogate model outperforms the direct calculation in that it can achieve smaller errors with the same FLOPs for all the three (P, Q) distributions in test. In Section 5.6.4, we show scatter plots of the inference against the ground truth for dimensionality $d = 2$ and 3, as well as the results of transferring the surrogate model to GANs.

5.6 Supplementary Material

5.6.1 Neural Network Architectures

In this section, we present the neural network architectures used in Section 5.5. All noises into the generators are multi-variant standard Gaussians. An additional sigmoid activation is applied to the discriminator outputs in vanilla GANs. We emphasize that the vanilla discriminator and measure-conditional discriminator D_{mc} share almost the same number of parameters in the same problem.

2D problem:

generator (33,666 parameters): $2 \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 2$; vanilla discriminator (33,537 parameters): $2 \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 1$; D_{mc} (33,921 parameters), f and g : $2 \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 64$, h : $128 \xrightarrow{\text{dense}} 128 \rightarrow \text{ReLU} \xrightarrow{\text{dense}} 1$.

Stochastic Dynamic Inference: generator for ρ_0 (33,409 parameters): $1 \xrightarrow{\text{dense}} 128 \rightarrow \tanh \xrightarrow{\text{dense}} 128 \rightarrow \tanh \xrightarrow{\text{dense}} 128 \rightarrow \tanh \xrightarrow{\text{dense}} 1$; vanilla discriminator (49,921×3 parameters): $1 \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 1$; D_{mc} (50,177×3 parameters), f and g : $1 \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 64$, h : $128 \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 1$. D_{sr} (66,881 parameters), f_1 , f_2 and g : $1 \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 64$, h : $192 \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 128 \rightarrow \text{LeakyReLU} \xrightarrow{\text{dense}} 1$. The discriminator architecture in BGAN is from the original paper [74], with 4 hidden layers, each of width 128, and LeakyReLU activation, 99,329×3 parameters in total.

KL Surrogate Model: D_{sr} (6,137 parameters), f_1 , f_2 and g : $2 \xrightarrow{\text{dense}} 32 \rightarrow \tanh \xrightarrow{\text{dense}} 32 \rightarrow \tanh \xrightarrow{\text{dense}} 8$, h : $24 \xrightarrow{\text{dense}} 32 \rightarrow \tanh \xrightarrow{\text{dense}} 32 \rightarrow \tanh \xrightarrow{\text{dense}} 1$.

5.6.2 More Results for 2D Problems

In Figure 5.5 we show the comparison between the vanilla discriminator and measure-conditional discriminator D_{mc} on three 2D problems, using vanilla GAN with 5:1 discriminator/generator iteration ratio. We encountered “NAN” issue occasionally with both discriminators, the corresponding runs are omitted. The

measure-conditional discriminator outperforms the vanilla one, as in Section 5.5.

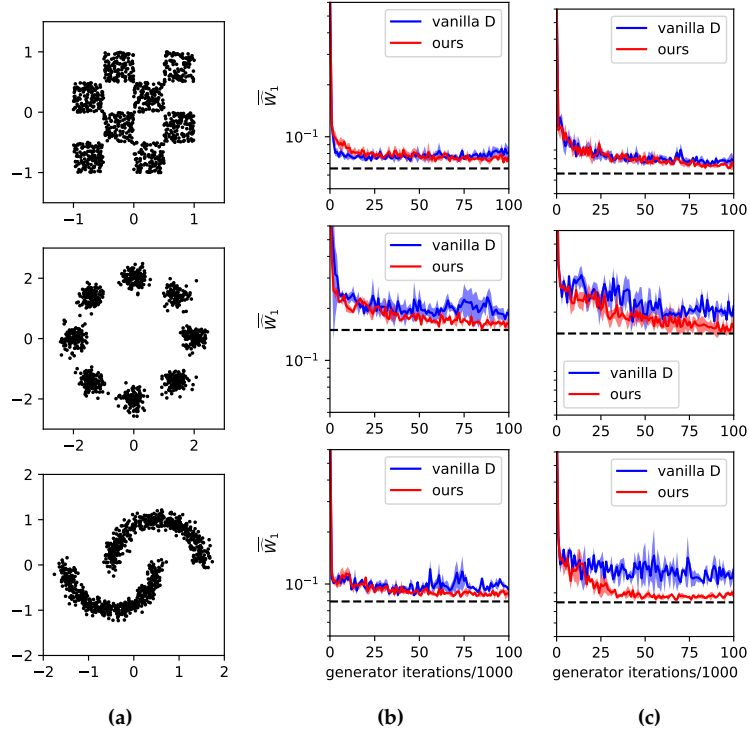


Figure 5.5: More results for the comparison between the vanilla discriminator and measure-conditional discriminator (ours) in three 2D problems. (a): The three target distributions. (b): Vanilla GAN, 5:1, (0.5, 0.9), (c): Vanilla GAN, 5:1, (0.9, 0.999). See a more detailed caption in Figure 5.2

5.6.3 More Results for Stochastic Dynamic Inference

In Figure 5.6 we show the results of D_{mc} and D_{sr} with the Optimistic Adam optimizer on the task of stochastic dynamic inference, using the same neural networks as in Section 5.5. The results are similar to those with the Adam optimizer, with a slight improvement.

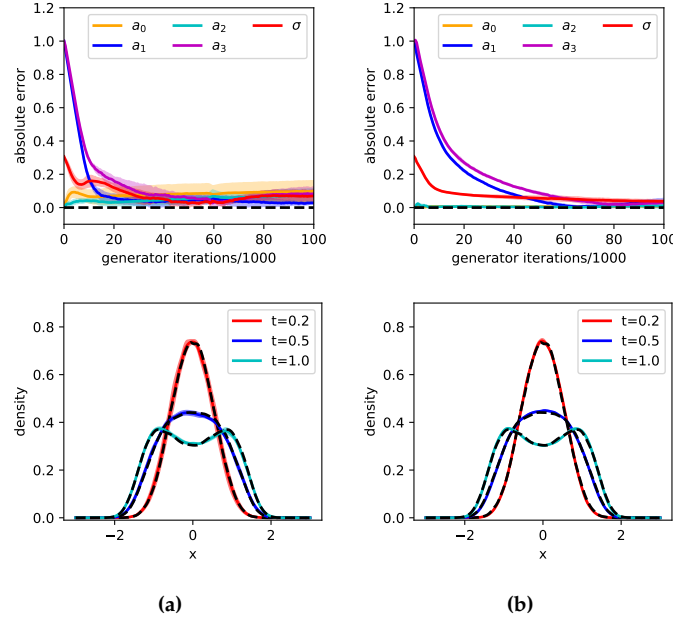


Figure 5.6: Comparison between different set-ups in the task of stochastic dynamic inference. (a): WGAN-GP, D_{mc} , Optimistic Adam optimizer, (b): WGAN-GP, D_{sr} , Optimistic Adam optimizer. See a more detailed caption in Figure 5.3.

In addition, in Figure 5.7 we show the results using smaller neural networks for the vanilla discriminator and D_{mc} (the number of hidden layers for the vanilla discriminator and h in D_{mc} is reduced by 1). For the vanilla discriminator, the Optimistic Adam optimizer manages to remove the high-frequency oscillation, compared with the Adam optimizer, but the inferred parameters are still incorrect. In contrast, both optimizers give good inference with D_{mc} discriminator, and the Optimistic Adam optimizer performs better in that the inference converges faster.

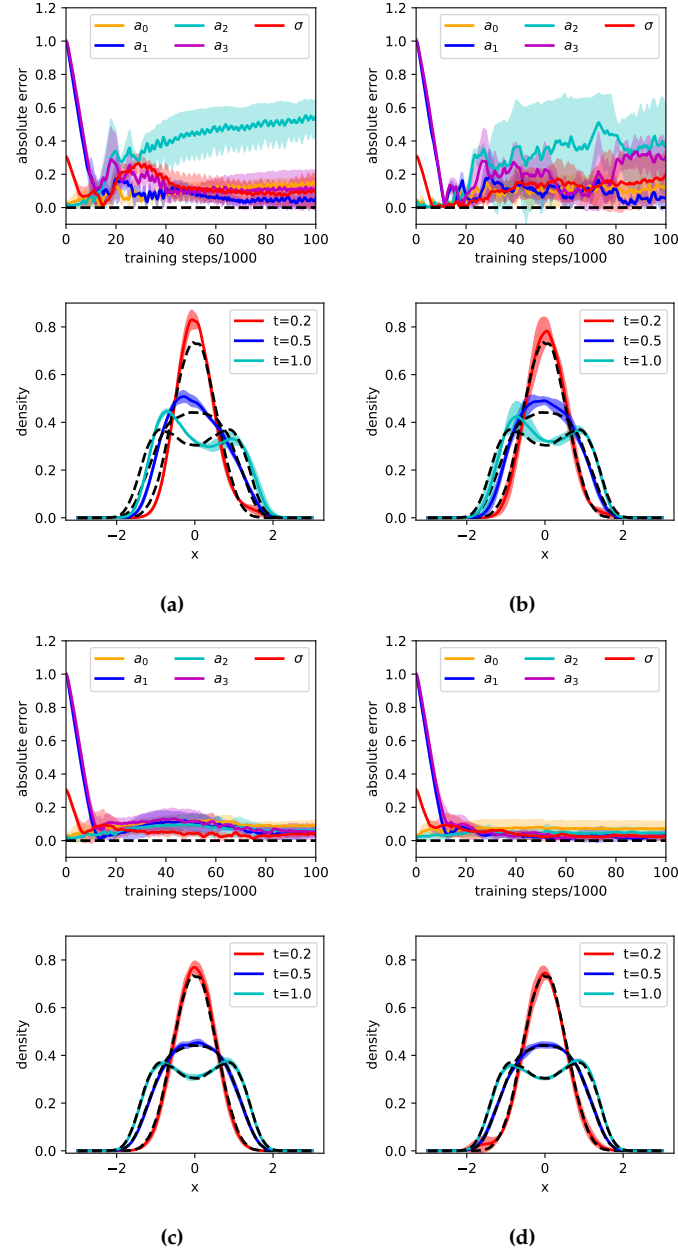


Figure 5.7: Comparison between different set-ups in the task of stochastic dynamic inference, using smaller discriminator neural networks. (a): WGAN-GP, vanilla discriminator, Adam optimizer, (b): WGAN-GP, vanilla discriminator, Optimistic Adam optimizer, (c): WGAN-GP, D_{mc} , Adam optimizer, (d): WGAN-GP, D_{mc} , Optimistic Adam optimizer. See a more detailed caption in Figure 5.3.

5.6.4 More Results for the Statistical Distance Surrogate

As a supplement of Figure 5.4, in Figure 5.8a, 5.8b we show the scatter plots of the inference of KL divergence against the ground truth for dimensionality $d = 2$. In Figure 5.8c, 5.8d we also show the results for the 3D case, with a larger D_{sr} neural network (128 as the hidden layer width and 32 as the output dimension of f_1 , f_2 and g).

As a proof of concept, we then employ the 2D surrogate model as a discriminator in GAN. The target distribution is set as $\mathcal{N}(m, \Sigma)$ with $m = [0.1, -0.1]$ and $\Sigma = \text{diag}([0.3, 0.6])$, which is a sample from μ_{test} . The generator is defined as $G(z) = Az + b$, and we test with two generator set-ups: (1) $A = \text{diag}([a_1, a_2])$, $b = [b_1, b_2]$ with 4 degrees of freedom, and (2) $A = [[a_1, a_2], [0, a_3]]$, $b = [b_1, b_2]$ with 5 degrees of freedom, both having ground truth for the parameters. We compare the following three set-ups of the discriminator: (a) D_{sr} transferred from the well-trained surrogate model and is further trained in GAN, (b) transferred D_{sr} without further training in GAN, (c) random initialized D_{sr} with training in GAN. The generator parameters during the GAN training are visualized in Figure 5.8e, 5.8f. One can see that for discriminator set-up (b), the generator parameters are not too bad in the first generator set-up with 4 degrees of freedom, but totally failed in the second generator setup. A possible explanation is that $G_{\#}\mathcal{N}$ becomes an outlier of μ_{train} during the training and thus D_{sr} cannot provide correct statistical distances. Discriminator set-ups (a) and (c) worked well in both generator setups, but note that the set-up (a), i.e. the one with transfer learning, converges faster and does not have the burrs on the curve. This demonstrates the benefit of the transfer learning with the pretrained D_{sr} .

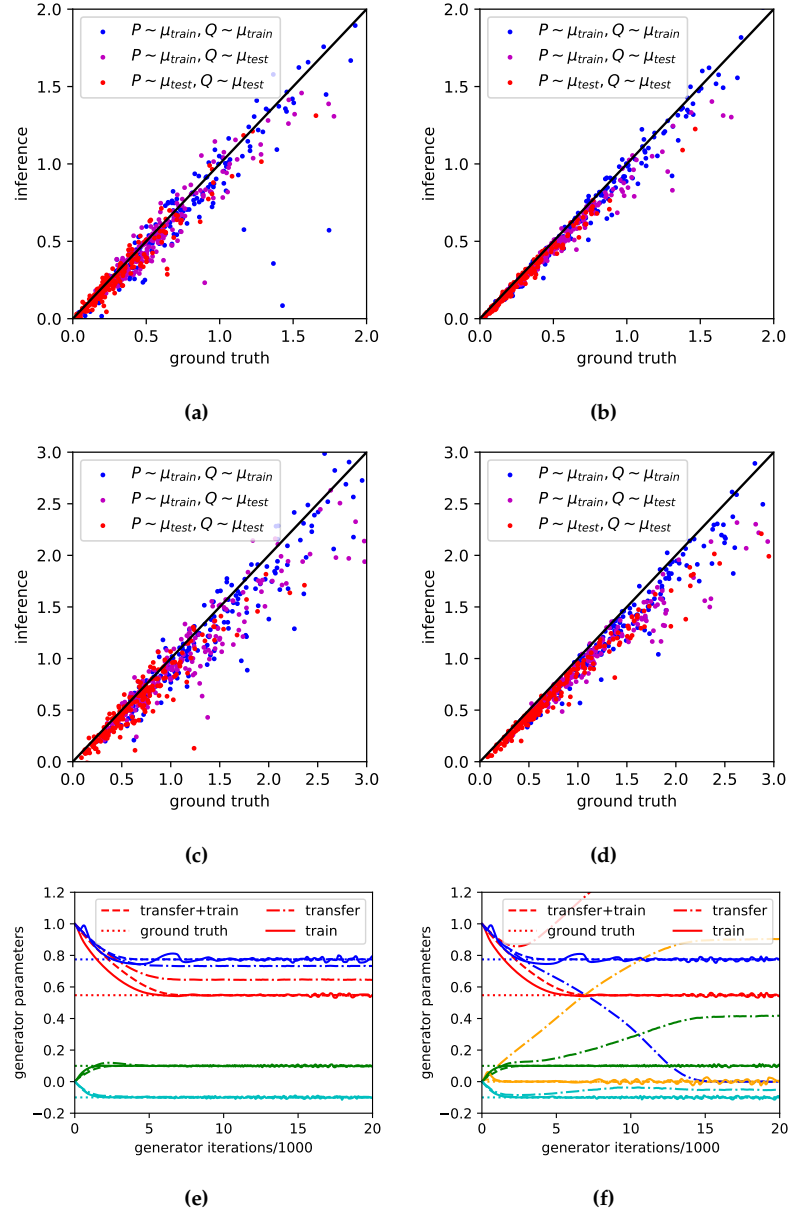


Figure 5.8: Results for the KL divergence surrogate model. (a-d): Inference against the ground truth. (a): 2D, 1000 samples, (b): 2D, 10000 samples, (c): 3D, 1000 samples, (d): 3D, 10000 samples. (e-f) Generator parameters during the GAN training with different generator and discriminator set-ups. (e): The first generator set-up, with 4 degrees of freedom. (f): The second generator set-up, with 5 degrees of freedom. Different colors represent different generator parameters, while different line styles represent the results from different discriminator setups and the ground truth.

5.6.5 Surrogate Model for Optimal Transport Map

In [123] the authors proposed a two-step method for learning the barycentric projections of regularized optimal transport, as approximations of optimal transport maps between continuous measures. Their method solves the map between one pair of measures in one training process, but we can make a modification with measure-conditional discriminators and obtain a surrogate model for the optimal transport maps between various pairs of measures.

Specifically, we use two D_{sr} neural networks, denoted as $D_{sr,G}$ and $D_{sr,D}$ to approximate the transport map and an auxiliary function, respectively. The first step in [123] is to maximize

$$\mathbb{E}_{x \sim P, y \sim Q} [u(x) + v(y) - \frac{1}{4\epsilon}(u(x) + v(y) - c(x, y))_+^2], \quad (5.23)$$

which is the variational form of the optimal transport cost with L^2 regularization, where u and v are two neural network to train, $\epsilon = 0.02$ is the regularization weight, and $c(x, y)$ is set as $\|x - y\|^2$. Utilizing the symmetry between the optimal u and v if we swap P and Q , we use $D_{sr,D}(P, Q, x)$ and $D_{sr,D}(Q, P, y)$ to replace $u(x)$ and $v(y)$, respectively. The loss function for $D_{sr,D}$ writes as

$$\begin{aligned} L_D = & \mathbb{E}_{(P,Q) \sim \mu} \mathbb{E}_{x \sim P, y \sim Q} \\ & [-D_{sr,D}(P, Q, x) - D_{sr,D}(Q, P, y) \\ & + \frac{1}{4\epsilon}(D_{sr,D}(P, Q, x) + D_{sr,D}(Q, P, y) - c(x, y))_+^2]. \end{aligned} \quad (5.24)$$

The second step in [123] is to train f to minimize

$$\mathbb{E}_{x \sim P, y \sim Q} [\frac{1}{2\epsilon}c(y, f(x))(u(x) + v(y) - c(x, y))_+], \quad (5.25)$$

so that the minimizer f^* is the barycentric projection of the regularized optimal transport, which can be viewed as an approximation of the optimal transport map from P to Q . We will use $D_{sr,G}(P, Q, x)$ to replace $f(x)$, and the loss function for $D_{sr,G}$ writes as

$$L_G = \mathbb{E}_{(P,Q) \sim \mu} \mathbb{E}_{x \sim P, y \sim Q} \left[\frac{1}{2\epsilon} c(y, D_{sr,G}(P, Q, x)) \right. \\ \left. (D_{sr,D}(P, Q, x) + D_{sr,D}(Q, P, y) - c(x, y))_+ \right], \quad (5.26)$$

Note that we take the expectation over (P, Q) in Equation 5.24 and 5.26, so that in the end of training, $D_{sr,G}(P, Q, x)$ will approximate the optimal transport map from P to Q for various (P, Q) pairs.

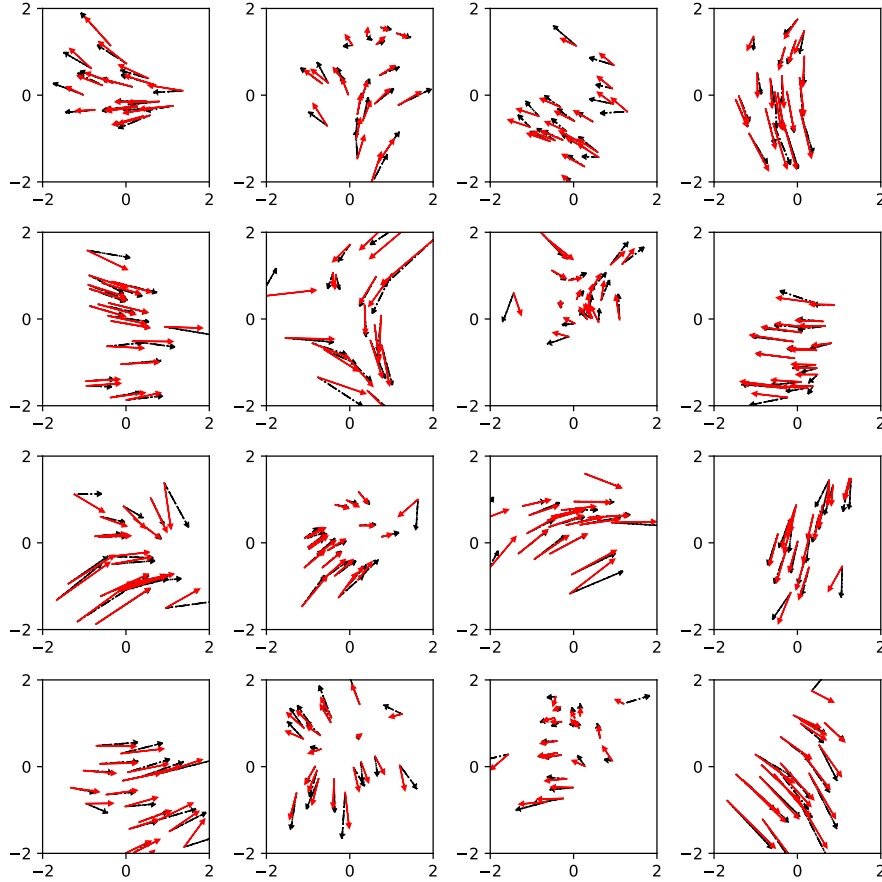


Figure 5.9: Results of the surrogate model for optimal transport maps between 16 pairs of Gaussian distributions. The red arrows represent the barycentric projection maps given by the surrogate model $D_{sr,G}$, while the black arrows represent the reference optimal transport maps from linear programming.

[123] propose to train u and v until convergence, and then train f . But we found that training $D_{sr,D}$ and $D_{sr,G}$ iteratively after a warming-up training of $D_{sr,D}$ also works. We train and test with P, Q independently sampled from the 2D μ_{train} in Section 5.5.3, and show the results after 200,000 iterations with 10,000 warming-up steps in Figure 5.9. The reference map is the empirical optimal transport map between 1000 samples, calculated by the POT package [33] using linear programming. One can see that the surrogate model provides a similar transport map as the reference.

5.7 Summary of Chapter

In this chapter we propose measure-conditional discriminators as a plug-and-play module for a variety of GANs. Conditioned on the generated distributions so that the target optimum is stationary during training, the measure-conditional discriminators are more robust w.r.t. the GAN losses, discriminator/generator iteration ratios, and optimizer hyperparameters, compared with the vanilla ones. A variant of the measure-conditional discriminator can also be employed in the scenarios with multiple target distributions, or as surrogate models of statistical distances.

Note that even outdated generated distributions can be used to training the measure-conditional discriminator. It is worth to study if training the discriminator with generated distributions from a replay buffer, which contains the generated distributions in history just as in off-policy reinforcement learning, can further improve the performance. Also, as a proof of concept, the neural network architectures in this chapter have a very straight-forward form, leaving a lot of room

for improvements. For example, different weights can be assigned to the samples of the input distributions, which is similar to importance sampling in statistics or the attention mechanism in deep learning. Moreover, the statistical distance surrogate can be applied as a building block in replacement of direct calculation in more complicated models.

CHAPTER SIX

Bayesian Uncertainty Quantification for Partial Differential Equations with Physics-Informed GANs

6.1 Introduction

The state-of-the-art in data-driven modeling has advanced significantly in applications across different fields [61, 113, 78, 17, 10], due to the rapid development of machine learning and the explosive growth of available data collected from different sensors (e.g., satellites, cameras, etc.). In general, purely data-driven methods require a large amount of data in order to get accurate results [110]. As a powerful alternative, recently the data-driven solvers for partial differential equations (PDEs) have drawn an increasing attention due to their capability to encode the underlying physical laws in the form of PDEs and give relatively accurate predictions for the unknown terms with limited data. In the first case we need “big data” while in the second case we can learn from “small data” as we explicitly utilize the physical laws or more broadly a parameterization of the physics.

Two typical approaches are the Gaussian processes regression (GPR) for PDEs [104], and the physics-informed neural networks (PINNs) [110, 73]. Built upon the Bayesian framework with built-in mechanism for uncertainty quantification, GPR is one of the most popular data-driven methods. However, vanilla GPR has difficulties in handling the nonlinearities when applied to solve PDEs, leading to restricted applications. On the other hand, PINNs have shown effectiveness in both forward and inverse problems for a wide range of PDEs [149, 140, 82, 83, 79]. However, PINNs are not equipped with built-in uncertainty quantification, which may restrict their applications, especially for scenarios where the data are noisy.

In previous work, the physics-informed generative adversarial networks were employed to quantify parametric uncertainty [140], and also polynomial chaos expansions in conjunction with dropout were utilized to quantify total uncer-

tainty [149]. In addition, approaches using the Bayesian inference for quantifying uncertainties of PDE problems have also been developed recently [67, 135, 136]. Note that (1) the methods developed in [67, 135, 136] focus on solving inverse PDE problems, and (2) the boundary conditions as well as the source terms are assumed to be known in [67, 135, 136]. However, we may only have access to sparse and noisy measurements on the boundary conditions and the source terms in real-world applications.

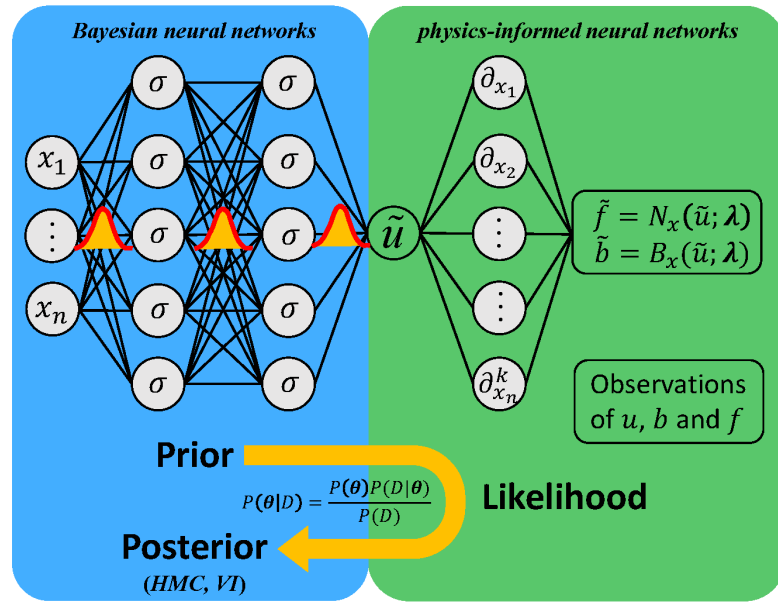


Figure 6.1: Schematic of the Bayesian physics-informed neural network (B-PINN). $P(\theta)$ is the prior for hyperparameters as well as the unknown terms in PDEs, $P(\mathcal{D}|\theta)$ represents the likelihood of observations (e.g., u, b, f), and $P(\theta|\mathcal{D})$ is the posterior. The blue panel represents the Bayesian neural network while the green panel represents the physics-informed part.

In the first part of this chapter, we propose Bayesian physics-informed neural networks (B-PINNs) to solve linear or nonlinear PDEs with noisy data for both forward and inverse problems, see Figure 6.1. We note that all the information of boundary conditions/sources terms comes from noisy measurements in the present approach. The uncertainties arising from the scattered noisy data could be naturally quantified due to the Bayesian framework [75]. B-PINNs consist of two parts: a parameterized surrogate model, i.e., a Bayesian neural network (BNN)

with prior for the unknown terms in a PDE, and an approach for sampling the posterior distributions of the parameters in the surrogate model. In particular, we employ the Hamiltonian Monte Carlo (HMC) [90, 89] or the variational inference (VI) [39, 14] for estimation of the posterior distributions. We will validate the proposed B-PINNs method and conduct a systematic comparison for both the forward and inverse PDE problems given noisy data.

The choice of priors for B-PINNs may have a significant influence on the posterior predictions especially in the cases with small data. In the second part of this chapter, we propose to use physics-informed GANs to learn functional priors, which can then be used in B-PINNs. We will show that physics-informed GANs can learn informative priors that benefit the uncertainty quantification. We will also discuss why sometimes the learned prior fails for posterior estimation, and how to make improvements.

6.2 B-PINNs: Bayesian Physics-Informed Neural Networks

We consider a general partial differential equation (PDE) of the form

$$\begin{aligned} \mathcal{N}_x(u; \lambda) &= f, & x \in D, \\ \mathcal{B}_x(u; \lambda) &= b, & x \in \Gamma, \end{aligned} \tag{6.1}$$

where $\mathcal{N}_x$ is a general differential operator, D is the d -dimensional physical domain, $u = u(x)$ is the solution of the PDE, and λ is the vector of parameters in the PDE. Also, $f = f(x)$ is the forcing term, and $\mathcal{B}_x$ is the boundary condition operator

acting on the domain boundary Γ . In forward problems λ is prescribed, and hence our goal is to infer the distribution of u at any x . In inverse problems, λ is also to be inferred from the data.

We consider the scenario where our available dataset $\mathcal{D}$ are scattered noisy measurements of u , f and b from sensors:

$$\mathcal{D} = \mathcal{D}_u \cup \mathcal{D}_f \cup \mathcal{D}_b, \quad (6.2)$$

where $\mathcal{D}_u = \{(\mathbf{x}_u^{(i)}, \bar{u}^{(i)})\}_{i=1}^{N_u}$, $\mathcal{D}_f = \{(\mathbf{x}_f^{(i)}, \bar{f}^{(i)})\}_{i=1}^{N_f}$, $\mathcal{D}_b = \{(\mathbf{x}_b^{(i)}, \bar{b}^{(i)})\}_{i=1}^{N_b}$. We assume that the measurements are independently Gaussian distributed centered at the hidden real value, i.e.,

$$\begin{aligned} \bar{u}^{(i)} &= u(\mathbf{x}_u^{(i)}) + \epsilon_u^{(i)}, \quad i = 1, 2, \dots, N_u, \\ \bar{f}^{(i)} &= f(\mathbf{x}_f^{(i)}) + \epsilon_f^{(i)}, \quad i = 1, 2, \dots, N_f, \\ \bar{b}^{(i)} &= b(\mathbf{x}_b^{(i)}) + \epsilon_b^{(i)}, \quad i = 1, 2, \dots, N_b, \end{aligned} \quad (6.3)$$

where $\epsilon_u^{(i)}$, $\epsilon_f^{(i)}$ and $\epsilon_b^{(i)}$ are independent Gaussian noises with zero mean. We also assume that the fidelity of each sensor is known, i.e., the standard deviations of $\epsilon_u^{(i)}$, $\epsilon_f^{(i)}$ and $\epsilon_b^{(i)}$ are known to be $\sigma_u^{(i)}$, $\sigma_f^{(i)}$ and $\sigma_b^{(i)}$, respectively. Note that the size of the noise could be different among measurements of different terms, and even between measurements of the same terms in the PDE.

We first consider the forward problem setup. The Bayesian framework starts from representing u with a surrogate model $\tilde{u}(x; \theta)$, where θ is the vector of parameters in the surrogate model with a prior distribution $P(\theta)$. Consequently, f and b are represented by:

$$\tilde{f}(x; \theta) = \mathcal{N}_x(\tilde{u}(x; \theta); \lambda), \quad \tilde{b}(x; \theta) = \mathcal{B}_x(\tilde{u}(x; \theta); \lambda). \quad (6.4)$$

Then, the likelihood can be calculated as:

$$\begin{aligned}
P(\mathcal{D}|\boldsymbol{\theta}) &= P(\mathcal{D}_u|\boldsymbol{\theta})P(\mathcal{D}_f|\boldsymbol{\theta})P(\mathcal{D}_b|\boldsymbol{\theta}), \\
P(\mathcal{D}_u|\boldsymbol{\theta}) &= \prod_{i=1}^{N_u} \frac{1}{\sqrt{2\pi\sigma_u^{(i)2}}} \exp\left(-\frac{(\tilde{u}(\mathbf{x}_u^{(i)}; \boldsymbol{\theta}) - \bar{u}^{(i)})^2}{2\sigma_u^{(i)2}}\right), \\
P(\mathcal{D}_f|\boldsymbol{\theta}) &= \prod_{i=1}^{N_f} \frac{1}{\sqrt{2\pi\sigma_f^{(i)2}}} \exp\left(-\frac{(\tilde{f}(\mathbf{x}_f^{(i)}; \boldsymbol{\theta}) - \bar{f}^{(i)})^2}{2\sigma_f^{(i)2}}\right), \\
P(\mathcal{D}_b|\boldsymbol{\theta}) &= \prod_{i=1}^{N_b} \frac{1}{\sqrt{2\pi\sigma_b^{(i)2}}} \exp\left(-\frac{(\tilde{b}(\mathbf{x}_b^{(i)}; \boldsymbol{\theta}) - \bar{b}^{(i)})^2}{2\sigma_b^{(i)2}}\right).
\end{aligned} \tag{6.5}$$

Finally, the posterior is obtained from Bayes' theorem:

$$P(\boldsymbol{\theta}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta})}{P(\mathcal{D})} \simeq P(\mathcal{D}|\boldsymbol{\theta})P(\boldsymbol{\theta}), \tag{6.6}$$

where “ $\simeq$ ” represents equality up to a constant. Usually the calculation of $P(\mathcal{D})$ is analytically intractable, thus in practice we only have an unnormalized expression of $P(\boldsymbol{\theta}|\mathcal{D})$. To give a posterior u at any $\mathbf{x}$, we can sample from $P(\boldsymbol{\theta}|\mathcal{D})$, denoted as $\{\boldsymbol{\theta}^{(i)}\}_{i=1}^M$, and then obtain statistics from samples $\{\tilde{u}(\mathbf{x}; \boldsymbol{\theta}^{(i)})\}_{i=1}^M$. We focus mostly on the mean and standard deviation of $\{\tilde{u}(\mathbf{x}; \boldsymbol{\theta}^{(i)})\}_{i=1}^M$, since the former represents the prediction of $u(\mathbf{x})$ while the latter quantifies the uncertainty.

In the case of inverse problems, we can build the surrogate model for u in the same way as above. However, apart from $\boldsymbol{\theta}$, we also need to assign a prior distribution for $\boldsymbol{\lambda}$, which could be independent of $P(\boldsymbol{\theta})$. The likelihood is the same as in Equation 6.5, except that $P(\mathcal{D}|\boldsymbol{\theta})$, $P(\mathcal{D}_u|\boldsymbol{\theta})$, $P(\mathcal{D}_f|\boldsymbol{\theta})$ and $P(\mathcal{D}_b|\boldsymbol{\theta})$ should be replaced by $P(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{\lambda})$, $P(\mathcal{D}_u|\boldsymbol{\theta}, \boldsymbol{\lambda})$, $P(\mathcal{D}_f|\boldsymbol{\theta}, \boldsymbol{\lambda})$ and $P(\mathcal{D}_b|\boldsymbol{\theta}, \boldsymbol{\lambda})$, respectively. Consequently,

we should calculate the joint posterior of $[\boldsymbol{\theta}, \boldsymbol{\lambda}]$ as

$$P(\boldsymbol{\theta}, \boldsymbol{\lambda}|\mathcal{D}) = \frac{P(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{\lambda})P(\boldsymbol{\theta}, \boldsymbol{\lambda})}{P(\mathcal{D})} \simeq P(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{\lambda})P(\boldsymbol{\theta}, \boldsymbol{\lambda}) = P(\mathcal{D}|\boldsymbol{\theta}, \boldsymbol{\lambda})P(\boldsymbol{\theta})P(\boldsymbol{\lambda}), \quad (6.7)$$

where the last equality comes from the fact that the priors for $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$ are independent.

The parameter $\boldsymbol{\lambda}$ in the PDE is a vector in the above problem setup, however, we remark that the same framework could be applied in the cases where the parameter is a field or fields depending on $\boldsymbol{x}$, by representing the parameter with another surrogate model, e.g., a neural network.

We remark that all the information of boundary conditions/source terms comes from noisy measurements in the present approach. If we have the analytical expression for these terms, in order to generate a proper synthetic data set for our tests we can select some points and get corresponding observations with very small noise in our framework.

Since the forward problems and inverse problems are formulated in the same framework, in the following we will use $\boldsymbol{\theta}$ to represent the vector of all the unknown parameters in the surrogate models for the solutions and parameters. We denote the dimension of $\boldsymbol{\theta}$, i.e., the number of unknown parameters, as $d_{\boldsymbol{\theta}}$.

6.2.1 Prior for Bayesian Physics-Informed Neural Networks

We consider a fully-connected neural network with $L \geq 1$ hidden layers as the surrogate model, see Figure 6.1. Let us denote the input of the neural network as

$\mathbf{x} \in \mathbb{R}^{N_x}$, the output of the neural network as $\tilde{u} \in \mathbb{R}$, and the l -th hidden layer as $\mathbf{z}_l \in \mathbb{R}^{N_l}$ for $l = 1, 2, \dots, N$. Then

$$\begin{aligned} \mathbf{z}_l &= \phi(\mathbf{w}_{l-1} \mathbf{z}_{l-1} + \mathbf{b}_{l-1}), \quad l = 1, 2, \dots, L, \\ \tilde{u} &= \mathbf{w}_L \mathbf{z}_L + \mathbf{b}_L, \end{aligned} \quad (6.8)$$

where $\mathbf{w}_l \in \mathbb{R}^{N_{l+1} \times N_l}$ are the weight matrices, $\mathbf{b}_l \in \mathbb{R}^{N_{l+1}}$ are the bias vectors, ϕ is the nonlinear activation function, which is the hyperbolic tangent function in the present study, and $\mathbf{z}_0 = \mathbf{x}$, $N_0 = N_x$, and $N_{L+1} = 1$ for the convenience of notation. When using a neural network as a surrogate model, the unknown parameters θ are the concatenation of all the weight matrices and bias vectors.

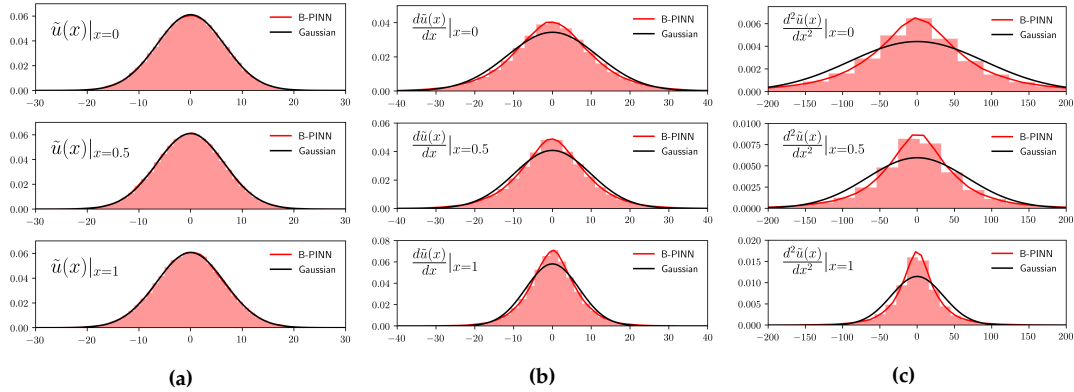


Figure 6.2: Comparison between the B-PINNs' prior distributions and Gaussian distributions. The red lines and histograms represent the density of B-PINNs' outputs (a) $\tilde{u}(x)$, (b) $d\tilde{u}(x)/dx$, and (c) $d^2\tilde{u}(x)/dx^2$ at $x = 0, 0.5$, and 1 . The black lines are the density functions of the corresponding Gaussian distributions with zero mean and the same standard deviations as the B-PINNs' outputs.

In the application of Bayesian neural networks, a commonly used prior for θ is that each component of θ is an independent Gaussian distribution with zero mean, and the entries of $\mathbf{w}_l$ and $\mathbf{b}_l$ have the variances $\sigma_{w,l}$ and $\sigma_{b,l}$, respectively, for $l = 0, 1, \dots, L$ [90, 89]. In this case, it can be shown that the prior of the function $\tilde{u}(x)$ is actually a Gaussian process as the width of hidden layers goes to infinity with $\sqrt{N_l}\sigma_{w,l}$ fixed for $l = 1, 2, \dots, L$ [89, 64, 95].

However, we remark that due to the finite width of the neural networks, the

derivatives of $\tilde{u}$ could be far from Gaussian processes. For example, in Figure 6.2 we compare the prior of B-PINNs with Gaussian distributions, where $N_x = 1$, $L = 2$, $N_1 = N_2 = 50$, $\sigma_{b,l} = \sigma_{w,l} = 1$, for $l = 0, 1, 2$. We can see that although the priors of $\tilde{u}(x)$ at various x match the corresponding Gaussian distributions, the priors of $d\tilde{u}(x)/dx$ and $d^2\tilde{u}(x)/dx^2$ are not close to the Gaussian distributions.

6.2.2 Posterior Sampling Approaches for Bayesian Physics-Informed Neural Networks

In this subsection, we introduce two approaches to sample from the posterior distribution of the parameters in B-PINNs: the Hamiltonian Monte Carlo (HMC) method and the variational inference (VI) method.

Hamiltonian Monte Carlo (HMC) Method

Hamiltonian Monte Carlo, which is known as a golden approach for sampling from posterior distributions, is an efficient Markov Chain Monte Carlo (MCMC) method based on the Hamiltonian dynamics [90, 89, 12]. In this approach, we first simulate the Hamiltonian dynamics using numerical integration, which is then corrected by a Metropolis-Hastings acceptance step.

Suppose the target posterior distribution for θ given a certain number of observations $\mathcal{D}$ is defined as

$$P(\theta|\mathcal{D}) \simeq \exp(-U(\theta)), \quad (6.9)$$

where

$$U(\theta) = -\ln P(\mathcal{D}|\theta) - \ln P(\theta). \quad (6.10)$$

To sample from the posterior, HMC first introduces an auxiliary momentum variable $\mathbf{r}$ to construct a Hamiltonian system

$$H(\theta, \mathbf{r}) = U(\theta) + \frac{1}{2} \mathbf{r}^T \mathbf{M}^{-1} \mathbf{r}, \quad (6.11)$$

where $\mathbf{M}$ is a mass matrix, which is often set to be identity matrix, $\mathbf{I}$. Then HMC generates samples from a joint distribution of $(\theta, \mathbf{r})$ as follows

$$\pi(\theta, \mathbf{r}) \sim \exp\left(-U(\theta) - \frac{1}{2} \mathbf{r}^T \mathbf{M}^{-1} \mathbf{r}\right). \quad (6.12)$$

As we simply discard the $\mathbf{r}$ samples, the θ samples have marginal distribution $P(\theta|\mathcal{D})$.

Specifically, the samples are generated from the following Hamiltonian dynamics

$$d\theta = \mathbf{M}^{-1} \mathbf{r} dt, \quad (6.13a)$$

$$d\mathbf{r} = -\nabla U(\theta) dt. \quad (6.13b)$$

We use the leapfrog method to discretize Equation 6.13. Furthermore, to reduce the discretization error, we employ a Metropolis-Hastings step. The details for implementing the HMC method are displayed in Algorithm 4.

Algorithm 4 Hamiltonian Monte Carlo

Require: initial states for θ^{t_0} and time step size δt .

```

1: for  $k = 1, 2, \dots, N$  do
2:   Sample  $\mathbf{r}^{t_{k-1}}$  from  $\mathcal{N}(0, M)$ ,
3:    $(\theta_0, \mathbf{r}_0) \leftarrow (\theta^{t_{k-1}}, \mathbf{r}^{t_{k-1}})$ .
4:   for  $i = 0, 1, \dots, (L - 1)$  do
5:      $\mathbf{r}_i \leftarrow \mathbf{r}_i - \frac{\delta t}{2} \nabla U(\theta_i)$ ,
6:      $\theta_{i+1} \leftarrow \theta_i + \delta t M^{-1} \mathbf{r}_i$ ,
7:      $\mathbf{r}_{i+1} \leftarrow \mathbf{r}_i - \frac{\delta t}{2} \nabla U(\theta_{i+1})$ ,
8:   end for
9:   Metropolis-Hastings step:
10:  Sample  $p$  from Uniform[0, 1],
11:   $\alpha \leftarrow \min\{1, \exp(H(\theta_L, \mathbf{r}_L) - H(\theta^{t_{k-1}}, \mathbf{r}^{t_{k-1}}))\}$ .
12:  if  $p \geq \alpha$  then
13:     $\theta^{t_k} \leftarrow \theta_L$ ,
14:  else
15:     $\theta^{t_k} \leftarrow \theta^{t_{k-1}}$ .
16:  end if
17: end for
18: Calculate  $\{\tilde{u}(\mathbf{x}, \theta^{t_{N+1-j}})\}_{j=1}^M$  as samples of  $u(\mathbf{x})$ , similarly for other terms.

```

Variational Inference (VI) Method

In the variational learning, the posterior density of the unknown parameter vector $\theta = (\theta_1, \theta_2, \dots, \theta_{d_\theta})$, i.e., $P(\theta|\mathcal{D})$, is approximated by another density function $Q(\theta; \zeta)$ parameterized by ζ .

While there are various choices for Q (the normalizing flow model introduced in Section 6.2.6 is also one of the choices), a common choice in the deep learning community is the mean-field Gaussian approximation [34, 112, 144], i.e. Q is a factorizable Gaussian distribution as follows:

$$Q(\boldsymbol{\theta}; \boldsymbol{\zeta}) = \prod_{i=1}^{d_{\boldsymbol{\theta}}} q(\theta_i; \zeta_{\mu,i}, \zeta_{\rho,i}), \quad (6.14)$$

where $\boldsymbol{\zeta} = (\boldsymbol{\zeta}_{\mu}, \boldsymbol{\zeta}_{\rho})$, $\boldsymbol{\zeta}_{\mu} = (\zeta_{\mu,1}, \zeta_{\mu,2}, \dots, \zeta_{\mu,d_{\boldsymbol{\theta}}})$, $\boldsymbol{\zeta}_{\rho} = (\zeta_{\rho,1}, \zeta_{\rho,2}, \dots, \zeta_{\rho,d_{\boldsymbol{\theta}}})$, and $q(\theta_i; \zeta_{\mu,i}, \zeta_{\rho,i})$ is the density of the one-dimensional Gaussian distribution with mean $\zeta_{\mu,i}$ and standard deviation $\ln(1 + \exp(\zeta_{\rho,i}))$. In this approach, we can tune $\boldsymbol{\zeta}$ to minimize

$$D_{KL}(Q(\boldsymbol{\theta}; \boldsymbol{\zeta}) \| P(\boldsymbol{\theta} | \mathcal{D})) \simeq \mathbb{E}_{\boldsymbol{\theta} \sim Q} [\ln Q(\boldsymbol{\theta}; \boldsymbol{\zeta}) - \ln P(\boldsymbol{\theta}) - \ln P(\mathcal{D} | \boldsymbol{\theta})], \quad (6.15)$$

where D_{KL} denotes the Kullback-Leibler (KL) divergence.

Here we employ the Adam optimizer [54] to train $\boldsymbol{\zeta}$. The detailed algorithm is given in Algorithm 5. We refer the readers to [14] for more details. As a clarification, “VI” in the rest of the chapter indicates the version of mean field Gaussian approximation with KL divergence minimization introduced in this section.

Algorithm 5 Variational inference

Require: an initial state for ζ .

- 1: **for** $k = 1, 2, \dots, N$ **do**
 - 2: Sample $\{z^{(j)}\}_{j=1}^{N_z}$ independently from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{d_\theta})$.
 - 3: $\theta^{(j)} \leftarrow \zeta_\mu + \ln(1 + \exp(\zeta_\rho)) \odot z^{(j)}$, $j = 1, 2, \dots, N_z$, $\odot$ denotes element-wise product.
 - 4: $L(\zeta) \leftarrow \frac{1}{N_z} \sum_{j=1}^{N_z} [\ln Q(\theta^{(j)}; \zeta) - \ln P(\theta^{(j)}) - \ln P(\mathcal{D}|\theta^{(j)})]$.
 - 5: Update ζ with gradient $\nabla_\zeta L(\zeta)$ using Adam optimizer.
 - 6: **end for**
 - 7: Sample $\{z^{(j)}\}_{j=1}^M$ independently from $\mathcal{N}(\mathbf{0}, \mathbf{I}_{d_\theta})$.
 - 8: $\theta^{(j)} \leftarrow \zeta_\mu + \ln(1 + \exp(\zeta_\rho)) \odot z^{(j)}$, $j = 1, 2, \dots, M$.
 - 9: Calculate $\{\tilde{u}(x, \theta^{(j)})\}_{j=1}^M$ as samples of $u(x)$, similarly for other terms.
-

6.2.3 Computational Results

In this section we present a systematic comparison among the B-PINNs with different posterior sampling methods, i.e., HMC (B-PINN-HMC) and VI (B-PINN-VI), as well as the dropout [149, 35] for 1D function approximation, and 1D/2D forward/inverse PDE problems.

In all the cases, we employ a neural network with 2 hidden layers, each with width of 50, for B-PINNs. The prior for θ is set as independent standard Gaussian distribution for each component. Such size of the neural network and the prior distribution are inherited from [144]. In the 1D case, the covariance function for $\tilde{u}$ is shown in Figure 6.15b (Section 6.2.6). In HMC, the mass matrix is set to the identity matrix, i.e., $M = \mathbf{I}$ [144], the leapfrog step is set to $L = 50\delta t$, the initial

time step is $\delta t = 0.1$, the burn-in steps are set to 2,000, and the total number of samples is 15,000. In VI, the Adam optimizer is employed for training, and the total number of training steps is $N = 200,000$ with batch size $N_z = 5$. The hyperparameters for Adam optimizer are set as $l = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. In the dropout method, we randomly drop a certain number of neurons with a predefined probability (i.e., dropout rate) at each training step [35]. The number of the training steps is 200,000. The hyperparameters for the Adam optimizer are set as $l = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$. To quantify the uncertainty, the strategy used in [149] is also employed here, i.e., after finishing the training we run the forward propagation of DNNs M times with the same dropout rate as in training, and then compute the mean and standard deviation based on the samples. When estimating the means and standard deviations, we set the number of samples $M = 10,000$ for all the methods in each test case.

Function Regression

In this section, we test the posterior sampling methods introduced above on a function regression task. In this task, the B-PINNs is reduced to a BNN. The framework is the same as that for solving PDEs, except that our data set only involves the unknown function $u(x)$, thus the likelihood is $P(\mathcal{D}|\theta) = P(\mathcal{D}_u|\theta)$. The test function is expressed as

$$u(x) = \sin^3(6x), \quad x \in [-1, 1], \quad (6.16)$$

and we use 32 training points placed in $[-0.8, -0.2] \cup [0.2, 0.8]$, with observation noise $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$.

We note that the prior for BNNs is a Gaussian process with zero mean as the width of each hidden layer goes to infinity [89, 64, 95]. In our case where the width is 50, we could see from Figure 6.2a that the prior at several single points matches well the Gaussian distribution. We thus view the prior as a Gaussian process approximately. Although the analytical expression for the kernel cannot be calculated, it could be estimated from independent samples of neural network functions (100,000 samples), as shown in Figure 6.15b. Therefore, GPR could be applied and provide reference solutions for the posterior estimations, denoted as BNN-GPR. Note that such reference solution is not analytical and the errors could come from the Gaussian process assumption for neural networks with finite width as well as the empirical estimation of the kernel. The results are shown in Figure 6.3. We can see from Figure 6.3a-6.3b that (1) the predictive means are observed to be similar to the exact function $u(x)$, and the predicted standard deviations from these two methods are quite similar, and (2) the standard deviations (i.e., uncertainty) become larger at the regions with fewer training data, i.e., $x \in [-1, -0.8] \cup [-0.2, 0.2] \cup [0.8, 1]$, which is reflected in the growing uncertainty due to lack of data.

Note that the dimension of the unknown parameters θ is 2701 in our case using BNNs as prior. Despite the high dimensionality, HMC also provides posterior estimation (Figure 6.3b) similar to the reference solution (Figure 6.3a) in this case, which shows its effectiveness in sampling from high dimensional distributions. As for the BNN-VI, the uncertainty at $x \in [-0.2, 0.2]$ is observed to be larger, while the uncertainty around $x \in [-1, -0.8] \cup [0.8, 1]$ is underestimated. Such results indicate that VI is not as accurate as HMC in posterior estimation, which could be attributed to the fact that the samples are actually drawn from Q in Equation 6.14, which is limited to a family that could be too small to give a reasonable

approximation of the posterior distribution.

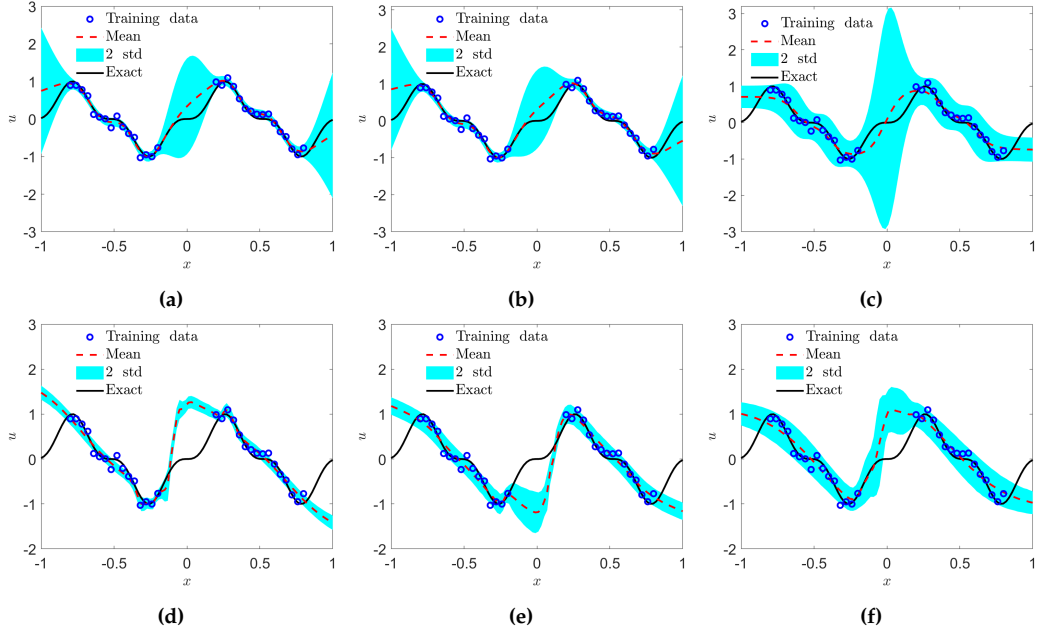


Figure 6.3: Function approximation and comparison among different approaches. (a) BNN-GPR, (b) BNN-HMC, (c) BNN-VI, (d) Dropout with drop rate 1%, (e) Dropout with drop rate 5%, (f) Dropout with drop rate 20% (4 hidden layers with 100 neurons per layer).

Since both the architecture of the DNNs and the dropout rate are known to have strong effects on the certainty quantification, we test there different cases: (1) 2 hidden layers with 50 neurons and dropout rate 0.01, (2) 2 hidden layers with 50 neurons and dropout rate 0.05, and (3) 4 hidden layers with 100 neurons and dropout rate 0.2. We use the same dropout rate at the prediction step as that used in the training process to quantify the uncertainty. As we can see from Figure 6.3d-6.3f, the uncertainties appear to be uniform for all the $x \in [-1, 1]$, which is not reasonable at all in this case. Furthermore, the predictive mean is quite different from the exact function $u(x)$ at the gap data zones, and the differences are far beyond the two standard deviations.

We further test the performance of the BNN-HMC for cases with larger noise scales, i.e., (a) $\epsilon_u \sim \mathcal{N}(0, 0.3^2)$, and (b) $\epsilon_u \sim \mathcal{N}(0, 0.5^2)$. As shown in Figure 6.4,

the results from BNN-HMC agree with those from BNN-GP. In addition, the standard deviations (i.e., uncertainty) become larger at the regions with fewer training data, i.e., $x \in [-1, -0.8] \cup [-0.2, 0.2] \cup [0.8, 1]$, which is similar as results in Figure 6.3. However, the predictive means do not fit the exact solutions in $x \in [-0.8, -0.2] \cup [0.2, 0.8]$ as well as the results in Figure 6.3a-6.3b due to the larger noise.

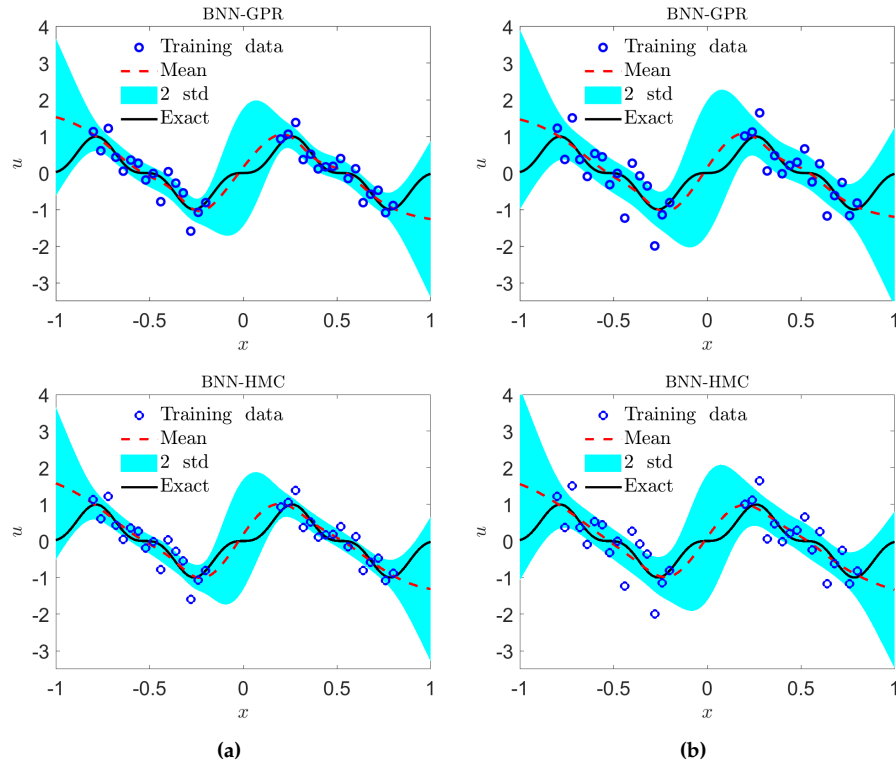


Figure 6.4: Function approximations for cases with larger noise. (a) $\epsilon_u \sim \mathcal{N}(0, 0.3^2)$. (b) $\epsilon_u \sim \mathcal{N}(0, 0.5^2)$.

It is worth mentioning that while the priors for the BNNs with infinite width have been well studied [89, 64, 95], the choice of priors for BNNs with finite width remains an open question. More discussion about the influence of architecture of the neural networks as well as the prior distributions of the parameters will be presented in Section 6.2.6. To keep consistent, we will employ the same architecture of DNNs with the dropout as the BNNs, i.e., 2 hidden layers with 50 neurons

in the following.

Forward Problem: 1D Poisson Equation

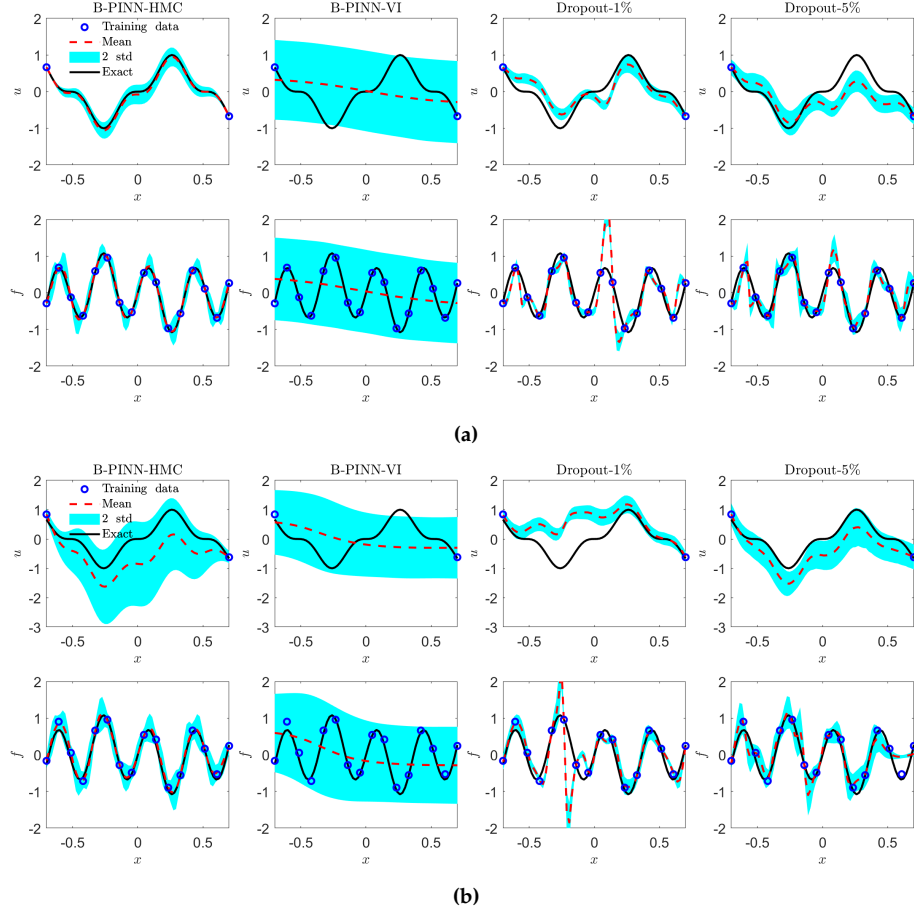


Figure 6.5: 1D linear Poisson equation - forward problem: predicted u and f from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

We consider the following linear Poisson equation:

$$\lambda \partial_x^2 u = f, x \in [-0.7, 0.7], \quad (6.17)$$

where $\lambda = 0.01$. The solution for u is $u = \sin^3(6x)$, and f can be derived from Equation 6.17. Here we assume that the exact expression of f is unknown, but instead

we have 16 sensors for f , which are equidistantly distributed in $x \in [-0.7, 0.7]$. Furthermore, we have two sensors at $x = -0.7$ and 0.7 to provide the left/right Dirichlet boundary conditions for u . We assume that all the measurements from the sensors are noisy, and we consider the following two different cases: (1) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$, and (2) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$. The results of B-PINN-HMC, B-PINN-VI and dropout, are illustrated in Figure 6.5.

We see that the B-PINN-HMC gives reasonable posterior estimation in that (1) the error between the means and the exact solution is mostly bounded by the two standard deviations, (2) the standard deviation increases with increasing noise scale. However, B-PINN-VI completely failed to provide predictions to the solution u . Finally, the prediction given by dropout seems to match the data, but the predictive means for u and f differ from the exact solutions. We also note that the uncertainties provided by the dropout are not reasonable, e.g., (1) there is no significant differences between uncertainties for u in the two cases with different noise scale, and (2) a large part of the exact solution does not lie in the two standard deviation confidence intervals.

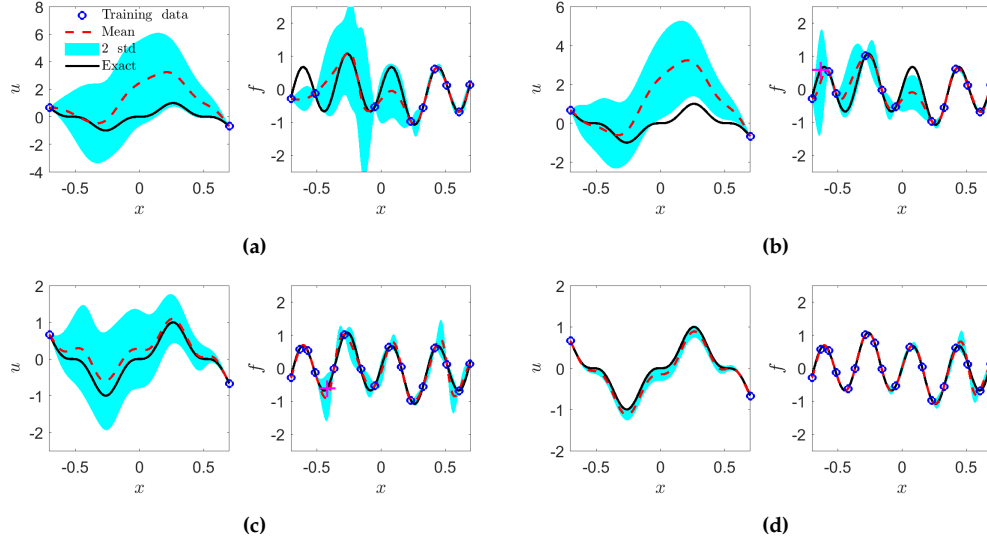


Figure 6.6: 1D linear Poisson equation: Application of active learning. (a) $N = 8$, (b) $N = 12$, (c) $N = 16$, (d) $N = 18$. N represents the number of training samples for f . Blue circle: Training samples at current step. Magenta plus: Added training sample for the next step.

We proceed to consider the case in which we have a very limited number of sensors for f . In particular, we assume that we only have 8 sensors for f , which are randomly distributed in $x \in [-0.7, 0.7]$. The predicted u and f are illustrated in Figure 6.6a. Note that here we only present the results from B-PINN-HMC since it is the most accurate among all the approaches tested in this study. As shown, the predicted means for u and f are quite different from the exact solutions. This is reasonable since 8 point are not enough for this complex f . Meanwhile, the predictive uncertainties for both u and f are observed to be much larger than in Figure 6.5, which suggests that the predicted u and f may be inaccurate given such a limited number of observations on f . To improve the accuracy and reduce the uncertainty, we apply active learning as in [103]. Specifically, we can add more training samples at the location, which has the largest uncertainty. As shown in Figure 6.6, the predicted means for u and f approach the exact solutions as more data of f have been collected, consistent with the shrinking uncertainty.

Forward Problem: 1D Flow Through Porous Media with Boundary Layer

We now consider flows through a horizontal channel filled with a uniform porous medium governed by the following equation:

$$-\frac{\nu_e}{\phi} \partial_x^2 u + \frac{\nu u}{K} = g, \quad x \in [0, 1], \quad (6.18)$$

where g denotes the external force, ν_e is the effective viscosity, ϕ is the porosity, ν is the fluid viscosity, K is the permeability; the x direction is orthogonal to the channel, and $u(x)$ is the velocity along the channel. No-slip boundary conditions are imposed on the upper/bottom walls, i.e., $u(x) = 0$ at $x = 0$ and 1 . The exact solution for u is

$$u = \frac{gK}{\nu} \left[1 - \frac{\cosh(r(x - H/2))}{\cosh(rH/2)} \right], \quad (6.19)$$

where $r = \sqrt{\frac{\nu\phi}{\nu_e K}}$. For the present case, $\nu_e = \nu = 10^{-3}$, $\phi = 0.4$, $K = 10^{-3}$. We set the hidden unknown external force $g = 1$ for which we collect data from sensors.

Here we assume that we have 16 sensors for g equidistantly placed in $x \in [0, 1]$. In addition, two sensors for u are placed at $x = 0$ and 1 for Dirichlet boundary conditions. Two different scales of Gaussian noise in the measurements are considered, i.e., (1) $\epsilon_g \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$, and (2) $\epsilon_g \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$, which is the same as in Section 6.2.3. The results of different methods are presented in Figure 6.7.

The B-PINN-HMC provides reasonable predictions for both means and uncer-

tainties, even for the steep velocity profile in the boundary layer. The dropout is able to provide accurate predictions for the means, but the predictive uncertainties are not reasonable, i.e., the noise scale has little influence on the predicted standard deviations in the two dropout cases. Meanwhile, the B-PINN-VI can provide accurate means for the case with $\epsilon_g \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$, but it failed for the case with larger noise.

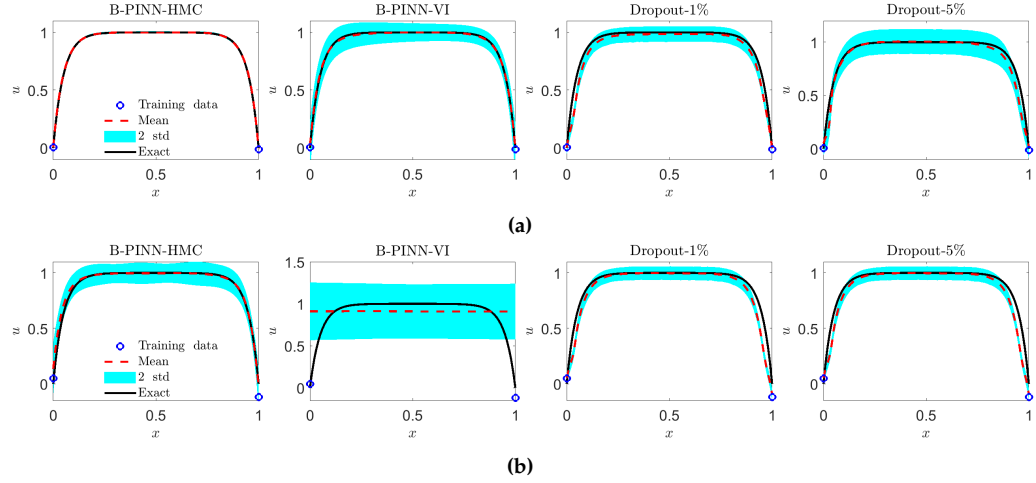


Figure 6.7: 1D flow through porous media with boundary layer: predicted u from different methods with two data noise scales. (a) $\epsilon_g \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_g \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

Forward Problem: 1D Nonlinear Poisson Equation

Here we consider the following 1D nonlinear PDE

$$\lambda \partial_x^2 u + k \tanh(u) = f, \quad x \in [-0.7, 0.7], \quad (6.20)$$

and we use the same solution for u as the case in Section 6.2.3, i.e., $u = \sin^3(6x)$. In addition, $\lambda = 0.01$, and $k = 0.7$ denotes the reaction rate which is a constant, while f can then be derived from Equation 6.20. We assume that we have 32 sensors for f , which are equidistantly placed in $x \in [-0.7, 0.7]$. In addition, two sensors for u

are placed at $x = -0.7$ and 0.7 to provide Dirichlet boundary conditions. Here, we also consider two different scales of Gaussian noise in the measurements, i.e., (1) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$, and (2) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$, which is the same as in Section 6.2.3. The results of B-PINN-HMC, B-PINN-VI and dropout are illustrated in Figure 6.8.

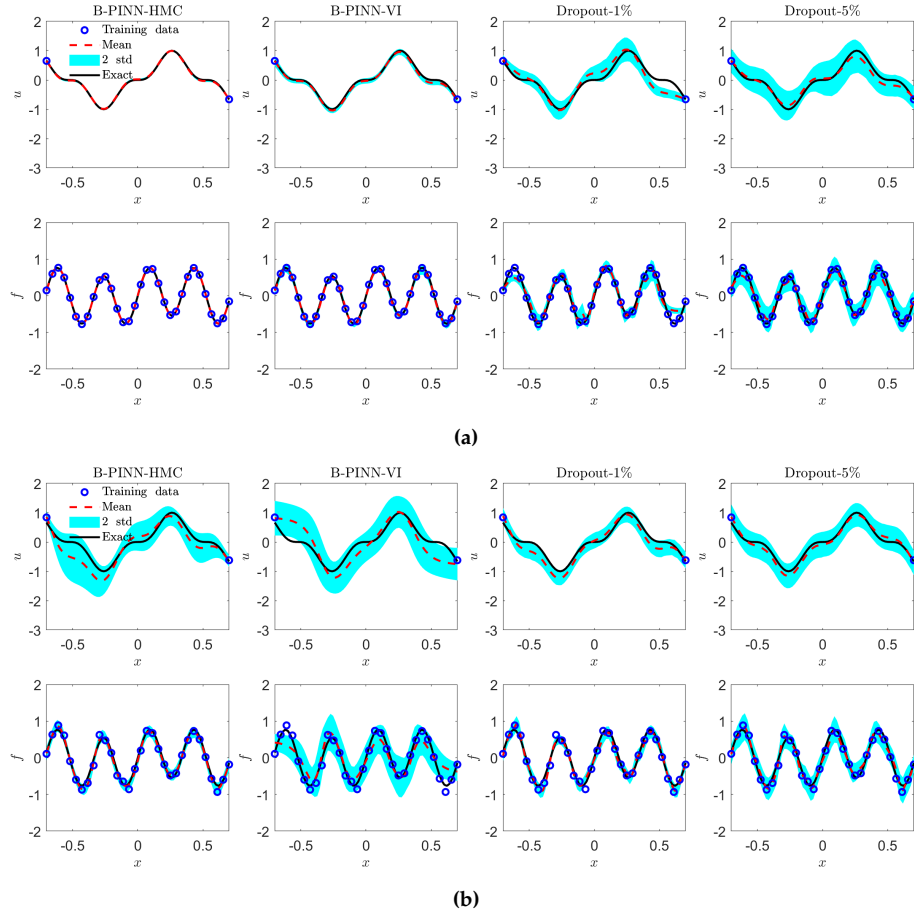


Figure 6.8: 1D nonlinear Poisson equation - forward problem: predicted u and f from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

The B-PINN-HMC provides good predictions for both u and f , which are similar as the results in Section 6.2.3. While able to give predicted means close to the exact solutions, B-PINN-VI and dropout can hardly give accurate uncertainty quantification for $u(x)$, e.g., (1) in the B-PINN-VI, the standard deviation for the

case with noise scale 0.1 is observed to be large at the boundaries even though we have observations for the boundary conditions, (2) the noise scale has little influence on the predicted standard deviations in the two dropout cases.

Forward Problem: 2D Nonlinear Reaction-Diffusion Equation

We further consider the following nonlinear Reaction-Diffusion equation, which is a widely used model for multi-phase flows:

$$\lambda(\partial_x^2 u + \partial_y^2 u) + u(u^2 - 1) = f, \quad x, y \in [-1, 1], \quad (6.21)$$

where $\lambda = 0.01$ represents the mobility, and u is the order parameter, which denotes different phases. Here, we employ the exact solution for $u = \sin(\pi x) \sin(\pi y)$. In addition, Dirichlet boundary conditions are imposed on all the boundaries. Similarly, we assume that we have 500 sensors for f , which are uniformly randomly distributed in the domain. In addition, we also have 25 equally distributed sensors for u at each boundary. Here we also consider two different noise scales on all the measurements, i.e., (1) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$, and (2) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$. The results of B-PINN-HMC, B-PINN-VI and dropouts, are illustrated in Figure 6.9.

Similarly, the B-PINN-HMC can provide predictive means close to the exact solutions, and the errors are mostly bounded by two standard deviations, which increases as the noise scale increases in the data. However, both the B-PINN-VI and the dropout with different drop rates fail to provide accurate means as well as uncertainties. Specifically, (1) the errors for the predicted u from the B-PINN-VI and the two dropouts can be even larger than 50% in part of the domain, (2) the

errors for u are not bounded by two standard deviations in the B-PINN-VI or in the two dropouts, and (3) the increase of the noise scale has little influence on the standard deviation when the dropout is employed.

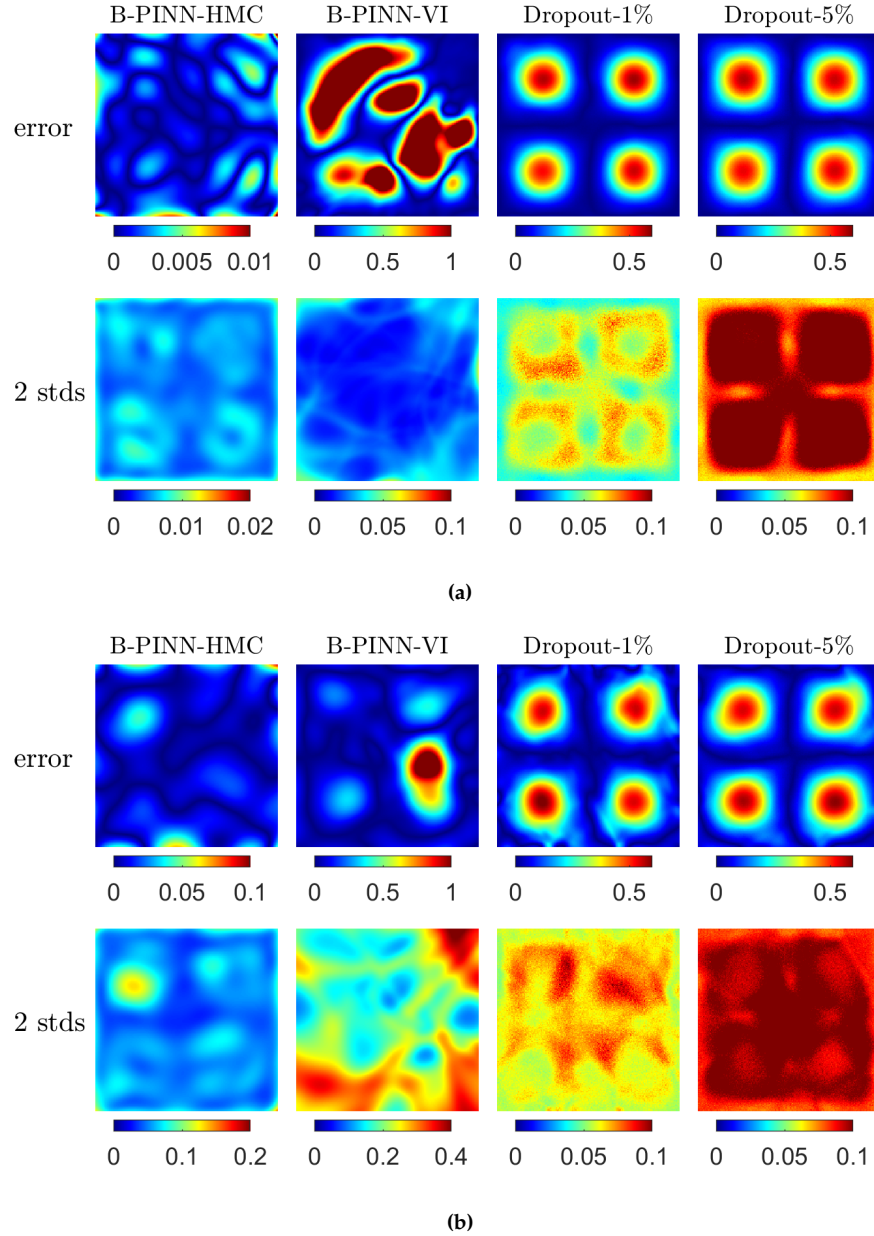


Figure 6.9: 2D Reaction-Diffusion equation - forward problem: Predicted errors and standard deviations for u from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

Inverse Problem: 1D Diffusion-Reaction System with Nonlinear Source Term

The PDE considered here is the same as Equation 6.20. However, k becomes an unknown parameter now. The objective here is to identify k based on partial measurements of f and u .

We assume that we have 32 sensors for f , which are equidistantly placed in $x \in [-0.7, 0.7]$. In addition, two sensors for u are placed at $x = -0.7$ and 0.7 to provide Dirichlet boundary conditions. Apart from the boundary conditions, another 6 sensors for u are placed in the interior of the domain to help identify k . We also assume that Gaussian noises are present for all the measurements. Two different scales of the noise are considered, i.e., (1) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$ and (2) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

The results of B-PINN-HMC, B-PINN-VI and dropouts are illustrated in Figure 6.10. The predicted means of u and f from the B-PINN-HMC fit the exact functions well for both cases. The errors of u and f using B-PINN-VI and dropout are observed to be larger than those using B-PINN-HMC for the case with noise scale 0.1.

The predicted values of k from different methods are displayed in Table 6.1. The predicted means of k for both cases using the B-PINN-HMC are quite accurate, with the error less than one standard deviation. Moreover, the standard deviation increases as the noise scale increases. The results show the effectiveness of the B-PINN-HMC in identifying the unknown parameter and quantifying the uncertainty arising from the scattered noisy data. The errors for B-PINN-VI are larger than B-PINN-HMC, although bounded by two standard deviations. As for the dropout, we use k from the last 10,000 training steps as samples to calculate the

mean and standard deviation, which is different from the posterior sampling used above. We observe that: (1) in both cases of noise scale, the errors of the predicted means with both dropout rates are larger than those from B-PINN-HMC, (2) the error increases as we increase the dropout rates, (3) for the case with dropout rate 5%, the standard deviation decreases as we increase the noise scale, which is not reasonable.

Noise scale		B-PINN-HMC	B-PINN-VI	Dropout-1%	Dropout-5%
0.01	Mean	0.705	0.708	0.714	0.669
	Std	5.75×10^{-3}	4.01×10^{-3}	4.38×10^{-3}	2.02×10^{-2}
0.1	Mean	0.665	0.775	0.746	0.633
	Std	5.63×10^{-2}	3.58×10^{-2}	6.508×10^{-3}	6.45×10^{-3}

Table 6.1: 1D diffusion-reaction system with nonlinear source term: Predicted mean and standard deviation for k using different uncertainty quantification methods. The exact value for k is 0.7.

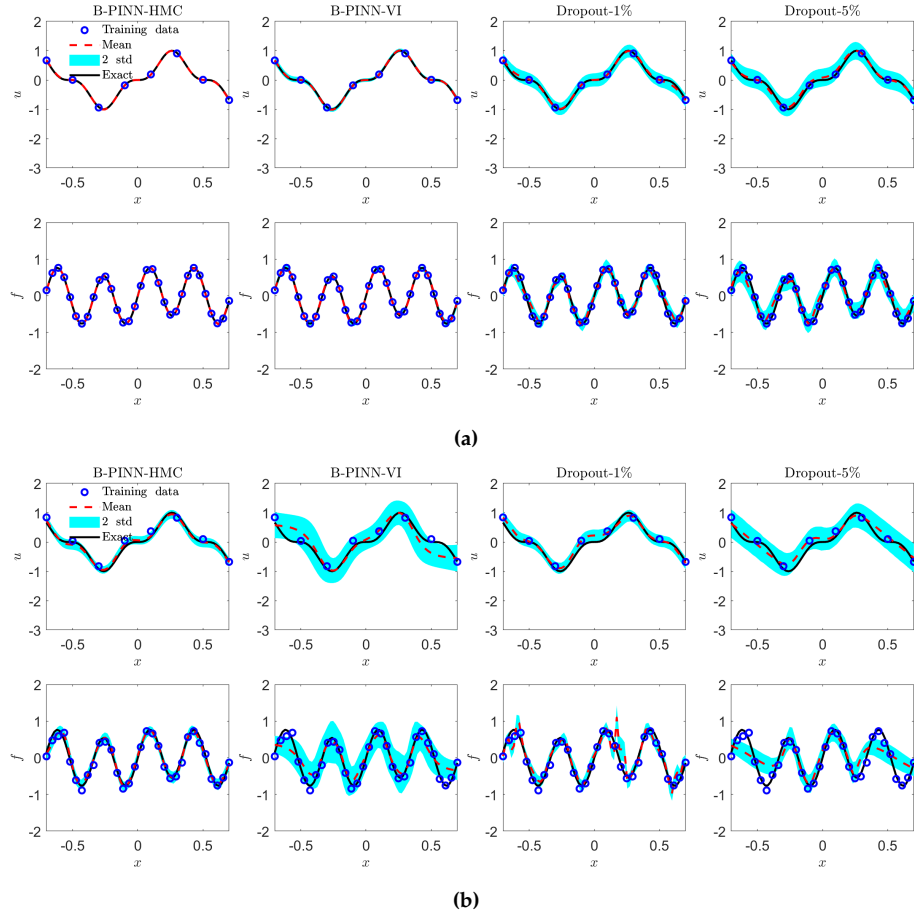


Figure 6.10: 1D diffusion-reaction system with nonlinear source term - inverse problem: predicted u and f from different methods with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

In Section 6.2.6 we also provide another case with less data on u . The error and uncertainty of inferred k is consequently larger than the results in Table 6.1 with the same noise size. The accurate prediction of u also demonstrates the effectiveness of B-PINN-HMC for the extrapolation of u with the help of the PDE constraint.

It is worth mentioning that the accuracy of the prediction would be significantly reduced if the scale of exact k does not match the prior. As an example, we first

rewrite the equation in Section 6.2.3 as

$$\lambda \partial_x^2 u + k\alpha \tanh(u) = f, \quad x \in [-0.7, 0.7], \quad (6.22)$$

where α is a constant, and $k\alpha = 0.7$. The objective is to infer k with known α and the measurements on u and f . Note that the solution for the above equation is the same as in Section 6.2.3 if the same boundary conditions are employed. Therefore, we use the same training data as the above case with noise scale $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, and $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$. We only present the results of B-PINN-HMC since it is the most accurate among all the tested approaches in the present study. To ensure the convergence of the HMC, we use two different burn-in steps, i.e., 50,000 and 100,000. For the case of $k = 50$, if we still use the prior $k \sim \mathcal{N}(0, 1)$, we would significantly underestimate k as shown in Table 6.2.

For cases where we do not have a clear knowledge of k 's scale, which for example could range from the magnitude of 1 to 100, another prior, e.g., $\log(k) \sim \mathcal{N}(0, 1^2)$, can be applied. In practice, we could replace k with $\exp(k')$ and set the prior for k' as $\mathcal{N}(0, 1^2)$. For the same problem where $k = 50$, as displayed in Table 6.2, the predicted k becomes more accurate using the new prior, and the computational errors are now bounded by the two standard deviations.

Prior for k		$N_{burnin} = 50,000$	$N_{burnin} = 100,000$
$k \sim \mathcal{N}(0, 1^2)$	Mean	42.77	43.27
	Std	1.39×10^{-1}	6.65×10^{-2}
$\log(k) \sim \mathcal{N}(0, 1^2)$	Mean	50.39	50.39
	Std	3.90×10^{-1}	3.96×10^{-1}

Table 6.2: 1D diffusion-reaction system with nonlinear source term: Predicted mean and standard deviation for k using different priors with exact value $k = 50$.

We further test the cases with $k = 100$ ($\alpha = 0.07$) and $k = 0.7$ ($\alpha = 1$) using the prior $\log(k) \sim \mathcal{N}(0, 1^2)$. As shown in Table 6.3, the predicted means for k converge to the exact values in both cases and the errors are bounded by two standard deviations.

exact k		$N_{burnin} = 50,000$	$N_{burnin} = 100,000$
100	Mean	100.78	100.75
	Std	7.95×10^{-1}	8.08×10^{-1}
0.7	Mean	0.705	0.706
	Std	5.6×10^{-3}	5.2×10^{-3}

Table 6.3: 1D diffusion-reaction system with nonlinear source term: Predicted mean and standard deviation for exact $k = 0.7$ and $k = 100$, using the prior $\log(k) \sim \mathcal{N}(0, 1^2)$.

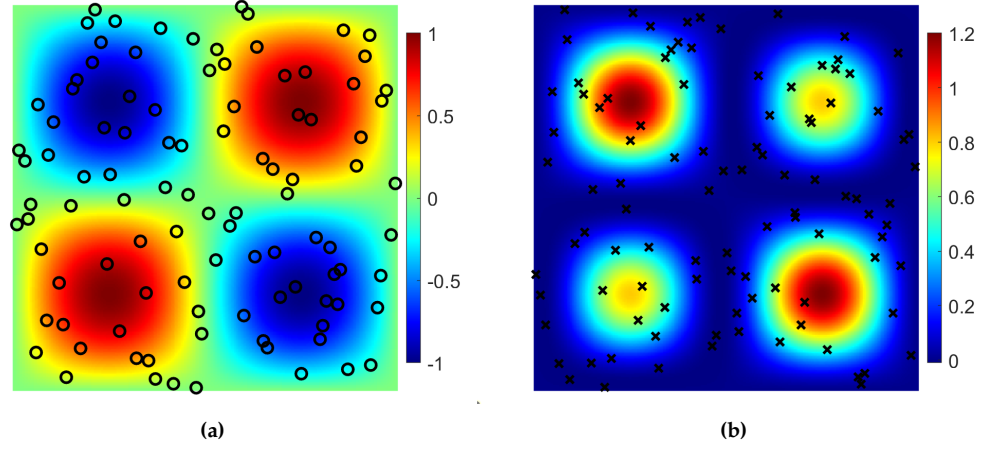


Figure 6.11: 2D nonlinear diffusion-reaction system: Training data for u and f . (a) Distribution of u . Black circle: training sample for u , (b) Distribution of f . Black cross: training samples for f .

Inverse Problem: 2D Nonlinear Diffusion-Reaction System

We consider the following PDE here

$$\lambda(\partial_x^2 u + \partial_y^2 u) + ku^2 = f, \quad x, y \in [-1, 1], \quad (6.23)$$

where $\lambda = 0.01$ is the diffusion coefficient, k represents the reaction rate, which is a constant, and f denotes the source term. Here we assume that the exact value for k is unknown, and we only have sensors for u and f . Specifically, we have 100 sensors which are randomly sampled in the physical domain (Figure 6.11) for u and f . we also have 25 equally distributed sensors for u at each boundary for the Dirichlet boundary condition. Similarly, all the measurements are noisy and two noise scales are considered here, i.e., (1) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$, and (2) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$. We then aim to estimate the reaction rate k given the measurements of u and f .

As we can see in Table 6.4, for both cases with noise scale 0.1 and 0.01, the predictive means using the B-PINN-HMC are quite accurate with errors less than

Noise scale		B-PINN-HMC	B-PINN-VI	Dropout-1%	Dropout-5%
0.01	Mean	1.003	0.895	1.050	1.168
	Std	5.75×10^{-3}	2.83×10^{-3}	2.00×10^{-3}	3.04×10^{-3}
0.1	Mean	0.978	1.116	1.020	1.169
	Std	4.98×10^{-2}	3.45×10^{-2}	4.21×10^{-3}	4.15×10^{-3}

Table 6.4: 2D nonlinear diffusion-reaction system: Predicted mean and standard deviation for the reaction rate k using different uncertainty-induced methods. $k = 1$ is the exact solution.

5%. Also, the standard deviations, which represent the uncertainties, are in the same order of magnitude as the errors of predictive means. The uncertainty decreases as the scale of noise in data decreases. The results show the effectiveness of these two models in identifying the unknown parameter and quantifying the uncertainties arising from the noise in data. As for the B-PINN-VI, the relative errors between the predicted mean value of k from and the exact k are greater than 10% for both cases. With regards to the dropout, the means and the standard deviations for k are calculated in the same way as in Section 6.2.3. The predicted means for k from the dropout with dropout rate 0.01 are quite close to the exact k . As the dropout rate increases to 0.05, the errors become about 17% for both cases. The above results show that the dropout rate has a strong effect on the predictive accuracy. However, there is no theory on the choice of the optimal dropout, which clearly restricts its usefulness in applications. In addition, the influence of the noise scales on the standard deviations is not significant, indicating dropout is not suitable for quantifying uncertainty from noisy data.

Inverse Problem: Six-Dimensional Source Inversion problem

We now apply the B-PINNs to a high-dimensional contaminant source inversion problem, in which the locations of three contaminant sources are inferred based on

a number of noisy observations. Specifically, we consider the following diffusion-reaction system in a two-dimensional spatial domain:

$$-\lambda(\partial_x^2 + \partial_y^2)u - f_2 = f_1, \quad x, y \in [0, 1], \quad (6.24)$$

where u is the concentration of the contaminant, $\lambda = 0.02$ is the diffusion coefficient, and f_2 is the unknown source, which is parameterized as

$$f_2 = \sum_{i=1}^3 k_i \exp \left[-0.5 \frac{\|\mathbf{x} - \mathbf{x}_{c,i}\|^2}{0.15^2} \right], \quad (6.25)$$

where $\mathbf{k} = (2, -3, 0.5)$ are known constants, and $\mathbf{x}_{c,i}$ denote the centers of the contaminant, which will be inferred based on the noisy observations of u and f_1 .

Here, we set the exact f_1 as $f_1 = 0.1 \sin(\pi x) \sin(\pi y)$ and the exact locations for three sources are: $\mathbf{x}_{c,1} = (0.3, 0.3)$, $\mathbf{x}_{c,2} = (0.75, 0.75)$, and $\mathbf{x}_{c,3} = (0.2, 0.7)$. We consider the case where the noise for the measurements is: $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$ and $\epsilon_{f_1} \sim \mathcal{N}(0, 0.01^2)$. The *solvepde* package in Matlab with 1893 triangle meshes is used to solve u . We then utilize 1,000 and 200 random samples for u and f_1 as training data, respectively. The priors for the hyperparameters in the BNNs and the unknown parameters in the PDE are $\mathcal{N}(0, 1^2)$ and $\log(\mathbf{x}_{c,i}) \sim \mathcal{N}(0, 1^2)$, respectively. As displayed in Table 6.5, the B-PINN-HMC is the only approach that is capable of providing accurate predictions for all $\mathbf{x}_{c,i}$, i.e, (1) the predicted means are quite close to the exact solution, and (2) the computational errors are all bounded by the one standard deviation.

		$\mathbf{x}_{c,1}$	$\mathbf{x}_{c,2}$	$\mathbf{x}_{c,3}$
B-PINN-HMC	Mean	(0.3024, 0.3014)	(0.7492, 0.7481)	(0.1926, 0.6984)
	Std ($\times 10^{-3}$)	(3.6, 4.9)	(3.4, 2.5)	(24.5, 24.8)
B-PINN-VI	Mean	(0.3226, 0.1847)	(0.4788, 6.27×10^{-4})	(0.2288, 0.2764)
	Std ($\times 10^{-4}$)	(2.04, 2.22)	(1.99, 1.40)	(8.11, 7.93)
Dropout-1%	Mean	(0.3037, 0.3218)	(0.7158, 0.7367)	(0.4300, 0.7336)
	Std ($\times 10^{-4}$)	(4.84, 6.82)	(3.63, 4.00)	(21.0, 19.0)
Dropout-5%	Mean	(0.4181, 0.4413)	(0.6693, 0.6521)	(0.7319, 0.6127)
	Std ($\times 10^{-4}$)	(5.20, 6.09)	(4.16, 4.41)	(8.15, 6.39)

Table 6.5: Source inversion problem: Predicted mean and standard deviation for the locations using different uncertainty-induced methods. The exact locations for three sources are: $\mathbf{x}_{c,1} = (0.3, 0.3)$, $\mathbf{x}_{c,2} = (0.75, 0.75)$, and $\mathbf{x}_{c,3} = (0.2, 0.7)$.

In Section 6.2.6, we show another high-dimensional case, where the unknown term in the PDE is a field parameterized by a neural network; these results also show the effectiveness of the B-PINN-HMC.

6.2.4 Comparison with PINNs

In this section, we will conduct a comparison between the B-PINN-HMC and PINNs for the 1D inverse problem in Section 6.2.3. We employ the Adam optimizer with $l = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ to train the PINN, with the number of the training steps set as 200,000. The results of the PINNs are shown in Figure 6.12.

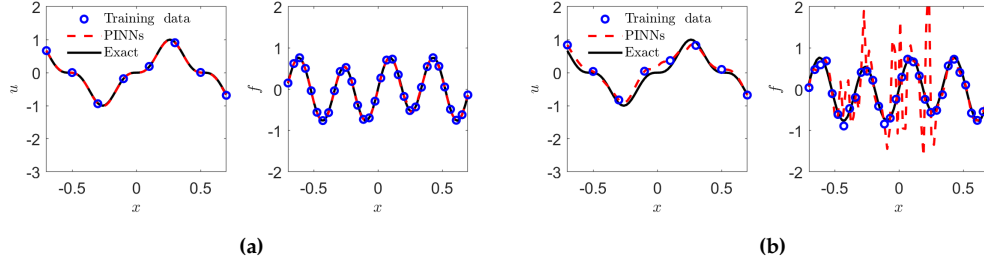


Figure 6.12: 1D diffusion-reaction system with nonlinear source term (PINNs): Predicted u and f with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

As shown in Figure 6.12, the predicted u and f could fit all the training points. In the cases where the noise scale is as small as 0.01, the predictive u and f agree well with the exact solutions. However, as the noise scale increases to 0.1, significant overfitting is observed in PINNs. In addition, PINNs predict k to be 0.705 and 0.591 for the noise level at 0.01 and 0.1, respectively, while the reference exact solution is 0.7. Comparing with the results of B-PINN-HMC in Table 6.1, we conclude that PINNs can provide prediction with similar accuracy as the B-PINN-HMC for the case with small noise in data, while B-PINN-HMC shows significant advantage in accuracy over PINNs for the case with large noise. While PINNs could overcome overfitting if additional regularization for the parameters are applied in the loss, it cannot quantify uncertainties of the predictive results.

Now, we conduct a brief comparison on the computational cost between PINNs and B-PINN-HMC based on the inverse problem. We run both the PINNs and B-PINN-HMC codes on two CPUs (Intel Xeon E5-2643). For the PINN, the computational time is about 10 minutes, while it takes about 20 minutes for the B-PINN-HMC. Despite this relatively small increase for B-PINNs versus PINNs for this small problem, we expect that when we scale up the data size and neural network size the difference in cost will increase accordingly. Considering the ac-

curacy as well as the reliable uncertainty provided, the B-PINN-HMC may be a better approach than the PINNs for scenarios with large noise.

6.2.5 Comparison with the Truncated Karhunen-Loève Expansion

So far we have shown the effectiveness of B-PINNs in solving PDE problems. As we know, a neural network is extremely overparametrized. Hence, we want to investigate if we can use other models with less parameters for our surrogate model in the Bayesian framework. For example, we consider the Karhunen-Loève expansion, a widely used representation for a stochastic process in the following study.

Truncated Karhunen-Loève Expansion

Assume $u(\mathbf{x})$ is a stochastic process with mean $\mu(\mathbf{x})$ and covariance function (also called “kernel”) $k(\mathbf{x}, \mathbf{x}')$, then the KL expansion of u is

$$u(\mathbf{x}) = \mu(\mathbf{x}) + \sum_{i=1}^{\infty} \sqrt{\alpha_i} \psi_i(\mathbf{x}) \theta_i, \quad (6.26)$$

where ψ_i are the orthogonal eigenfunctions, α_i are the corresponding eigenvalues of the kernel, and θ_i are mutually uncorrelated random variables. In practice, we could truncate the expansion to n terms as our surrogate model for u :

$$\tilde{u}(\mathbf{x}; \boldsymbol{\theta}) = \mu(\mathbf{x}) + \sum_{i=1}^n \sqrt{\alpha_i} \psi_i(\mathbf{x}) \theta_i, \quad (6.27)$$

where $\theta = (\theta_1, \theta_2 \dots \theta_n)$ is the parameter in the surrogate model whose prior distribution is given by the KL expansion.

One of the main differences between the truncated Karhunen-Loève expansion and neural networks as surrogate models is the number of parameters. For example, in this section, the neural network used in 1D problems has 2701 parameters. As a comparison, the truncated Karhunen-Loève expansion used in Section 6.2.5 only has 20 parameters. The small number of parameters makes it possible to use another approach to sample from the posterior, namely the deep normalizing flow (DNF) models.

In general, using DNF to sample from a target distribution ν consists of the following three steps [139]:

1. Define a bijective transformation $G : \mathbb{R}^{d_\theta} \rightarrow \mathbb{R}^{d_\theta}$ and prescribe an input distribution μ_I of the dimension $\mathbb{R}^{d_\theta}$. Usually, the bijective transformation is parameterized by deep neural networks and the input distribution can be a standard multivariate Gaussian distribution.
2. Note that the bijective transformation G will map the input distribution to an output distribution $\mu_O = G_\# \mu_I$. We then train the parameters in G to minimize $F(\mu_O, \nu)$, where F is a functional that measures the difference between two distributions. Ideally, μ_O and ν will be sufficiently close to each other after this procedure.
3. Finally, we sample from the input distribution μ_I , denoted as $\{z^{(j)}\}_{j=1}^M$. Then $\{G(z^{(j)})\}_{j=1}^M$ as samples of μ_O can be used to approximate the statistics of ν .

We leave the details of the DNF in Section 6.2.6.

Results and Comparisons

In this section we apply the truncated Karhunen-Loève expansion to solve the forward and inverse nonlinear PDE problems as described in Section 6.2.3 and Section 6.2.3. In particular, we consider the Gaussian process of zero mean and exponential kernel

$$k(x, x') = \exp\left(-\frac{|x - x'|}{0.25}\right), \quad x \in [-1, 1], \quad (6.28)$$

and use the first 20 terms of the KL expansion as our surrogate model for u , which retains about 92% of the energy. For this case, the eigenvalues and eigenfunctions in the KL expansion are solved analytically, and the prior for the unknown parameters is the product of independent standard Gaussian distributions. We refer the readers to example 4.1 in Chapter 4 in [134] for details.

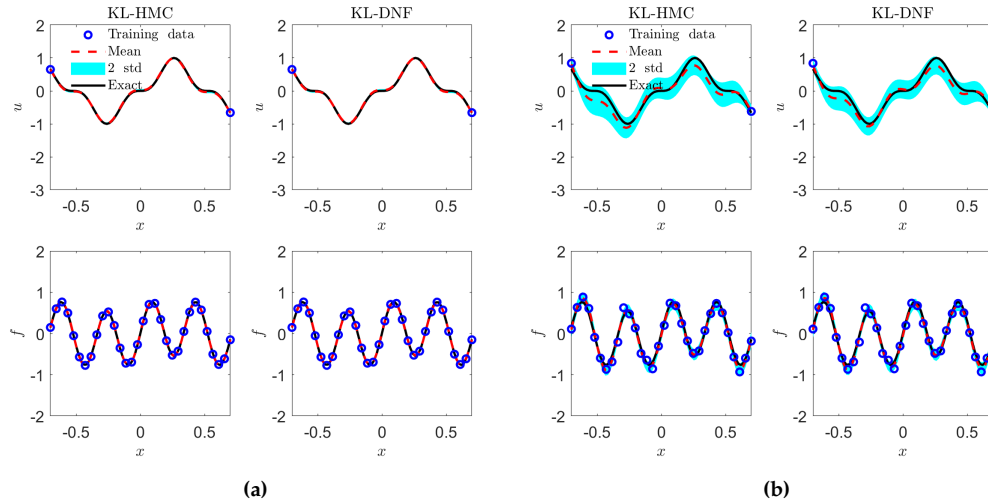


Figure 6.13: 1D nonlinear Poisson equation (KL) - forward problem: Predicted u and f with two data noise scales. (a): $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b): $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

The predicted u and f are illustrated in Figure 6.13-6.14. The results from the KL-HMC and KL-DNF are almost the same, and the predicted means for u and f are close to the exact solutions. In addition, the predicted k is displayed in

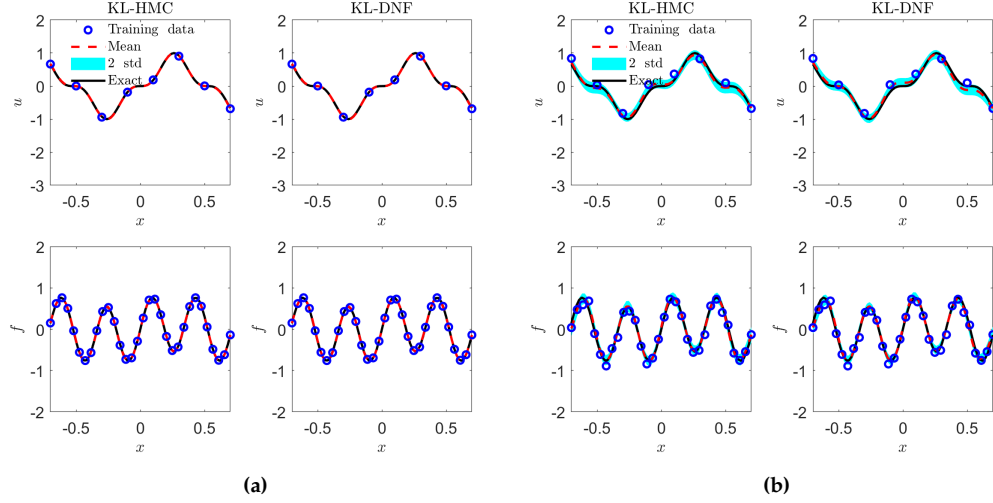


Figure 6.14: 1D diffusion-reaction system with nonlinear source term (KL): Predicted u and f with two data noise scales. (a) $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_b \sim \mathcal{N}(0, 0.1^2)$.

Noise scale		KL-HMC	KL-DNF
0.01	Mean	0.706	0.705
	Std	5.63×10^{-3}	2.63×10^{-3}
0.1	Mean	0.694	0.709
	Std	5.82×10^{-2}	5.21×10^{-2}

Table 6.6: 1D diffusion-reaction system with nonlinear source term (KL): Predicted mean and standard deviation for k using KL. The exact value for k is 0.7.

Table 6.6. Similarly, the predicted means for both cases fit the exact solution quite well. Furthermore, we also note that (1) the standard deviation increases with the increasing noise scale, and (2) the errors are bounded by two standard deviations. All the results are similar as those from the B-PINNs presented in Section 6.2.3 and Section 6.2.3.

As for the computational cost, the DNF takes about one day to finish the training for the 1D diffusion-reaction problem, while it takes about 4 mins for the HMC. Although the DNF is computationally much more expensive than the HMC, we would like to remark that upon completion of the training, it is more convenient to draw *independent* samples from the target distribution using DNF

compared with HMC. This strength of DNF has no significant benefit in current work, but could be helpful for other tasks.

Here, we also conduct a brief comparison on the computational cost between the KL-HMC and the B-PINN-HMC based on the 1D diffusion-reaction problem. Due to the relatively small number of parameters in the truncated KL expansion, in the 1D test cases, KL-HMC takes much less time than B-PINN-HMC. In particular, KL-HMC takes about 4 mins compared to about 20 mins for B-PINN-HMC. However, we remark that the truncated KL expansion would suffer from the “curse of dimensionality” when approximating high dimensional functions, while deep neural networks are known to be efficient for high-dimensional function approximation [20].

6.2.6 Supplementary Material

BNNs with Different Priors

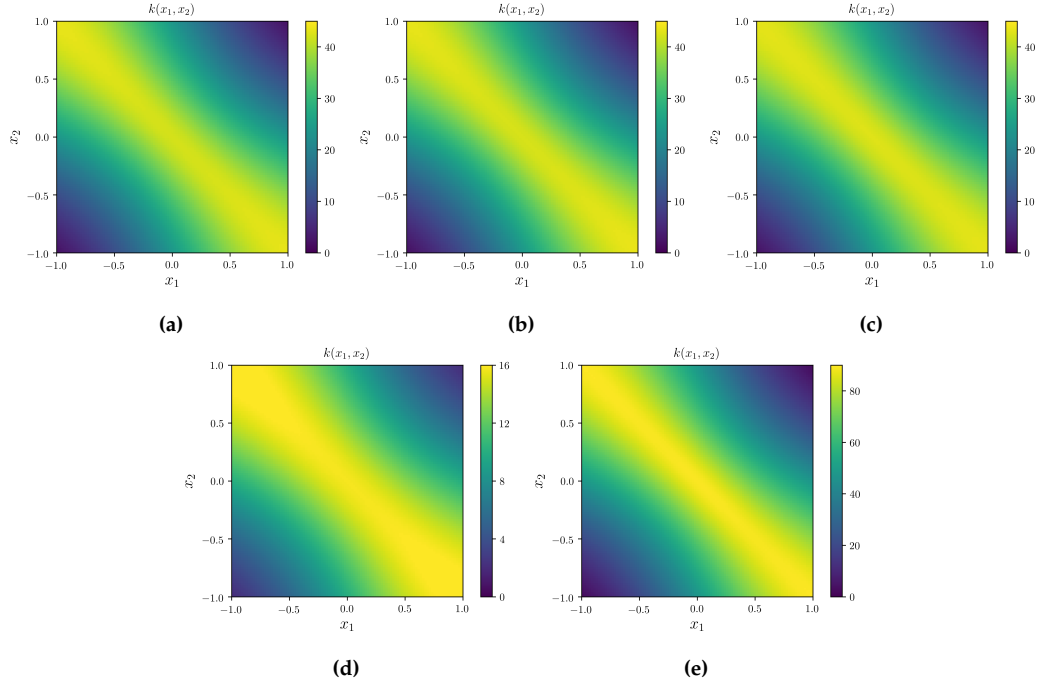


Figure 6.15: Covariance functions $k(x_1, x_2) = \text{cov}(\tilde{u}(x_1), \tilde{u}(x_2))$ for BNNs with different architectures. $L = 2, \sigma_{b,l} = 1$ for $l = 0, 1, 2$ in all the cases. (a) $N_1 = N_2 = 20, \sigma_{w,0} = 1.0, \sigma_{w,1} = \sigma_{w,2} = \sqrt{5}/2$ (b) $N_1 = N_2 = 50, \sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$, (c) $N_1 = N_2 = 100, \sigma_{w,0} = 1.0, \sigma_{w,1} = \sigma_{w,2} = \sqrt{1}/2$. (d) $N_1 = N_2 = 20, \sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$, (e) $N_1 = N_2 = 100, \sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$.

The posterior distribution depends on both the prior and the observed data in the Bayesian framework. Given the same observation, surrogate models with similar prior distributions should also provide similar posterior distributions. For the cases of input dimension $N_x = 1$, we illustrate the covariance functions for five representative priors in Figure 6.15, which are estimated from 100,000 independent samples of neural network parameters drawn from the prior. Note that $\sqrt{N_l}\sigma_{w,l}$ is fixed for cases (a), (b) and (c), and we can see that the covariance functions are similar for the three cases.

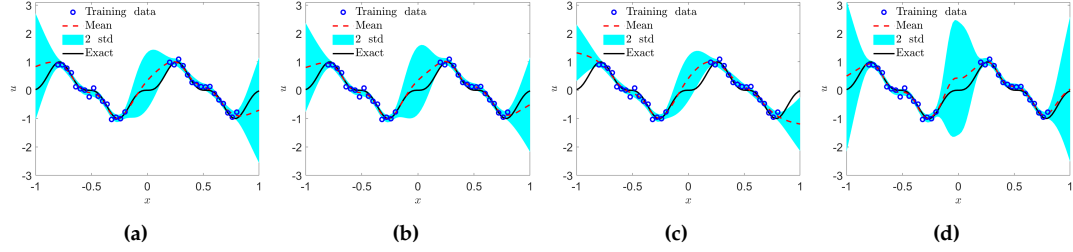


Figure 6.16: BNN-HMC with different priors for function approximation. $L = 2, \sigma_{b,l} = 1$ for $l = 0, 1, 2$ in all the cases. (a) $N_1 = N_2 = 20, \sigma_{w,0} = 1.0, \sigma_{w,1} = \sigma_{w,2} = \sqrt{5}/2$, (b) $N_1 = N_2 = 100, \sigma_{w,0} = 1.0, \sigma_{w,1} = \sigma_{w,2} = \sqrt{1}/2$, (c) $N_1 = N_2 = 20, \sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$, (d) $N_1 = N_2 = 100, \sigma_{w,0} = \sigma_{w,1} = \sigma_{w,2} = 1.0$.

We plot the results for approximating the same function in Equation 6.16 with same data, using BNNs with different architectures in Figure 6.16. The predicted means and standard deviations are observed to be quite similar for cases in Figure 6.16a-6.16c, which is consistent with the fact that the covariance functions of the priors for these three cases are similar (Figure 6.15a-6.15c). In addition, the results in Figure 6.16c-6.16d are different from those in Figure 6.3e, which is not surprising since their covariance functions are totally different (Figure 6.15).

Deep Normalizing Flow Models

Deep normalizing flow (DNF) models provide a powerful mechanism for sampling from a wide range of probability distributions. While there have been many versions of normalizing flow models, as a demonstrating example, in this chapter we use a potential flow to build the bijective transformation. We refer the readers to [139, 152] for similar approaches. In particular, the bijective transformation G is defined as the map from $\mathbf{u}$ at time $t = 0$ to T of the following ODE:

$$\frac{d\mathbf{u}}{dt} = \nabla\varphi(\mathbf{u}, t; \boldsymbol{\zeta}), \quad (6.29)$$

where $\varphi(\mathbf{u}, t; \zeta)$ is represented by a deep neural network with parameter ζ , which takes the concatenation of $\mathbf{u}$ and t as input, and outputs a real number. Consequently, we have the following ODE for the probability density:

$$\frac{d \ln P(\mathbf{u}(t), t)}{dt} = -\nabla^2 \varphi(\mathbf{u}, t; \zeta), \quad (6.30)$$

where $P(\mathbf{u}, 0) = P_{\mu_I}(\mathbf{u})$ is the density of μ_I at $\mathbf{u}$, and $P(\mathbf{u}, T) = P_{\mu_O}(\mathbf{u})$ is the density of μ_O at $\mathbf{u}$.

Here, we use the forward Euler scheme to solve the ODE (6.29). Suppose the time step is $\delta t = T/n$, then

$$\begin{aligned} \mathbf{u}_0(z) &= z, \\ \mathbf{u}_i(z) &= \mathbf{u}_{i-1}(z) + \delta t \nabla \varphi(\mathbf{u}_{i-1}(z), \frac{(i-1)T}{n}; \zeta), \quad i = 1, 2, \dots, n \end{aligned} \quad (6.31)$$

so that $G(z) = \mathbf{u}_n(z)$. Similarly, we have the forward Euler scheme for ODE (6.30):

$$\begin{aligned} \ln P_0(\mathbf{u}_0(z)) &= \ln P_{\mu_I}(z), \\ \ln P_i(\mathbf{u}_i(z)) &= \ln P_{i-1}(\mathbf{u}_{i-1}(z)) - \delta t \nabla^2 \varphi(\mathbf{u}_{i-1}(z), \frac{(i-1)T}{n}; \zeta), \quad i = 1, 2, \dots, n \end{aligned} \quad (6.32)$$

so that $\ln P(G(z), T) = \ln P_n(\mathbf{u}_n(z))$.

For our problems, where the target distribution ν is given by the posterior

density $P(\theta|\mathcal{D})$, we tune the parameters in φ to minimize

$$\begin{aligned}
F(\mu_O, \nu) &= D_{KL}(\mu_O||\nu) \\
&= \mathbb{E}_{\theta \sim \mu_O} [\ln P(\theta, T) - \ln P(\theta|\mathcal{D})] \\
&\simeq \mathbb{E}_{\theta \sim \mu_O} [\ln P(\theta, T) - \ln P(\theta) - \ln P(\mathcal{D}|\theta)] \\
&= \mathbb{E}_{z \sim \mu_I} [\ln P(G(z), T) - \ln P(G(z)) - \ln P(\mathcal{D}|G(z))],
\end{aligned} \tag{6.33}$$

where D_{KL} represents the Kullback-Leibler divergence, and “ $\simeq$ ” represents equality up to a constant. In this chapter we employ the Adam optimizer to train ζ .

Ideally, μ_O and ν would be sufficiently close to each other after the convergence of $F(\mu_O, \nu)$. We could then sample $\{z^{(j)}\}_{j=1}^M$ from μ_I , and get statistics of $P(\theta|\mathcal{D})$ from $\{G(z^{(j)})\}_{j=1}^M$.

The detailed algorithm is given in Algorithm 6.

Algorithm 6 Normalizing Flow

Require: an initial state for ζ .

- 1: **for** $k = 1, 2, \dots, N$ **do**
 - 2: Sample $\{z^{(j)}\}_{j=1}^{N_z}$ independently from μ_I .
 - 3: $L(\zeta) \leftarrow \frac{1}{N_z} \sum_{j=1}^{N_z} [\ln P(G(z^{(j)}), T) - \ln P(G(z^{(j)})) - \ln P(\mathcal{D}|G(z^{(j)}))]$.
 - 4: Update ζ with gradient $\nabla_{\zeta} L(\zeta)$ using Adam optimizer.
 - 5: **end for**
 - 6: Sample $\{z^{(j)}\}_{j=1}^M$ independently from μ_I .
 - 7: Calculate $\{\tilde{u}(x, G(z^{(j)}))\}_{j=1}^M$ as samples of $u(x)$, similarly for other terms.
-

Here we set time span $T = 1$, and time steps in the forward Euler scheme $n = 50$ in the forward problems, while $n = 10$ in the inverse problems. The neural

networks for φ have 3 hidden layers, each of width 128. For all the cases, the total training steps $N = 100,000$ and batch size $N_z = 16$. The hyperparameters for the Adam optimizer are set as $l = 10^{-4}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$.

Inferring Space-Dependent Reaction Rate

We consider the following one-dimensional diffusion-reaction system in which the reaction rate is a function:

$$\lambda \partial_x^2 u - ku = f, \quad x \in [0, 1], \quad (6.34)$$

where $\lambda = 0.01$ is the diffusion coefficient, u is the solute concentration, k is a space-dependent reaction rate, and $f = \sin(2\pi x)$ is the source term. The objective is to infer k given noisy measurements on u and f .

Here, we set the unknown reaction rate as

$$k = 0.1 + \exp \left[-0.5 \frac{(x - 0.5)^2}{0.15^2} \right]. \quad (6.35)$$

In addition, the condition $u(x) = 0$ is imposed at $x = 0$ and 1 . To generate training data, we first employ the *bvp5c* package with 1,000 uniform grids in Matlab to solve u . We then randomly select 50 points for u and f , respectively, and make observations with Gaussian noise on these points as our training data, specifically, two noise scales are considered: (a) $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, and (b) $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$.

We employ another neural network to parameterize the reaction rate, which has the same architecture as the NN for u used before, i.e., 2 hidden layers with

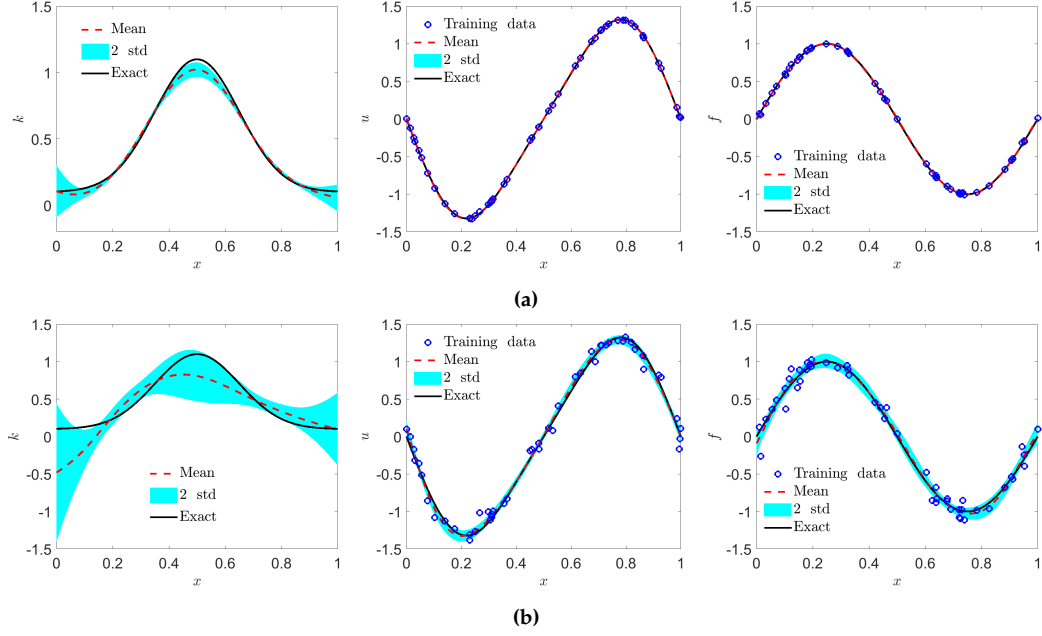


Figure 6.17: Space-dependent reaction rate: Predicted k , u and f using B-PINN-HMC. (a) $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, and $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, and $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$.

50 neurons per layer. In addition, the activation function as well as the priors for the weights/biases are all kept the same as the previous ones. For brevity, we only present the results from the B-PINN-HMC. As shown in Figure 6.17, the predicted means for k in Figure 6.17a fit the exact solution better than in Figure 6.17b. Furthermore, the errors in both cases are mostly bounded by the two standard deviations, which indicates the reliability of the predictive uncertainties.

Extrapolation with B-PINNs

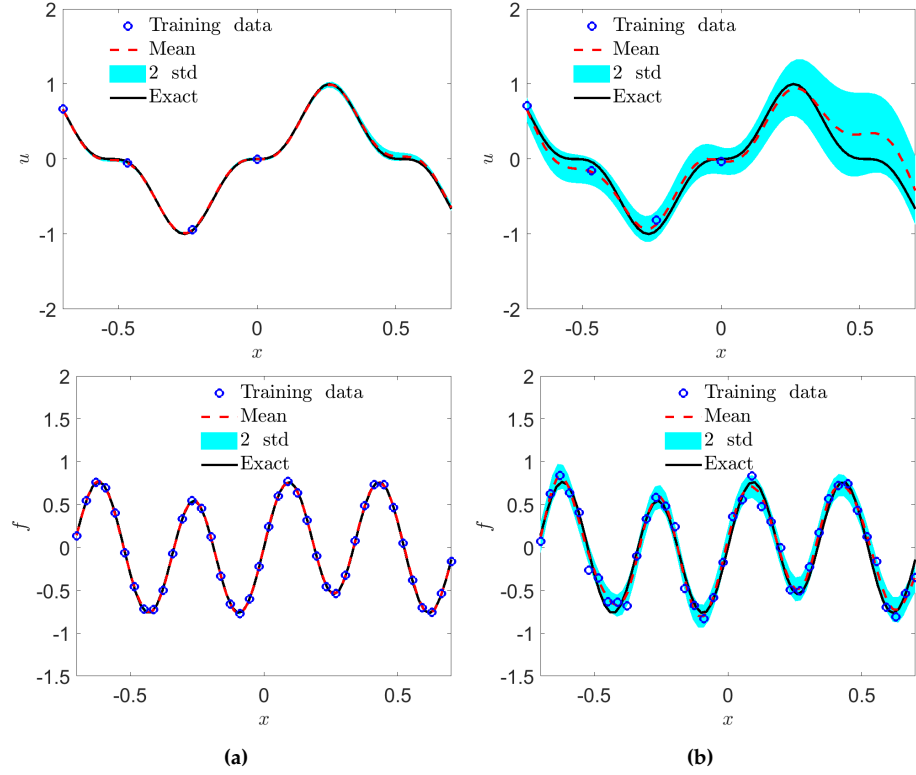


Figure 6.18: Extrapolation with B-PINNs: Predicted u and f using B-PINN-HMC with a limited number of observations on u . (a) $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$. (b) $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$.

Here we test B-PINN-HMC on the 1D inverse problem considered in Section 6.2.3 lacking one-side boundary condition of u . In particular, we only employ 4 measurements on u at $x \leq 0$, aiming to infer k and also extrapolate u at $x > 0$ with the help of the PDE constraint as well as the 40 uniform measurements on f . We also test two cases with different noise scales, i.e., (a) $\epsilon_u \sim \mathcal{N}(0, 0.01^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.01^2)$, and (b) $\epsilon_u \sim \mathcal{N}(0, 0.1^2)$, $\epsilon_f \sim \mathcal{N}(0, 0.1^2)$.

As shown in Figure 6.18, it is interesting to find that the predicted means for u fit the exact solution quite well in the range of $x \in [0, 0.7]$ even though we have no observations of u for $x > 0$, which clearly demonstrates the effectiveness of

our method for the extrapolation of u with the PDE constraint. In addition, the predicted means and standard deviations for k in the two cases are: (a) 0.695 and 0.009, and (b) 0.647 and 0.078. Both results are in good agreement with the exact values.

6.3 Physics-Informed GANs for Priors in Bayesian Uncertainty Quantification

In Section 6.2 we introduced how to use B-PINNs to perform uncertainty quantification for forward and inverse PDE problems in the Bayesian framework. There still remains an important question: how to set the prior distributions? While we have discussed the prior for B-PINNs in Section 6.2.6, we still have little knowledge about how to set the prior distributions for the neural network parameters, given a prescribed functional prior. Such functional priors could come from various sources, here we list some of them:

- A random field model based on people's belief, e.g., a Gaussian process with a certain kernel.
- Simulation results with different parameters, boundary/initial conditions, etc.
- Historical data collect from sensors placed on an hidden random field.

In the latter two cases, usually we do not have access to an analytical expression of the prior, but instead only have finite data collected from a hidden distribution.

Also, these data could be collected from multiple random fields in a system, which are governed by some PDEs, and we thus need to leverage the physical knowledge to integrate these data for the prior of the system.

Actually, the physics-informed GANs (PI-GANs) introduced in Chapter 1 already provided a tool to integrate data from multiple random fields and model stochastic physical systems, leveraging the physical knowledge in the form of stochastic differential equations. This inspires us to use PI-GANs to learn the prior. In [96, 97], GANs are also proposed to provide priors for images, but here we use physics-informed GANs to learn the functional priors with data collected from scattered sensors placed on various terms that are intrinsically connected by the physics.

6.3.1 Learning Priors with Physics-Informed GANs

Taking an equation of the form $\mathcal{N}_x u(x) = f(x)$ as an example, where $\mathcal{N}$ is a general differential operator. We will use a generative model $G_\eta(x; \xi)$ (namely a generator), which is a function of the spatial coordinate x and noise ξ , typically standard Gaussian, to represent the prior for u . Here η denotes the parameters in G_η . We will discuss the architectures of G_η later. Consequently, we use $\mathcal{N}_x G_\eta(x; \xi)$ to represent the prior for f , where the derivatives of G_η can be calculated by auto differentiation. Similarly for the boundary b . With the “snapshots” as data, i.e., simultaneous reads at finite sensors of u , f and b , we will train $G_\eta(x; \xi)$ in an adversarial way, so that generated “fake” snapshots can approximate the real ones. The details are presented as follows.

We will use

$$\begin{aligned}
\tilde{u}_\eta(x; \xi) &= G_\eta(x; \xi) \\
\tilde{f}_\eta(x; \xi) &= \mathcal{N}_x G_\eta(x; \xi) \\
\tilde{b}_\eta(x; \xi) &= \mathcal{B}_x G_\eta(x; \xi)
\end{aligned} \tag{6.36}$$

to represent $u(x)$, $f(x)$, and boundary $b(x)$, where $\mathcal{B}_x$ is the boundary operator. Suppose we place $\bar{N}_u$ u -sensors at $\{\bar{x}_u^{(i)}\}_{i=1}^{\bar{N}_u}$, $\bar{N}_f$ f -sensors at $\{\bar{x}_f^{(i)}\}_{i=1}^{\bar{N}_f}$, and $\bar{N}_b$ b -sensors at $\{\bar{x}_b^{(i)}\}_{i=1}^{\bar{N}_b}$. Simultaneous read of all these sensors would yield so-called “snapshots”, which is a random vector in $\mathbb{R}^{\bar{N}_u + \bar{N}_f + \bar{N}_b}$, denoted as $T \sim P_r, T \in \mathbb{R}^{\bar{N}_u + \bar{N}_f + \bar{N}_b}$, where P_r is the distribution for T . This is the data we use to train the generator G_η to learn the prior.

On the other hand, we can generate “fake snapshots” $Q(\xi) \in \mathbb{R}^{\bar{N}_u + \bar{N}_f + \bar{N}_b}$ with G_η :

$$\begin{aligned}
Q(\xi) &= (\tilde{U}(\xi), \tilde{F}(\xi), \tilde{B}(\xi)), \\
\tilde{U}(\xi) &= (\tilde{u}_\eta(\bar{x}_u^{(i)}; \xi))_{i=1}^{\bar{N}_u}, \\
\tilde{F}(\xi) &= (\tilde{f}_\eta(\bar{x}_f^{(i)}; \xi))_{i=1}^{\bar{N}_f}, \\
\tilde{B}(\xi) &= (\tilde{b}_\eta(\bar{x}_b^{(i)}; \xi))_{i=1}^{\bar{N}_b}.
\end{aligned} \tag{6.37}$$

We can use another neural network as the discriminator, which takes the real or fake snapshots as input and output a real number, denoted as D_ρ with ρ as the parameters. Here we use Wasserstein GAN with gradient penalty (WGAN-GP) [43] as the a version of GANs in this section, which can be interpreted as using Wasserstein-1 distance as the metric between the generated distributions and the

target one. Then the loss function for G_η and D_ρ are

$$\begin{aligned} L_G &= -\mathbb{E}_\xi[D_\rho(Q(\xi))], \\ L_D &= \mathbb{E}_\xi[D_\rho(Q(\xi))] - \mathbb{E}_{T \sim P_r}[D_\rho(T)] + \lambda \mathbb{E}_{\hat{T} \sim P_i}(\|\nabla_{\hat{T}} D_\rho(\hat{T})\|_2 - 1)^2, \end{aligned} \quad (6.38)$$

where P_i is the distribution induced by uniform sampling on interpolation lines between independent samples of T and $Q(\xi)$, and λ is the gradient penalty coefficient. In this section we set $\lambda = 0.1$.

Once $G_\eta(x; \xi)$ is well-trained, it can be viewed as a map that transports a distribution of ξ to the functional distribution of u , and in particular, the standard multivariate Gaussian distribution to the learned functional prior of u . To estimate the posterior of u and f , we only need to sample the posterior distribution of ξ , and this can be done in the same spirit as B-PINNs.

Suppose we have the dataset $\mathcal{D} = \mathcal{D}_u \cup \mathcal{D}_f \cup \mathcal{D}_b$ defined in Equation 6.2 and 6.3 in the posterior estimation stage. Then, the likelihood for ξ can be written as:

$$\begin{aligned} P(\mathcal{D}|\xi) &= P(\mathcal{D}_u|\xi)P(\mathcal{D}_f|\xi)P(\mathcal{D}_b|\xi), \\ P(\mathcal{D}_u|\xi) &= \prod_{i=1}^{N_u} \frac{1}{\sqrt{2\pi\sigma_u^{(i)2}}} \exp\left(-\frac{(\tilde{u}_\eta(\mathbf{x}_u^{(i)}; \xi) - \bar{u}^{(i)})^2}{2\sigma_u^{(i)2}}\right), \\ P(\mathcal{D}_f|\xi) &= \prod_{i=1}^{N_f} \frac{1}{\sqrt{2\pi\sigma_f^{(i)2}}} \exp\left(-\frac{(\tilde{f}_\eta(\mathbf{x}_f^{(i)}; \xi) - \bar{f}^{(i)})^2}{2\sigma_f^{(i)2}}\right), \\ P(\mathcal{D}_b|\xi) &= \prod_{i=1}^{N_b} \frac{1}{\sqrt{2\pi\sigma_b^{(i)2}}} \exp\left(-\frac{(\tilde{b}_\eta(\mathbf{x}_b^{(i)}; \xi) - \bar{b}^{(i)})^2}{2\sigma_b^{(i)2}}\right). \end{aligned} \quad (6.39)$$

Combined with the prior $P(\xi) = (2\pi)^{-k/2} \exp(-\|\xi\|^2/2)$ where k is the dimensionality

of ξ , the posterior is obtained from Bayes' theorem:

$$P(\xi|\mathcal{D}) = \frac{P(\mathcal{D}|\xi)P(\xi)}{P(\mathcal{D})} \simeq P(\mathcal{D}|\xi)P(\xi), \quad (6.40)$$

With the posterior sampling approaches introduced in Section 6.2.2, e.g., HMC, we can sample from $P(\xi|\mathcal{D})$, denoted as $\{\xi^{(i)}\}_{i=1}^M$, and then obtain the posteriors of u and f from samples $\{\tilde{u}_\eta(\mathbf{x}; \xi^{(i)})\}_{i=1}^M$ and $\{\tilde{f}_\eta(\mathbf{x}; \xi^{(i)})\}_{i=1}^M$.

Comparing the posterior in this Section and in Section 6.2, the main modification is to replace the Bayesian neural network $\tilde{u}(\mathbf{x}; \theta)$ with a more general generator $G_\eta(\mathbf{x}; \xi)$. To get different functional priors, in $\tilde{u}(\mathbf{x}; \theta)$ we can only tune the architecture and activation functions, or the priors distributions for θ . Here in $G_\eta(\mathbf{x}; \xi)$, the prior for ξ is fixed, e.g., Gaussian, and we can tune η for different functional priors. But we remark that $\tilde{u}(\mathbf{x}; \theta)$ can also be viewed as a special case of the generator $G_\eta(\mathbf{x}; \xi)$, if we reparameterize θ with ξ and η , e.g., $\theta = \xi \odot \eta_1 + \eta_2$, $\eta = (\eta_1, \eta_2)$.

6.3.2 Different Generator Architectures for Learning Priors

In this section, we test with different architectures for $G_\eta(\mathbf{x}; \xi)$. In each architecture, the prior for ξ is set as the standard multivariate Gaussian distribution. For all neural networks we use hyperbolic tangent activation function.

The first architecture is induced from Bayesian neural networks. We set $G_\eta(\mathbf{x}; \xi) = \tilde{u}(\mathbf{x}; \xi \odot \eta_1 + \eta_2)$, $\eta = (\eta_1, \eta_2)$, where $\tilde{u}(\mathbf{x}; \theta)$ is a neural network with θ as parameters. The entries of η_1 and η_2 are initialized as 1 and 0, respectively. The input and hidden layers are scaled by the square root of layer dimensions

when multiplied with weight matrices. Since the dimensionality of ξ equals to the number of neural network parameters, here we test a relatively small feed forward neural network with two hidden layers, each of width 20, thus ξ has 481 dimensions.

The second architecture is similar to the generator in PI-GANs. We set $G_\eta(x; \xi)$ as a deep neural network taking the concatenation of x and ξ as input, and η as the neural network parameters. Here we test a feed forward neural network with three hidden layers, each of width 128.

The third architecture is inspired by Karhunen-Loève expansion 6.26. In particular, we set $G_\eta(x; \xi) = \tilde{g}_{\eta_1}(x) \cdot \tilde{h}_{\eta_2}(\xi) / \sqrt{d}$, where d is the dimensionality of ξ , $\tilde{g}_{\eta_1}$ and $\tilde{h}_{\eta_2}$ are two neural networks with η_1 and η_2 as parameters, respectively. Here we test $\tilde{g}_{\eta_1}$ and $\tilde{h}_{\eta_2}$ with 3 hidden layers, each of width 64, and output dimension the same as the dimension of ξ . We can see here $\tilde{g}_{\eta_1}(x)$ acts as the “basis” and $\tilde{h}_{\eta_2}(\xi)$ acts as the “random variables”. Note that $\tilde{h}_{\eta_2}(\xi)$ is not limited to be Gaussian, thus $G_\eta(x; \xi)$ can represent non-Gaussian stochastic processes.

Here we compare the above three architectures on the Poisson equation

$$-0.1\Delta u = f \tag{6.41}$$

with different priors for u and f .

In the first problem, we set the target priors for u as

$$u = \text{softplus}(A) \sin(\text{softplus}(\omega)x) \tag{6.42}$$

with $A \sim \mathcal{N}(1.5, 0.3^2)$ and $\omega \sim \mathcal{N}(3.0, 0.7^2)$. The prior for f can be obtained

correspondingly. We place two u -sensors at -1 and 1 and 21 equidistant f -sensors from -1 to 1, and generate 10000 snapshots to train the generators. The dimension for the input noise ξ is set as 10 for the second and third generator architecture, and the batch size is set as 128. The training takes 10^6 steps for all three generators. The target prior and learned priors are visualized in Figure 6.19. In the posterior estimation stage, we provide two observation data with noise size 0.2 for each of u and f , and use HMC to estimate the posteriors, which are visualized in Figure 6.20. We can see that for this case the learned priors and posteriors are close to the target ones.

Note that the estimated u posteriors always pass through (0,0), which is because the u priors always pass through (0,0). Such an informative prior for u can only be obtained through the PDE since we only have two u -sensors at -1 and 1. This example shows the capability of PI-GANs to learn an informative prior, which benefits the Bayesian uncertainty quantification, with the help of the PDE.

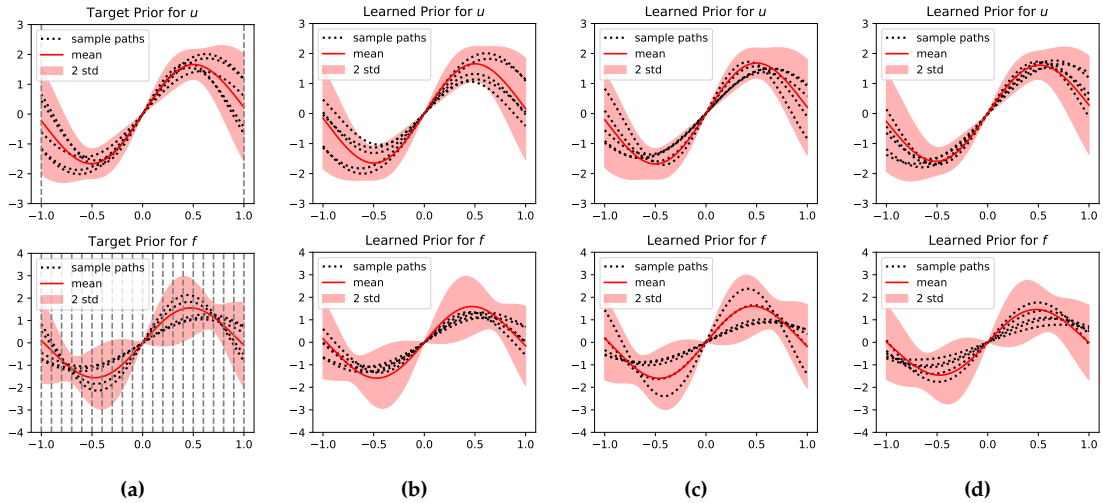


Figure 6.19: Target priors for u and f , as well as the learned priors with different generator architectures. (a) The target priors, the grey vertical dashed lines represent the locations of sensors. (b-d) The learned priors with the first, second, and third generator architecture, respectively. The red curves and shaded areas represent the mean and two standard deviations, while the black dashed curves represent the sample functions.

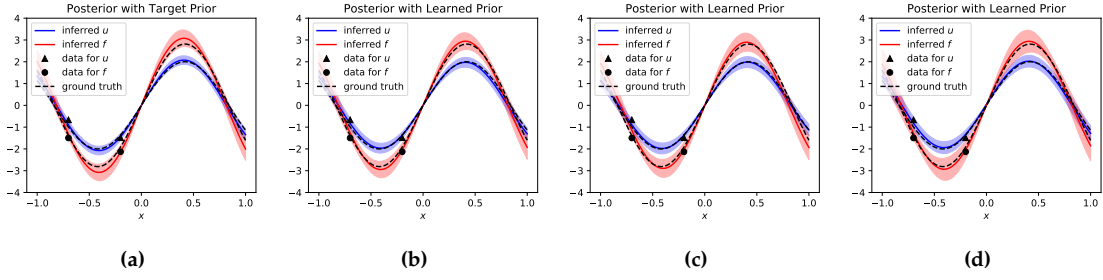


Figure 6.20: Posteriors for u and f , using the target priors and the learned priors with different generator architectures. (a) The posteriors using the target priors. (b-d) The posteriors using the learned priors, with the first, second, and third generator architecture, respectively. The curves and shaded areas represent the mean and two standard deviations.

In the second example, we set the target prior for u as a Gaussian process:

$$u \sim \mathcal{GP}(0, k(x, x'))$$

$$k(x, x') = \exp\left(-\frac{(x - x')^2}{2 \times 0.4^2}\right). \quad (6.43)$$

The prior for f can be obtained correspondingly. We place 41 u -sensors and 41 f -sensors equidistantly from -1 to 1, and generate 10000 snapshots to train the generators. The dimension for the input noise ξ is set as 40 for the second and third generator architecture, and the batch size is set as 1024. The training takes 10^6 steps for the first generator architecture, and 10^5 for the rest two. We visualize the covariance of the generated functions at x and y against the distance $|x - y|$ in Figure 6.21. In the posterior estimation stage, we provide two observation data for u and nine observation data for f , all with noise size 0.01, and use HMC to estimate the posteriors, which are visualized in Figure 6.22.

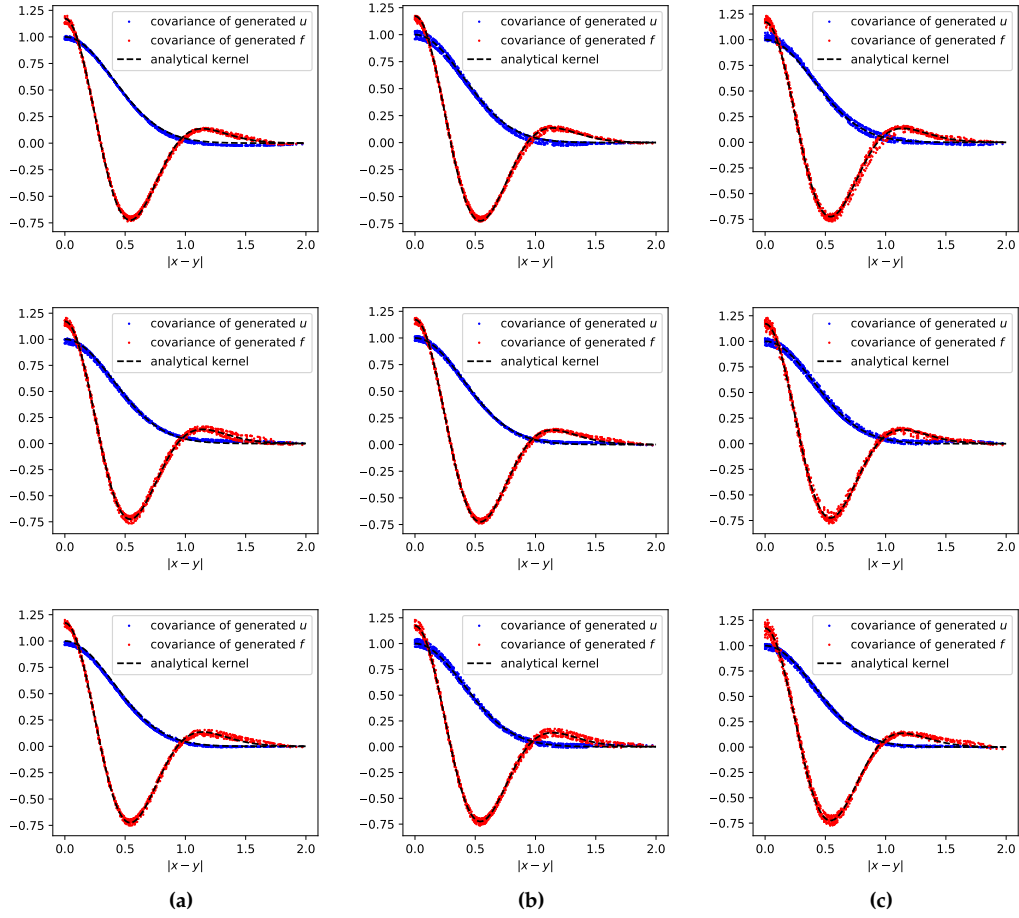


Figure 6.21: Covariances of learned priors at x and y against the distance $|x - y|$, with different generator architecture. (x, y) are randomly sampled from $[-1, 1]^2$. (a-c): The first, second, and third generator architecture, respectively. The three rows show the results with three random seeds.

For this case, although the covariances of the priors learned by the three generators match the covariances of the target prior (the analytical kernel), the estimated posteriors are pretty different. The posteriors given by the first architecture are reasonable in that the ground truth is mostly bounded by the two standard deviations, but the error and uncertainty are larger than the posterior with the target prior. The second architecture fails in that error is large, and the uncertainty is significantly underestimated. The third architecture outperforms the second one, but is not very robust.

The results here shows the influence of generator architectures. It is important to notice that even the learned prior are statistically consistent with the target prior, the corresponding estimated posterior could be inconsistent. We discuss this issue in the next section.

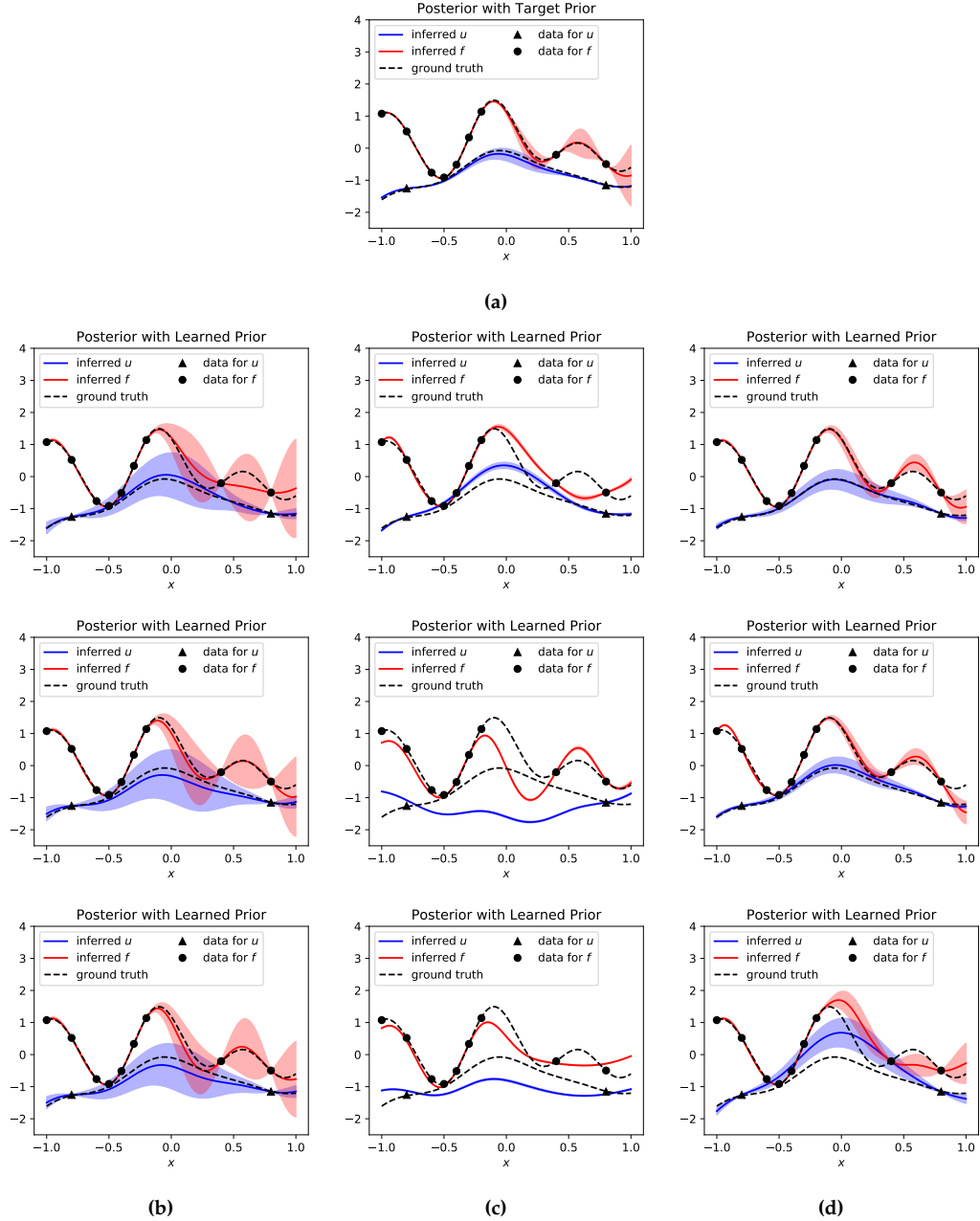


Figure 6.22: Posteriors for u and f using the target prior and learned priors. (a) The posteriors corresponding to the target Gaussian process prior. (b-d) The posteriors estimated with the priors learned by the first, second, and third generator architecture, respectively. The three rows show the results with three random seeds. The curves and shaded areas represent the mean and two standard deviations.

6.3.3 Why the Prior Learned by Wasserstein GANs Fails for Posterior Estimation

There are mainly two reasons:

- A series of generated multivariate distributions converge to the target distribution in Wasserstein distance does not implies that the statistics of the conditional distributions converge to the statistics of the target conditional distribution.
- Even if the conditional distributions converge to the target conditional distribution, the corresponding distributions in the input noise space could be multi-modal, and hard to be sampled from with HMC method.

Here we construct an example. We set the target distribution of T as $\mathcal{U}([0, 1]^2)$, the uniform distribution on $[0, 1]^2$, and the generated distributions Q_n as uniform distributions on the graph of $y = \arccos(\cos(n\pi x))/\pi, x \in [0, 1]$, for $n = 1, 2, \dots$. The input noise z could be sampled from $\mathcal{U}([0, 1])$ and the generator function is $(z, \arccos(\cos(n\pi z))/\pi)$. We visualize the generated distributions Q_n in Figure 6.23.

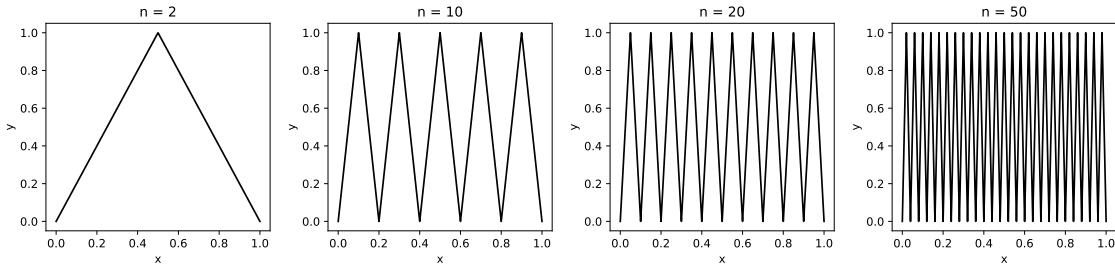


Figure 6.23: Illustration of the generated distributions Q_n for $n = 2, 5, 20, 50$. Q_n is the uniform distributions on the graph of the function.

It is not hard to see that Q_n converges to T in the Wasserstein distance. However,

for any n and for any $x \in [0, 1]$, the conditional distributions $Q_n(y|x)$ are Dirac delta distributions $\delta_{\arccos(\cos(n\pi x))/\pi}$, while the conditional distribution $T(y|x)$ is $\mathcal{U}([0, 1])$. On the other hand, for any $y \in (0, 1)$, while we can show that $Q_n(x|y)$ converge to $T(x|y)$, i.e., $\mathcal{U}([0, 1])$, the corresponding distributions for z are uniform discrete distributions with n modes.

In the above example the input noise dimension is less than the target distribution dimension, but even if the former is no less than the latter, it is still possible that some of the input noise dimensions are curled up so that the generated distributions still concentrate on low dimensional manifolds.

6.3.4 Improve the Learned Priors with Gradient Penalty for Generators

In the above illustrative example, the gradient of the generator function w.r.t. the input noise z increases as n goes to infinity. This inspires us to penalize the gradient of generator to improve the learned prior for posterior estimation.

Take the second problem in Section 6.3.2 as an example, where we have $\bar{N}_u = 41$ u -sensors and $\bar{N}_f = 41$ f -sensors to learn the prior. We define the gradient penalty as

$$L_{grad} = \mathbb{E}_{\xi} \frac{1}{\bar{N}_u + \bar{N}_f} \left(\sum_{i=1}^{\bar{N}_u} \|\nabla_{\xi} \tilde{u}_{\eta}(\bar{x}_u^{(i)}; \xi)\|^2 + \sum_{i=1}^{\bar{N}_f} \|\nabla_{\xi} \tilde{f}_{\eta}(\bar{x}_f^{(i)}; \xi)\|^2 \right), \quad (6.44)$$

where $\bar{x}_u^{(i)}$ and $\bar{x}_f^{(i)}$ are the locations of u -sensors and f -sensors. We augment the original generator loss L_G in Equation 6.38 with γL_{grad} to train the generator, where γ is the penalty weight. $\mathbb{E}_{\xi}$ in Equation 6.44 can be estimated by Monte Carlo, using the same batch of ξ as in the original generator loss function L_G in Equation 6.38.

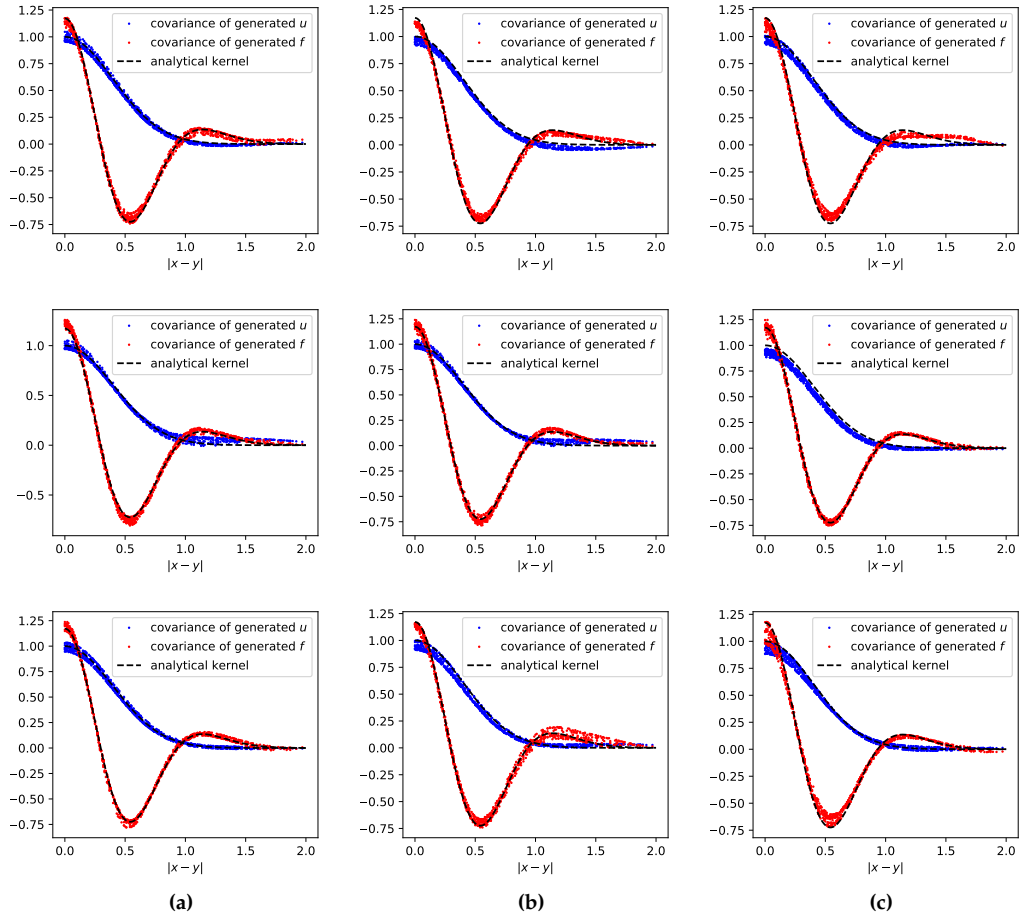


Figure 6.24: Covariances of learned priors at x and y against the distance $|x - y|$, with different gradient penalty coefficients γ for the second generator architecture. (x, y) are randomly sampled from $[-1, 1]^2$. (a-c): $\gamma = 0.1, 1.0, 10.0$, respectively. The three rows show the results with three random seeds.

In Figure 6.24 and Figure 6.25 we show the learned priors and corresponding estimated posteriors for using the second generator architecture with $\gamma = 0.1, 1.0, 10.0$. Here the batch size is reduced to 128 to reduce computational cost. The estimated posteriors show no improvements for $\gamma = 0.1$, but show the trend of improvement as we further increase γ . However, as γ increases, from the covariance we can also notice the inaccuracy of the learned prior, which is due to the bias introduced by the gradient penalty. Future research could investigate other regularization techniques for the generator to improve the training of GANs

to learn priors, for example, spectral normalization, which was originally proposed to regularize the discriminator [86] and later also applied to regularize the generator [150] in image generation tasks.

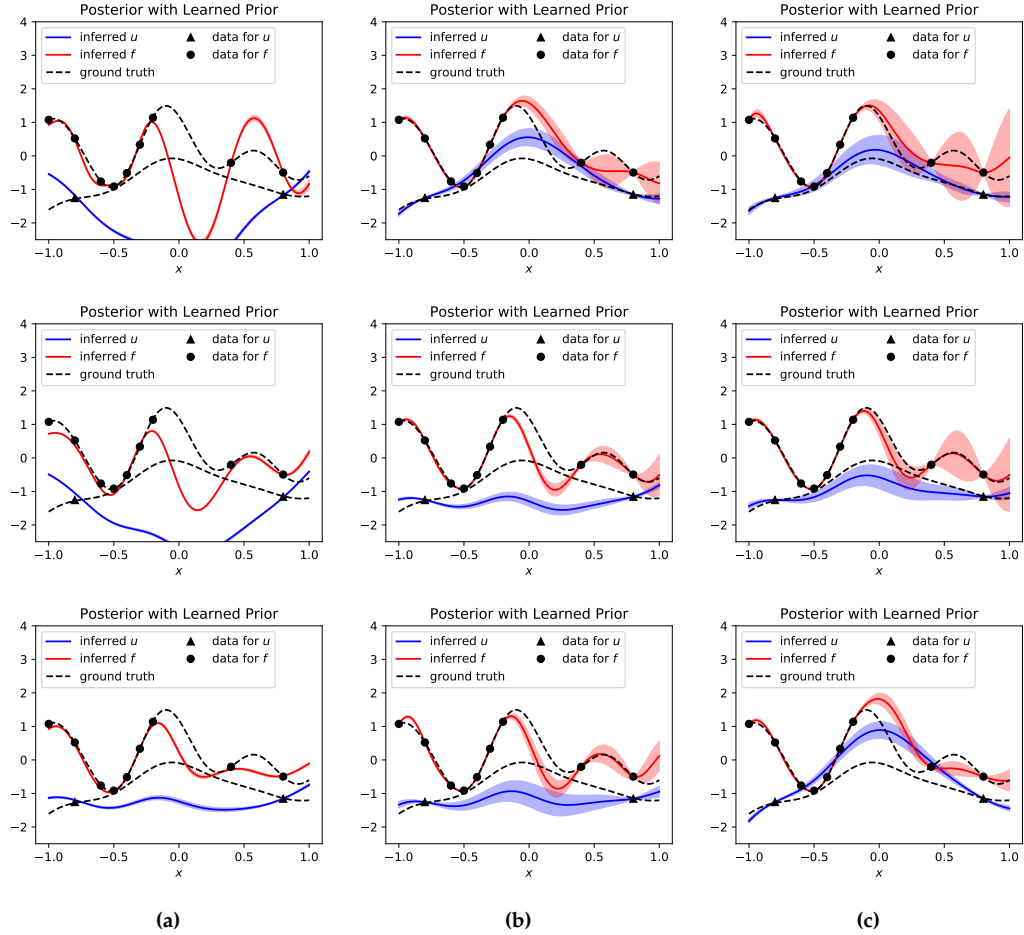


Figure 6.25: Posteriors for u and f using the learned priors, with different gradient penalty coefficients γ for the second generator architecture. (a-c) $\gamma = 0.1, 1.0, 10.0$, respectively. The three rows show the results with three random seeds. The curves and shaded areas represent the mean and two standard deviations.

6.3.5 Supplementary Material

Deep Normalizing Flow for Posterior Surrogates with the Learned Priors

While we can perform posterior estimation with HMC, it takes times for HMC to converge. In some scenarios, we may need online posterior estimation, and HMC is too slow to give timely results. In such scenarios, a posterior surrogate which takes the online observations as input and output the posterior samples, would be useful for fast posterior estimation.

Here we use deep normalizing flow models to learn posterior surrogates, and show a simple example as a proof of concept. The idea of using invertible neural networks for posterior surrogates is proposed in [99], here we use it in Bayesian inference of PDE problems. Specifically, we use the prior learned by the second architecture, in the first problem of Section 6.3.2. We then make observations at two u -sensors and two f -sensors, with the ground truth (A, ω) sampled from a hidden distribution ν , and the observation noise is $\mathcal{N}(0, 0.2^2)$. Note that ν can be different from the prior, here we set $A \sim \mathcal{U}([1.0, 2.0])$ and $\omega \sim \mathcal{U}([2.0, 4.0])$, A and ω are independent.

In Section 6.2.6, we already introduced how to train deep normalizing flow models to approximate the posterior distributions conditioned on a single set of observations. The method used here is similar. We list the differences and important hyperparameters as follows:

- Instead using $\nabla\varphi$ to represent the velocity, here we directly use a neural network φ to represent the velocity, aiming to reduce the computational

cost. In particular, φ is a ResNet [45] with 5 hidden layers, each of width 128, shortcut skipping one hidden layer. We set time span $T = 1$, and use 20 time steps, i.e. step size is 0.05.

- Apart from u and t , φ also takes the four observations of u and f as input.
- In each train step of φ , we randomly generate a set of noisy observations of u and f , and calculate the corresponding posterior used in the loss function of the flow model Equation 6.33. Note that the loss function is based on analytical unnormalized posterior densities, and we do not need to prepare posterior samples for the training.

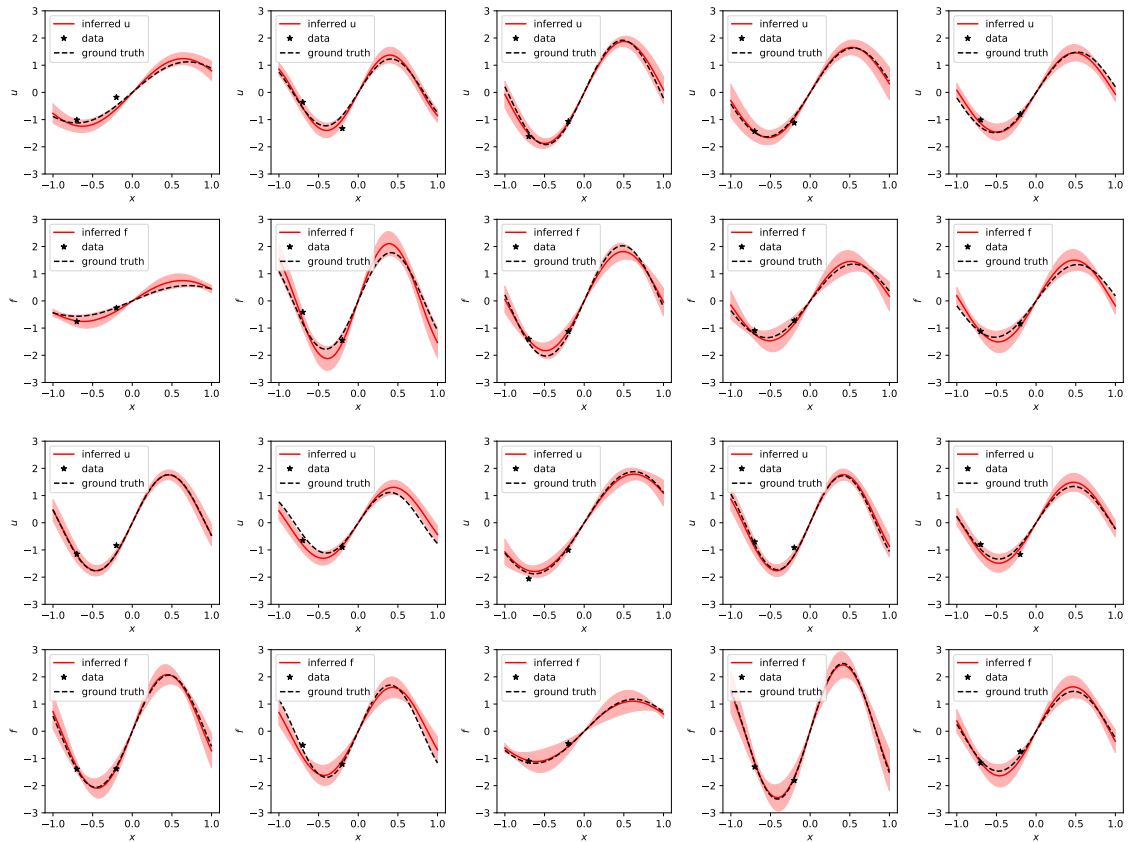


Figure 6.26: The posterior conditioned on ten sets of observations, predicted by the deep normalizing flow as a surrogate model. The first and third row are for u , while the second and forth row are for corresponding f . The red lines and shaded areas represent the mean and two standard deviations.

We train the flow model for 10^6 steps. Once trained offline, the surrogate model need no further training when predicting the posteriors with new sets of observations. It takes about 3 seconds to make predictions for 10 sets of observations on MacBook Pro 2018 with 2.9 GHz 6-Core Intel Core i9 CPU. We show the predicted posteriors in Figure 6.26, where we can see that the trained surrogate model gives reasonable predictions.

6.4 Summary of Chapter

There are many sources of uncertainty in data-driven PDE solvers, including uncertainty associated with noisy and gappy data, uncertainty associated with unknown parameters, and uncertainty associated with the type of PDE that models the target phenomena. In this chapter, we address the data uncertainty for solving forward and inverse PDE problems in the Bayesian framework, based on noisy data associated with the solution, source terms and boundary conditions.

In the first part of this chapter, in order to quantify uncertainty and improve the accuracy of PINNs, we propose Bayesian physics-informed neural networks (B-PINNs), using a Bayesian neural network for the solution with likelihood function induced from the PDE, combined with different estimators for the posterior, namely, the Hamiltonian Monte Carlo (HMC) method and the variational inference (VI). We conduct a comprehensive comparison among different methods, i.e., the B-PINNs with HMC, B-PINNs with VI (more specifically the mean field Gaussian approximation with Kullback-Leibler divergence minimization), and PINNs with dropout, which is also used to quantify the uncertainty of neural networks. Our experiments demonstrate good accuracy and robustness of B-PINN-HMC, but

B-PINN-VI usually gives unreasonable uncertainties, which could be attributed to the fact that the posterior distribution is approximated by a factorizable Gaussian distribution in this version of VI. The above findings are consistent with those reported in [34, 112, 144]. Moreover, dropout can hardly provide satisfactory uncertainty quantification, in agreement with [144].

While the choice of priors for B-PINNs may have a significant influence on the posterior predictions especially in the cases with small data, such choice of priors, including the architecture of neural networks and the prior distribution for the parameters etc., is a great challenge. In the second part of this chapter, we propose to use physics-informed GANs to learn priors for the solution. The learned prior can then be used in B-PINNs. We compared different architectures to represent stochastic processes for the solution, and show that even the learned prior are statistically consistent with the target prior, the corresponding estimated posterior could be inconsistent. We discussed the underlying reasons with an illustrative example, and propose to penalize the gradient of generator to improve the learned prior for Bayesian uncertainty quantification.

CHAPTER SEVEN

Conclusion

7.1 Summary

Motivated by the ubiquitous demand of leveraging both data and partial knowledge of physical laws for stochastic modeling and uncertainty quantification, in this dissertation, we developed a series of methods centered around generative adversarial networks (GANs) for physics-informed learning.

We firstly proposed physics-informed GANs to solve forward, inverse, and mixed problems of time-independent stochastic differential equations, using data collected from sensor measurements. Using neural networks for parameterizing the stochastic processes, we showed that physics-informed GANs can solve problems with high stochastic dimensions. We then extended the application of GANs to dynamic inference with observations of particle ensembles as data, namely “generative ensemble regression”. A physics-informed loss function is applied when inferring the nonlocal interactions in generative ensemble regression. A similar idea is also used in the development of the potential flow generator for GANs, which not only tries to transport the input distribution to the target one, but also aims to find the one with minimum L_2 transport cost. We showed the effectiveness of the potential flow generator in image translation tasks with unpaired training data. To address the oscillation and instability problem of GANs in generative ensemble regression tasks, we developed the measure-conditional discriminator for GANs. We showed that the measure-conditional discriminator is more robust than the vanilla one, and improves the stochastic dynamic inference in generative ensemble regression tasks. Finally, we introduced Bayesian uncertainty quantification for partial differential equations (PDEs) with physics-informed GANs. We first proposed Bayesian physics-informed neural networks to solve forward and inverse PDE problems with noisy and gappy data, and sub-

sequently proposed to use physics-informed GANs to learn functional priors for the Bayesian uncertainty quantification.

7.2 Other Generative Models

Apart from GANs, there are other deep generative models, here we list them and discuss their attributes. Future research could explore the possibility of applying these generative models for physics-informed learning.

Autoencoders [46] are deep learning models for dimensionality reduction. They introduce two neural networks: encoder and decoder. The former one is designed to translate a high-dimension vector into a corresponding low-dimensional code, while the latter one is designed to reconstruct the high-dimensional vector from the low-dimensional code. In general, autoencoder is trained by enforcing the composition of encoder and decoder to be an identity map. The variational autoencoder (VAE) [56], as a probabilistic version of autoencoder, translates the input vector into stochastic codes, and the decoder can be viewed as a generative model if one samples the codes and decodes them into reconstructed vectors. VAE and GANs can both generate synthetic data and provide latent representations of the given data. Currently the CIFAR-10 image generation benchmark shows that GANs-based models [51] outperforms VAE-based models [133], but the gap is very small, suggesting the future potential of VAE-based models.

Normalizing flow models [27, 19] are constructed by the composition of a sequence of invertible transformations. The most important attribute of normalizing

flow models is that the density of the data can be estimated via such invertible transformations. Therefore, they can be trained with density-related loss functions, and used in density-related tasks. Apart from generating synthetic data and provide latent representations as GANs and VAE, normalizing flow models can also perform density estimation for data, or sample from prescribed (unnormalized) density functions, which could be hard for common versions of GANs and VAE. In Chapter 6 we applied normalizing flow models in posterior sampling.

Deep autoregressive models formulate a high-dimensional probability distribution as the product of a series of conditional distributions. Autoregressive models can also estimate the density and generate synthetic data, but an important characteristic is that they generate high-dimensional samples dimension by dimension based on the conditional distributions, where the i -th component is conditioned on the first $i - 1$ components. They can be used for general data like images [129], but perhaps they are more suitable for very-high dimensional data with natural conditioning structure, for example, time series [92].

Finally, we should note that many models have been proposed as the hybrid of the deep generative models introduced above, for example, the hybrid of adversarial training and VAE [84], and the hybrid of GANs and normalizing flow models [41]. Such models could combine the advantages of different deep generative models.

7.3 Open Problems and Future Work

The application of physics-informed GANs in Chapter 2 and 6 relies on using neural networks to approximate stochastic processes. While there have been works on approximating stochastic processes using neural networks [126, 8], it is still not clear how to select the neural network architectures for a better approximation of specific stochastic processes, which requires future investigation.

When applying GANs to learn functional priors, we empirically found that WGAN-GP could fail with a certain generator architecture, although the learned prior is still statistically consistent with the target one. While we have discussed the underlying reason and proposed a method for improvement, more systematic studies are still required. Future research should investigate employing different adversarial loss functions, different generator architectures, or different regularization methods for the generator, to improve the learned prior for posterior estimation.

In Chapter 2 we use physics-informed GANs to solve stochastic differential equations and mainly consider the mean and standard deviation of the stochastic solution as our quantity of interest. For such purposes, we empirically observed that as we increase the stochastic dimension from 30 to about 120, the computational time grows in a low-order polynomial rate with the same number of snapshots as data. However, if we apply GANs for other purposes, for example, learning prior distributions from data for posterior estimation, the growth rate of data requirement and computational cost with respect to the stochastic dimension is not clear yet. Also, it is not clear if we can reduce the data requirement by designing neural networks with special architectures, or apply soft constraints

and regularization techniques to reflect our knowledge of the distribution. Such problems need future theoretical and empirical study.

We have shown in Chapter 3 that GANs can be applied to learning particle dynamics based on observations of ensembles. It is possible to apply similar methods to learn the interaction parameters for particle-based simulation methods. In particular, we will fit the target mass and velocity distributions coming from the analytical solution or other simulation methods that are accurate but expensive. We leave this promising research direction for future work.

Bibliography

- [1] L. Ambrosio and W. Gangbo. Hamiltonian ODEs in the Wasserstein space of probability measures. *Communications on Pure and Applied Mathematics: A Journal Issued by the Courant Institute of Mathematical Sciences*, 61(1):18–53, 2008.
- [2] L. Ambrosio, N. Gigli, and G. Savaré. *Gradient flows: in metric spaces and in the space of probability measures*. Springer Science & Business Media, 2008.
- [3] C. Archambeau, D. Cornford, M. Opper, and J. Shawe-Taylor. Gaussian process approximations of stochastic differential equations. *Journal of Machine Learning Research*, 1:1–16, 2007.
- [4] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223, 2017.
- [5] J. Bakarji and D. M. Tartakovsky. Data-driven discovery of coarse-grained equations. *Journal of Computational Physics*, page 110219, 2021.
- [6] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic

- differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–153, 2017.
- [7] M. G. Bellemare, I. Danihelka, W. Dabney, S. Mohamed, B. Lakshminarayanan, S. Hoyer, and R. Munos. The Cramer distance as a solution to biased Wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.
- [8] M. Belli, M. Conti, P. Crippa, and C. Turchetti. Artificial neural networks as approximators of stochastic processes. *Neural Networks*, 12(4-5):647–658, 1999.
- [9] J.-D. Benamou and Y. Brenier. A computational fluid mechanics solution to the Monge-Kantorovich mass transfer problem. *Numerische Mathematik*, 84(3):375–393, 2000.
- [10] J. Berg and K. Nyström. Data-driven discovery of PDEs in complex datasets. *J. Comp. Phys.*, 384:239–252, 2019.
- [11] D. Berthelot, T. Schumm, and L. Metz. BEGAN: Boundary equilibrium generative adversarial networks. *arXiv preprint*, page arXiv:1703.10717, 2017.
- [12] M. Betancourt. A conceptual introduction to Hamiltonian Monte Carlo. *arXiv preprint arXiv:1701.02434*, 2017.
- [13] I. Billionis. Probabilistic solvers for partial differential equations. *arXiv preprint*, page arXiv:1607.03526, 2016.
- [14] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *arXiv preprint arXiv:1505.05424*, 2015.
- [15] Y. Brenier. Polar factorization and monotone rearrangement of vector-valued functions. *Communications on pure and applied mathematics*, 44(4):375–417, 1991.

- [16] A. Brock, J. Donahue, and K. Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.
- [17] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine learning, Dynamical systems, and Control*. Cambridge University Press, 2019.
- [18] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, page 201517384, 2016.
- [19] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.
- [20] P. Cheridito, A. Jentzen, and F. Rossmannek. Efficient approximation of high-dimensional functions with deep neural networks. *arXiv preprint arXiv:1912.04310*, 2019.
- [21] A. Creswell and A. A. Bharath. Inverting the generator of a generative adversarial network. *IEEE transactions on neural networks and learning systems*, 30(7):1967–1974, 2018.
- [22] F. Cucker and S. Smale. Emergent behavior in flocks. *IEEE Transactions on automatic control*, 52(5):852–862, 2007.
- [23] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng. Training GANs with optimism. In *International Conference on Learning Representations*, 2018.
- [24] C. Daskalakis, A. Ilyas, V. Syrgkanis, and H. Zeng. Training GANs with optimism. In *International Conference on Learning Representations*, 2018.

- [25] I. Deshpande, Z. Zhang, and A. G. Schwing. Generative modeling using the sliced Wasserstein distance. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3483–3491, 2018.
- [26] M. Ding, C. Daskalakis, and S. Feizi. Subadditivity of probability divergences on Bayes-Nets with applications to time series GANs. *CoRR*, abs/2003.00652, 2020.
- [27] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016.
- [28] D. L. Donoho et al. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture*, 1(2000):32, 2000.
- [29] W. E, J. Han, and A. Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *arXiv preprint*, page arXiv:1706.04702, jun 2017.
- [30] O. Elerian, S. Chib, and N. Shephard. Likelihood inference for discretely observed nonlinear diffusions. *Econometrica*, 69(4):959–993, 2001.
- [31] B. Eraker. MCMC analysis of diffusion models with application to finance. *Journal of Business & Economic Statistics*, 19(2):177–191, 2001.
- [32] W. Fedus, I. Goodfellow, and A. M. Dai. MaskGAN: Better text generation via filling in the _____. In *International Conference on Learning Representations*, 2018.
- [33] R. Flamary and N. Courty. Pot python optimal transport library, 2017.
- [34] A. Y. Foong, Y. Li, J. M. Hernández-Lobato, and R. E. Turner. “In-Between”

- uncertainty in Bayesian neural networks. *arXiv preprint arXiv:1906.11537*, 2019.
- [35] Y. Gal and Z. Ghahramani. Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International Conference on Machine Learning*, pages 1050–1059, 2016.
- [36] W. Gangbo and R. J. McCann. The geometry of optimal transportation. *Acta Mathematica*, 177(2):113–161, 1996.
- [37] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [38] T. Graepel. Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. In *International Conference on Machine Learning*, pages 234–241, 2003.
- [39] A. Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.
- [40] I. Grondman, L. Busoniu, G. A. Lopes, and R. Babuska. A survey of actor-critic reinforcement learning: Standard and natural policy gradients. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):1291–1307, 2012.
- [41] A. Grover, M. Dhar, and S. Ermon. Flow-gan: Combining maximum likelihood and adversarial learning in generative models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.
- [42] G. L. Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. C. Farias, and A. Aspuru-Guzik. Objective-reinforced generative adversarial net-

- works (ORGAN) for sequence generation models. *arXiv preprint*, page arXiv:1705.10843, 2017.
- [43] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. Improved training of Wasserstein GANs. In *Advances in Neural Information Processing Systems*, pages 5767–5777, 2017.
- [44] J. Han, A. Jentzen, and E. Weinan. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.
- [45] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.
- [46] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.
- [47] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.
- [48] J. Jia and A. R. Benson. Neural jump stochastic differential equations. In *Advances in Neural Information Processing Systems*, pages 9843–9854, 2019.
- [49] I. Jolliffe. *Principal component analysis*. Springer, 2011.
- [50] T. Karras, T. Aila, S. Laine, and J. Lehtinen. Progressive growing of GANs for improved quality, stability, and variation. *arXiv preprint*, page arXiv:1710.10196, 2017.
- [51] T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila. Training generative adversarial networks with limited data. *arXiv preprint arXiv:2006.06676*, 2020.

- [52] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *arXiv preprint*, page arXiv:1707.03351, 2017.
- [53] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1857–1865. JMLR. org, 2017.
- [54] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint*, page arXiv:1412.6980, 2014.
- [55] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pages 10215–10224, 2018.
- [56] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. *arXiv preprint*, page arXiv:1312.6114, 2013.
- [57] A. Klenke. *Probability theory: a comprehensive course*. Springer Science & Business Media, 2013.
- [58] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.
- [59] I. E. Lagaris, A. C. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, Sept 1998.
- [60] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, Sept 2000.

- [61] Y. LeCun, Y. Bengio, and G. Hinton. Deep Learning. *Nature*, 521(7553):436–444, 2015.
- [62] Y. LeCun, C. Cortes, and C. Burges. MNIST handwritten digit database. *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist>, 2, 2010.
- [63] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690, 2017.
- [64] J. Lee, Y. Bahri, R. Novak, S. S. Schoenholz, J. Pennington, and J. Sohl-Dickstein. Deep neural networks as Gaussian processes. *arXiv preprint arXiv:1711.00165*, 2017.
- [65] J. Leygonie, J. She, A. Almahairi, S. Rajeswar, and A. Courville. Adversarial computation of optimal transport maps. *arXiv preprint arXiv:1906.09691*, 2019.
- [66] C.-L. Li, M. Zaheer, Y. Zhang, B. Poczos, and R. Salakhutdinov. Point cloud GAN. *arXiv preprint arXiv:1810.05795*, 2018.
- [67] J. Li and Y. M. Marzouk. Adaptive construction of surrogates for the Bayesian solution of inverse problems. *SIAM Journal on Scientific Computing*, 36(3):A1163–A1186, 2014.
- [68] X. Li, T.-K. L. Wong, R. T. Chen, and D. Duvenaud. Scalable gradients for stochastic differential equations. *arXiv preprint arXiv:2001.01328*, 2020.
- [69] X. Liang, Z. Hu, H. Zhang, C. Gan, and E. P. Xing. Recurrent topic-transition

- GAN for visual paragraph generation. *arXiv preprint*, page arXiv:1703.07022, 2017.
- [70] J. Liu, Z. Long, R. Wang, J. Sun, and B. Dong. RODE-Net: Learning ordinary differential equations with randomness from data. *arXiv preprint arXiv:2006.02377*, 2020.
- [71] X. Liu, T. Xiao, S. Si, Q. Cao, S. Kumar, and C.-J. Hsieh. Neural SDE: Stabilizing neural ODE networks with stochastic noise. *arXiv preprint arXiv:1906.02355*, 2019.
- [72] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3730–3738, 2015.
- [73] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.
- [74] T. Lucas, C. Tallec, Y. Ollivier, and J. Verbeek. Mixed batches and symmetric discriminators for gan training. In *International Conference on Machine Learning*, pages 2844–2853. PMLR, 2018.
- [75] X. Luo and A. Kareem. Bayesian deep learning with hierarchical prior: Predictions from limited and noisy data. *Struct. Saf.*, 84:101918, 2020.
- [76] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.
- [77] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*, 2013.

- [78] N. M. Mangan, J. N. Kutz, S. L. Brunton, and J. L. Proctor. Model selection for dynamical systems via sparse regression and information criteria. *P. Roy. Soc. A Math. Phys.*, 473(2204):20170009, 2017.
- [79] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Comput. Methods Appl. M.*, 360:112789, 2020.
- [80] Z. Mao, Z. Li, and G. E. Karniadakis. Nonlocal flocking dynamics: Learning the fractional order of pdes from particle simulations. *Communications on Applied Mathematics and Computation*, 1(4):597–619, 2019.
- [81] C. N. Mavridis, A. Tirumalai, and J. S. Baras. Learning interaction dynamics from particle trajectories and density evolution. In *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 1014–1019. IEEE, 2020.
- [82] X. Meng and G. E. Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *J. Comp. Phys.*, 401:109020, 2020.
- [83] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent PDEs. *arXiv preprint arXiv:1909.10145*, 2019.
- [84] L. Mescheder, S. Nowozin, and A. Geiger. Adversarial variational bayes: Unifying variational autoencoders and generative adversarial networks. In *International Conference on Machine Learning*, pages 2391–2400. PMLR, 2017.
- [85] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.
- [86] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

- [87] O. Mogren. C-RNN-GAN: Continuous recurrent neural networks with adversarial training. *arXiv preprint*, page arXiv:1611.09904, 2016.
- [88] M. A. Nabian and H. Meidani. A deep neural network surrogate for high-dimensional random partial differential equations. *arXiv preprint*, page arXiv:1806.02957, 2018.
- [89] R. M. Neal. *Bayesian learning for neural networks*, volume 118 of *Lecture Notes in Statistics*. Springer, 1996.
- [90] R. M. Neal et al. MCMC using Hamiltonian Dynamics. *Handbook of Markov Chain Monte Carlo*, 2(11):2, 2011.
- [91] X. Nguyen, M. J. Wainwright, and M. I. Jordan. Estimating divergence functionals and the likelihood ratio by convex risk minimization. *IEEE Transactions on Information Theory*, 56(11):5847–5861, 2010.
- [92] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.
- [93] F. Otto. Viscous fingering: an optimal bound on the growth rate of the mixing zone. *SIAM Journal on Applied Mathematics*, 57(4):982–990, 1997.
- [94] G. Pang, L. Yang, and G. E. Karniadakis. Neural-net-induced Gaussian process regression for function approximation and PDE solution. *arXiv preprint*, page arXiv:1806.11187, june 2018.
- [95] G. Pang, L. Yang, and G. E. Karniadakis. Neural-net-induced Gaussian process regression for function approximation and PDE solution. *J. Comp. Phys.*, 384:270–288, 2019.

- [96] D. V. Patel and A. A. Oberai. Quantifying uncertainty with {gan}-based priors, 2020.
- [97] D. V. Patel, D. Ray, H. Ramaswamy, and A. A. Oberai. Bayesian inference in physics-driven problems with adversarial priors.
- [98] D. Pfau, J. S. Spencer, A. G. Matthews, and W. M. C. Foulkes. Ab initio solution of the many-electron schrödinger equation with deep neural networks. *Physical Review Research*, 2(3):033429, 2020.
- [99] S. T. Radev, U. K. Mertens, A. Voss, L. Ardizzone, and U. Köthe. Bayesflow: Learning complex stochastic models with invertible neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.
- [100] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
- [101] M. Raissi. Forward-backward stochastic neural networks: Deep learning of high-dimensional partial differential equations. *arXiv preprint*, page arXiv:1804.07010, apr 2018.
- [102] M. Raissi and G. E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125–141, 2018.
- [103] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Inferring solutions of differential equations using noisy multi-fidelity data. *Journal of Computational Physics*, 335:736–746, 2017.
- [104] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Machine learning of linear

- differential equations using Gaussian processes. *J. Comp. Phys.*, 348:683–693, 2017.
- [105] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Machine learning of linear differential equations using Gaussian processes. *Journal of Computational Physics*, 348:683–693, 2017.
 - [106] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part I): Data-driven solutions of nonlinear partial differential equations. *arXiv preprint*, page arXiv:1711.10561, nov 2017.
 - [107] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Physics informed deep learning (part II): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint*, page arXiv:1711.10566, nov 2017.
 - [108] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Numerical gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198, 2018.
 - [109] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. *SIAM Journal on Scientific Computing*, 40(1):A172–A198, 2018.
 - [110] M. Raissi, P. Perdikaris, and G. E. 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.
 - [111] D. J. Rezende and S. Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015.

- [112] C. Riquelme, G. Tucker, and J. Snoek. Deep bayesian bandits showdown: An empirical comparison of bayesian deep networks for Thompson sampling. *arXiv preprint arXiv:1802.09127*, 2018.
- [113] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Sci. Adv.*, 3(4):e1602614, 2017.
- [114] L. Ruthotto, S. Osher, W. Li, L. Nurbekyan, and S. W. Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *arXiv preprint arXiv:1912.01825*, 2019.
- [115] L. Ruthotto, S. J. Osher, W. Li, L. Nurbekyan, and S. W. Fung. A machine learning framework for solving high-dimensional mean field game and mean field control problems. *Proceedings of the National Academy of Sciences*, 117(17):9183–9193, 2020.
- [116] T. Salimans, H. Zhang, A. Radford, and D. Metaxas. Improving GANs using optimal transport. *arXiv preprint arXiv:1803.05573*, 2018.
- [117] F. Santambrogio. {Euclidean, metric, and Wasserstein} gradient flows: an overview. *Bulletin of Mathematical Sciences*, 7(1):87–154, 2017.
- [118] S. Särkkä. Linear operators and stochastic partial differential equations in Gaussian process regression. In *International Conference on Artificial Neural Networks*, pages 151–158. Springer, 2011.
- [119] S. Särkkä, J. Hartikainen, I. S. Mbalawata, and H. Haario. Posterior inference on parameters of stochastic differential equations via non-linear Gaussian filtering and adaptive MCMC. *Statistics and Computing*, 25(2):427–437, 2015.
- [120] M. Schmidt and H. Lipson. Distilling free-form natural laws from experimental data. *Science*, 324(5923):81–85, 2009.

- [121] D. W. Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.
- [122] V. Seguy, B. B. Damodaran, R. Flamary, N. Courty, A. Rolet, and M. Blondel. Large-scale optimal transport and mapping estimation. *arXiv preprint arXiv:1711.02283*, 2017.
- [123] V. Seguy, B. B. Damodaran, R. Flamary, N. Courty, A. Rolet, and M. Blondel. Large-scale optimal transport and mapping estimation. In *Proceedings of the International Conference in Learning Representations*, 2018.
- [124] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.
- [125] A. M. Stuart. Inverse problems: A Bayesian perspective. *Acta Numerica*, 19:451–559, 2010.
- [126] C. Turchetti, M. Conti, P. Crippa, and S. Orcioni. On the approximation of stochastic processes by approximate identity neural networks. *IEEE Transactions on Neural Networks*, 9(6):1069–1085, 1998.
- [127] B. Tzen and M. Raginsky. Neural stochastic differential equations: Deep latent Gaussian models in the diffusion limit. *arXiv preprint arXiv:1905.09883*, 2019.
- [128] B. Tzen and M. Raginsky. Theoretical guarantees for sampling and inference in generative models with latent diffusions. *arXiv preprint arXiv:1903.01608*, 2019.
- [129] A. Van Oord, N. Kalchbrenner, and K. Kavukcuoglu. Pixel recurrent neural

- networks. In *International Conference on Machine Learning*, pages 1747–1756. PMLR, 2016.
- [130] C. Villani. *Optimal transport: old and new*, volume 338. Springer Science & Business Media, 2008.
 - [131] M. D. Vrettas, M. Oppen, and D. Cornford. Variational mean-field algorithm for efficient inference in large systems of stochastic differential equations. *Physical Review E*, 91(1):012148, 2015.
 - [132] E. Weinan 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.
 - [133] Z. Xiao, K. Kreis, J. Kautz, and A. Vahdat. Vaebm: A symbiosis between variational autoencoders and energy-based models. *arXiv preprint arXiv:2010.00654*, 2020.
 - [134] D. Xiu. *Numerical methods for stochastic computations: a spectral method approach*. Princeton University Press, 2010.
 - [135] L. Yan and T. Zhou. Adaptive multi-fidelity polynomial chaos approach to Bayesian inference in inverse problems. *J. Comput. Phys.*, 381:110–128, 2019.
 - [136] L. Yan and T. Zhou. An adaptive surrogate modeling based on deep neural networks for large-scale Bayesian inverse problems. *arXiv preprint arXiv:1911.08926*, 2019.
 - [137] K. D. Yang and C. Uhler. Scalable unbalanced optimal transport using generative adversarial networks. *arXiv preprint arXiv:1810.11447*, 2018.

- [138] L. Yang, C. Daskalakis, and G. E. Karniadakis. Generative ensemble-regression: Learning stochastic dynamics from discrete particle ensemble observations. *arXiv preprint arXiv:2008.01915*, 2020.
- [139] L. Yang and G. E. Karniadakis. Potential flow generator with L_2 optimal transport regularity for generative models. *arXiv preprint arXiv:1908.11462*, 2019.
- [140] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *SIAM Journal on Scientific Computing*, 42(1):A292–A317, 2020.
- [141] L.-C. Yang, S.-Y. Chou, and Y.-H. Yang. Midinet: A convolutional generative adversarial network for symbolic-domain music generation. *arXiv preprint*, page arXiv:1703.10847, 2017.
- [142] X. Yang, G. Tartakovsky, and A. Tartakovsky. Physics-informed kriging: A physics-informed Gaussian process regression method for data-model convergence. *arXiv preprint*, page arXiv:1809.03461, Sept. 2018.
- [143] Y. Yang and P. Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.
- [144] J. Yao, W. Pan, S. Ghosh, and F. Doshi-Velez. Quality of uncertainty quantification for Bayesian neural network inference. *arXiv preprint arXiv:1906.09686*, 2019.
- [145] Z. Yi, H. Zhang, P. Tan, and M. Gong. DualGAN: Unsupervised dual learning for image-to-image translation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2849–2857, 2017.

- [146] L. Yu, W. Zhang, J. Wang, and Y. Yu. Seqgan: Sequence generative adversarial nets with policy gradient. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.
- [147] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep sets. In *Advances in neural information processing systems*, pages 3391–3401, 2017.
- [148] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *arXiv preprint*, page arXiv:1809.08327, 2018.
- [149] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *J. Comp. Phys.*, 397:108850, 2019.
- [150] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena. Self-attention generative adversarial networks. In *International Conference on Machine Learning*, pages 7354–7363, 2019.
- [151] L. Zhang, J. Han, H. Wang, R. Car, and E. Weinan. Deep potential molecular dynamics: a scalable model with the accuracy of quantum mechanics. *Physical review letters*, 120(14):143001, 2018.
- [152] L. Zhang, L. Wang, et al. Monge-Ampère flow for generative modeling. *arXiv preprint arXiv:1809.10188*, 2018.
- [153] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin. Adversarial feature matching for text generation. *arXiv preprint*, page arXiv:1706.03850, 2017.

- [154] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin. Adversarial feature matching for text generation. In D. Precup and Y. W. Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 4006–4015, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.
- [155] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*, page arXiv:1703.10593, 2017.
- [156] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2223–2232, 2017.
- [157] 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.