University of California
Santa Barbara

Machine Learning and Computational Mathematics: Studies of Accelerating Polymer Phase Discovery and Finding Optimal Policies for a Pandemic

A dissertation submitted in partial satisfaction
of the requirements for the degree

Doctor of Philosophy
in
Mathematics

by

Yao Xuan

Committee in charge:

Professor Hector D. Ceniceros, Chair
Professor Carlos J. Garcia-Cervera
Professor Glenn H. Fredrickson

June 2021

The Dissertation of Yao Xuan is approved.

Professor Carlos J. Garcia-Cervera

Professor Glenn H. Fredrickson

Professor Hector D. Ceniceros, Committee Chair

June 2021

Machine Learning and Computational Mathematics: Studies of Accelerating Polymer Phase Discovery and Finding Optimal Policies for a Pandemic

by

Yao Xuan

I dedicate my dissertation work to my parents and grandparents for their everlasting love, and to all of my mentors for their invaluable guidance.

Acknowledgements

I am extremely grateful to my advisor, Prof. Hector D. Ceniceros, for his invaluable guidance on my academic research and a great influence on my life. Four years ago, he offered me a visionary suggestion and direction on this novel research, which perfectly matched my interest and built the stepping-stone for my future career. I will always remember how moved I was when he told me "You choose your career goal, and it is my job to help you achieve your career goal." I sincerely appreciate Prof. Ceniceros's exceptional expertise in research, excellent teaching and optimistic view of life, which led to a fantastic learning journey in the past several years of me.

I am deeply grateful to Prof. Glenn H. Fredrickson for his great expertise in SCFT theory and his support of my project. Prof. Glenn H. Fredrickson always sent me helpful feedbacks, where he suggested the ways to improve both in the big picture and small details. I am greatly grateful to my committee member, Prof. Carlos J. Garcia-Cervera, who also was the instructor of my first graduate course. I deeply appreciate his patience and guidance on graduate courses that I have taken with him, qualifying exams and my research project. I am also deeply grateful to Dr. Kris T. Delaney, who is greatly knowledgeable in computational science on SCFT models. Dr. Delaney always replied emails with thorough and careful explanations to answer my questions. This has helped greatly expanding my knowledge in such areas. I am grateful to Prof. Ruimeng Hu for her knowledgeable guidance and invaluable help on the stochastic control project. Prof. Hu's scientific writing never failed to impress me with her detailed edits. I also want to extend my sincere appreciation to my collaborators, Robert Balkin and Dr. Jiequn Han, for their great efforts and contributions in our projects. I am extremely fortunate to work with these professors and collaborators who demonstrated great quality of top scientific researchers, and the research presented in this dissertation would not have been

possible without their help.

I appreciate Prof. Xu Yang for offering valuable suggestions on how to start a PhD study in a foreign country and how to get over difficulties during the graduate school. He had a great impact on both my studies and life. I appreciate Medina Price for her kind help in the past 5 years. I also express my extreme gratitude to Lan Liu, who offered me invaluable help on my PhD study and building career goal. Without her support and encouragement, my PhD journey would not be so smooth.

I acknowledge support from National Science Foundation on the projects in this dissertation.

Finally, I am grateful to my families for their unconditional support. It was my childhood dream to pursue a PhD degree, and they always encouraged me to chase this dream. I deeply appreciate their everlasting love and support. I also would like to deeply thanks Lucy Lu for her love and company.

Curriculum Vitæ

Yao Xuan

Education

2021	Ph.D. in Mathematics (Expected), University of California, Santa Barbara.
2018	M.A. in Mathematics, University of California, Santa Barbara.
2016	B.S. in Mathematics, Harbin Institute of Technology.

Experience

2017 Summer	Instructor, University of California, Santa Barbara
2020 Summer	Machine Learning Engineer Intern, Facebook Inc., Menlo Park

Publications

Enhancing the Locality and Breaking the Memory Bottleneck of Transformer on Time Series Forecasting, Shiyang Li, Xiaoyong Jin, Yao Xuan, Xiyou Zhou, Wenhu Chen, Yu-Xiang Wang, Xifeng Yan, *Advances in Neural Information Processing Systems, 2019*.

Deep Learning and Self-Consistent Field Theory: A Path Towards Accelerating Polymer Phase Discovery, Yao Xuan, Kris T. Delaney, Hector D. Ceniceros, Glenn H. Fredrickson, *accepted by Journal of Computational Physics*, 2021.

Optimal Policies for a Pandemic: A Stochastic Game Approach and a Deep Learning Algorithm, Yao Xuan, Robert Balkin, Jiequn Han, Ruimeng Hu, Hector D. Ceniceros, *Mathematical and Scientific Machine Learning Conference (MSML2021)*.

Talks

Two Three-stage Methods for Extreme Multi-label Classification, *West Coast NLP Summit, October 2020*.

Deep Learning and Self-Consistent Field Theory: A Path Towards Accelerating Polymer Phase Discovery, *APS March Meeting, March 2021*.

Abstract

Machine Learning and Computational Mathematics: Studies of Accelerating Polymer Phase Discovery and Finding Optimal Policies for a Pandemic

by

Yao Xuan

Two directions of utilizing machine learning techniques in computational mathematics are explored with case studies on two real-world projects.

The first direction is in the supervised learning category where data generated from traditional numerical solvers are leveraged to train a machine learner for fast prediction or other specific task, like an inverse problem. A new framework that leverages data obtained from self-consistent field theory (SCFT) simulations with deep learning to accelerate the exploration of parameter space for block copolymers is presented in the first project. Deep neural networks are adapted and trained in Sobolev space to better capture the saddle point nature of the SCFT approximation. The proposed approach consists of two main problems: 1) the learning of an approximation to the effective Hamiltonian as a function of the average monomer density fields and the relevant physical parameters and 2) the prediction of saddle density fields given the polymer parameters. There is an additional challenge: the effective Hamiltonian has to be invariant under shifts (and rotations in 2D and 3D). Two different methods are developed to achieve the invariance: 1) a data-enhancing approach and an appropriate regularization and 2) a designed convolutional neural network (CNN). Generative adversarial networks (GAN) are connected with the SCFT models to predict the initial saddle density fields. These fields are selected from either GAN generations or the training set, and are fine-tuned by the deep Sobolev space-trained neural networks to get the final prediction.

The second direction is in the unsupervised learning category where machine learning techniques are developed to solve a previously unsolvable system. The proposed new approach embeds deep neural networks inside numerical schemes, builds suitable objective functions, and determines the neural network parameters by minimizing the objective functions. As a case study, a multi-region SEIR (Susceptible - Exposed - Infectious - Recovered) model based on stochastic differential game theory is proposed. This model's aim is to aid in the formulation of optimal regional policies for infectious diseases. Specifically, the standard epidemic SEIR model is enhanced by taking into account the social and health policies issued by multiple region planners. This enhancement makes the model more realistic and powerful. However, it also introduces a formidable computational challenge due to the high dimensionality of the solution space brought by the presence of multiple regions. This significant numerical difficulty of the model structure motivates a generalization of the deep fictitious algorithm introduced in [Han and Hu, MSML2020, pp.221–245, PMLR, 2020]. As a result, an improved algorithm, which uses deep neural networks to approximate the value function and control parameters and that builds objective functions based on the error of the approximated value function at the terminal time is developed. The proposed extended model and the new deep learning-based algorithm are applied to study the COVID-19 pandemic in three states: New York, New Jersey and Pennsylvania. The results show the effects of the lockdown/travel ban policy on the spread of COVID-19 for each state and how their policies affect each other.

Contents

Chapter 1

Introduction

The past two decades have witnessed the great development of machine learning techniques and the applications of machine learning in many scientific fields. Machine learning (ML), part of artificial intelligence (AI), is the study of computational algorithms based on mathematics, statistics and computational science to solve problems by computers. Computational mathematics focuses on computational algorithms or numerical methods and their application and analysis. Computational mathematics plays an important role in Engineering and Science realms by providing numerical algorithms and associated software for solving a spectrum of problems in ordinary differential equations (ODEs), differential algebraic equations (DAEs), and partial differential equations (PDEs)[1]. Since most of computational mathematics algorithms are based on computers, the revolution of computational power brought by machine learning naturally might become a driving force in advancement of computational mathematics. The main topic of this dissertation is to explore this new marriage of computational mathematics and machine learning on algorithms and applications in scientific fields.

Applied mathematics relies often on numerical methods to find approximations of solutions to intricate models of a given system. However, in many instances, these com-

putations can quickly become prohibitively expensive. In order to utilize machine learning to improve the efficiency of numerical methods or to solve previously intractable problems, we propose two innovative general directions and validate them with real-world problems.

The first direction is to learn the model and underlying physics through data generated by numerical simulations. The mathematical models translate the physics into mathematical equations. The data generated by a few numerical simulations contain information of both the physics captured by the model and the relation between the relevant variables. Thus, learning a map between the model's variables from data by machine learning methods can provide an economical and accurate approach to accelerate scientific computation and lead to an expedited prediction of the model's unknown variables. In the subsequent prediction, numerically solving complicated systems is replaced by the evaluation of the learner, a new surrogate for the model, which will significantly reduce computation time. Moreover, the computation of intermediate variables is skipped and this further reduces both computational complexity and time. This new supervised learning-based approach offers a the bridge between the traditional mathematical models and a fast machine learning predictor.

The second direction is to solve problems that were previously intractable by traditional numerical methods. Numerical methods for high dimensional differential equations and/or coupled systems often suffer from low computational efficiency and instability. The lack of more effective numerical methods make some problems unsolvable and it becomes impossible to generate training data. Suppose we are trying to solve for $u(t)$ from a system $F(u(t)) = 0, t \in [0, T]$. We could approximate $u(t)$ by a neural network $NN(t)$, discretize $F(NN(t))$ and implement simulation, trying to minimize say $g(NN()) = \|F(NN(t))\|$ in an L_p norm. This type of method can be viewed as an upgraded version of "the method of undetermined coefficients" based on machine learn-

ing. Instead of taking a function basis like polynomials or trigonometric functions in the method of undetermined coefficients, we use the composition of linear (more precisely affine) functions and nonlinear activation functions to simulate the system and search for the coefficients which minimize a messure of the error by stochastic gradient methods. Much of the work in this area focuses on stochastic differential equation systems and uses Monte-Carlo simulation to estimate the expectation of variables of interest. This type of methods can be broadly considered as unsupervised learning-based because no data is generated in advance of training.

This thesis contains efforts in both directions with the design of new algorithms and their application in real-world applications. The first project merges polymer self-consistent field theory (SCFT) and deep learning algorithms. We generate data using traditional SCFT computational methods and try to learn the physics from the data. This is Direction 1 mentioned above, where the machine learning algorithm learns the model from numerical data. We propose to train machine learners in Sobolev space and then apply a gradient-based method to search for local minimizers of the learner. This framework could be applied to general optimization/inverse problems. We specifically apply this new framework to quickly predict saddle density fields in SCFT to accelerate polymer phase discovery. To keep the true relation between density fields and the Hamiltonian, we also develop techniques to equip machine learning learners with rotation/translation invariance. The learner trained by the SCFT data matches accurately the direct SCFT results and is much more efficient to evaluate.

The second project is to solve previously unsolvable high dimensional problems by using machining learning method to approximate the solution of a system to model a pandemic. The model is based on the stochastic game approach to explore an optimal control for the pandemic. This is the aforementioned Direction 2, where we use a deep neural network to approximate the solution of a high dimensional system, plug it into

the numerical simulation of the system, determine the neural network parameters by minimizing the loss on some variables after simulation. The mathematical model is SEIR (Susceptible - Exposed - Infectious -Recovered). The corresponding Hamilton-Jacobi-Bellman (HJB) and Backward Stochastic Differential Equation (BSDE) systems are high dimensional and are a formidable challenge for traditional numerical methods. In this thesis, we approximate the value function and the optimal control by deep neural networks. We discretize the BSDE system, simulate the system with the neural network-approximated solution, and then search for the parameters of the solution by minimizing the loss at the terminal time. The numerical solution satisfies all the constraints of the system and provides a good approximation to the true solution.

The rest of the dissertation is organized as follows. The first project on accelerating polymer discovery based on self-consistent field theory leveraged with machine learning is included in Chapter 2 and Chapter 3. Chapter 2 introduces the self-consistent field theory and the design of machine learning algorithms to accelerate the computations for one spatial dimension. Chapter 3 develops machine learning algorithms to accelerate the computations for polymer phase discovery in 2D or higher spatial dimensions.

1.1 Permissions and Attributions

1. The content of Chapter 2 and Appendix A is the result of a collaboration with Kris T. Delaney, Hector D. Ceniceros and Glenn H. Fredrickson, and has been accepted by *Journal of Computational Physics* ("Deep Learning and Self-Consistent Field Theory: A Path Towards Accelerating Polymer Phase Discovery").

2. The content of Chapter 3 is the result of a collaboration with Kris T. Delaney, Hector D. Ceniceros and Glenn H. Fredrickson, and is in preparation for submission.

3. The content of Chapter 4 and Appendix B is the result of a collaboration with Robert Balkin, Jiequn Han, Ruimeng Hu and Hector D. Ceniceros, is to be appeared in the (conference) *Mathematical and Scientific Machine Learning 2021* and is to be published at *Proceedings of Machine Learning Research* ("Optimal Policies for a Pandemic: A Stochastic Game Approach and a Deep Learning Algorithm").

Chapter 2

Deep Learning and Self-Consistent Field Theory: 1D Polymer Phase Discovery

2.1 Introduction

Numerical simulations using self-consistent field theory (SCFT) are a valuable method to study the energetics and structure of polymer phases [2, 3, 4]. However, these computations are generally expensive. Relaxation methods to the SCFT (saddle point) solutions are slow to converge and each iteration is costly as it requires the solution of one or several modified diffusion (Fokker-Planck) equations [2, 5, 6]. There are some recent attempts to combine SCFT with machine learning. Nakamura [7] proposed to predict polymer phase types by a neural network with a theory-embedded layer that captures the characteristic features of the phase via coarse-grained mean-field theory. Wei, Jiang, and Shi [8] introduced a neural network approach as a solver to the SCFT modified diffusion equations to accelerate the computation of the mean fields. In this work we leverage techniques

of machine learning to obtain fast and accurate predictions of SCFT solutions after a suitable map from parameter space and average monomer density is learned. This new approach, which merges computer-simulated data with supervised learning, works for a large range of parameters after one single training and has the potential to dramatically accelerate the discovery and study of new polymer phases.

The proposed approach consists of two main problems: 1) the learning of an approximation to the effective Hamiltonian as a function of the average monomer density fields and the relevant physical parameters and 2) the prediction of saddle density fields given the polymer parameters. There is an additional challenge: the effective Hamiltonian has to be invariant under shifts (and rotations in 2D and 3D). A data-enhancing approach and an appropriate regularization are introduced to effectively achieve said invariance.

The rest of the chapter is organized as follows. In Section 2.4, we propose the neural network-based method to approximate H in Sobolev space. By using a Sobolev metric we are able to predict both H and its gradient (thermodynamic force) simultaneously. Additional properties are added to the neural network based on the periodic shift invariance of the modeled system and the mathematical existence of the desired neural network is rigorously proved. Section 2.6 is devoted to a summary of numerical results for an AB diblock system and for an AB_3 mikto-arm star system, both with density-field variations in only one space dimension. Finally, some concluding remarks, including the prospects for extension to higher spatial dimensions, are given in Section 2.7.

2.2 The SCFT AB Diblock Copolymer Model

To describe the proposed approach in more detail, we discuss the self-consistent field theory on an incompressible AB diblock copolymer melt. The effective Hamiltonian of

an AB diblock copolymer melt is expressed in terms of two fields, w_A and w_B [2]

$$H[w_A, w_B] = \int [-(w_A + w_B)/2 + (w_B - w_A)^2/(4\chi N)]d\mathbf{r} - V \ln Q[w_A, w_B], \tag{2.1}$$

where χ is the Flory parameter and measures the strength of binary contacts, N is the copolymer degreee of polimerization, and V is the system volume. $Q[w_A, w_B]$ is the partition function for a single copolymer with field w_A acting on the A block and field w_B acting on the B block. This functional can be evaluated by the formula

$$Q[w_A, w_B] = \frac{1}{V} \int q(\mathbf{r}, 1; [w_A, w_B]), \tag{2.2}$$

where $q(\mathbf{r}, s; [w_A, w_B])$ is the copolymer propagator (s is the contour variable which parametrizes a copolymer chain) which satisfies the Fokker-Planck equation (often referred to as the modified diffusion equation)

$$\frac{\partial q}{\partial s} = \nabla^2 q - \psi q, \quad q(\mathbf{r}, 0; [w_A, w_B]) = 1. \tag{2.3}$$

Here,

$$\psi(\mathbf{r}, s) = \begin{cases} w_A(\mathbf{r}), & 0 \leq s \leq f, \\ w_B(\mathbf{r}), & f < s \leq 1, \end{cases} \tag{2.4}$$

where f is the average volume fraction of type A blocks. The SCFT solution corresponds to saddle points of H, where H is a minimum with respect to the chemical potential-like field

$$w_-(\mathbf{r}) \equiv \frac{1}{2}[w_B(\mathbf{r}) - w_A(\mathbf{r})] \tag{2.5}$$

and a maximum with respect to pressure-like field

$$w_+(\mathbf{r}) \equiv \frac{1}{2}[w_B(\mathbf{r}) + w_A(\mathbf{r})]. \tag{2.6}$$

To find the saddle point fields, via gradient descent (for w^-) and gradient ascent (for w^+) we need to evaluate the first variation ("gradient") of H. This can be done in terms of monomer density fields ρ_A and ρ_B:

$$\begin{aligned} \frac{\delta H[w_+, w_-]}{\delta w_+(\mathbf{r})} &= \rho_A(\mathbf{r}; [w_+, w_-]) + \rho_B(\mathbf{r}; [w_+, w_-]) - 1, \\ \frac{\delta H[w_+, w_-]}{\delta w_-(\mathbf{r})} &= \frac{2}{\chi N} w_- + \rho_B(\mathbf{r}; [w_+, w_-]) - \rho_A(\mathbf{r}; [w_+, w_-]). \end{aligned} \tag{2.7}$$

In turn, ρ_A and ρ_B are computed using the Feynmman-Kac formulas

$$\begin{aligned} \rho_A(\mathbf{r}; [w_+, w_-]) &= \frac{1}{Q[w_+, w_-]} \int_0^f q(\mathbf{r}, s; [w_+, w_-]) q^\dagger(\mathbf{r}, 1-s; [w_+, w_-]) ds, \\ \rho_B(\mathbf{r}; [w_+, w_-]) &= \frac{1}{Q[w_+, w_-]} \int_f^1 q(\mathbf{r}, s; [w_+, w_-]) q^\dagger(\mathbf{r}, 1-s; [w_+, w_-]) ds, \end{aligned} \tag{2.8}$$

where the propagator $q^\dagger$ accounts for the lack of head-to-tail symmetry of the diblock. Analogously to q, $q^\dagger$ satisfies the equation,

$$\frac{\partial q^\dagger}{\partial s} = \nabla^2 q^\dagger - \psi^\dagger q^\dagger, \quad q^\dagger(\mathbf{r}, 0; [w_A, w_B]) = 1, \tag{2.9}$$

where

$$\psi^\dagger(\mathbf{r}, s) = \begin{cases} w_B(\mathbf{r}), & 0 \le s < 1 - f, \\ w_A(\mathbf{r}), & 1 - f \le s \le 1. \end{cases} \tag{2.10}$$

A typical SCFT computation requires hundreds or thousands of evaluations of (2.7) and hence of solutions to the Fokker-Planck equations. This makes polymer SCFT simulations

very expensive.

2.3 Problem Definition and General Strategy

In this section, we formulate the problems from the bottleneck of SCFT models and describe the general strategy to solve the problems based on machine learning approach. As shown in Section 2.2, in the model of incompressible AB diblock copolymer melt, the relevant parameters are χN (χ is the Flory parameter and N is the copolymer degree of polymerization), which measures the strength of segregation of the two components, L, the cell length in units of the unperturbed radius of gyration R_g, and f, the volume fraction of component A. Let $\rho = \rho_A$ be the local average monomer density of blocks A. Then, the framework we propose consists of solving the following two problems:

- **Problem 1**: Learn a map $(\chi N, L, f, \rho) \longmapsto \tilde{H}(\chi N, L, f, \rho)$, where $\tilde{H}$ is an accurate approximation of the field-theoretic intensive Hamiltonian H, the Helmholtz free-energy per chain at saddle points.

- **Problem 2**: For specific values χN^*, L^*, f^*, find accurately and efficiently the density field ρ^* that minimizes $\tilde{H}$.

Once $\tilde{H}$, a surrogate for the effective Hamiltonian at the saddle points, is learned (Problem 1), the procedure to solve Problem 2 can be expediently applied to screen the parameter space for new phase candidates.

To solve the first problem we recast H as

$$H(\chi N, L, f, \rho) = \chi N f - \frac{\chi N}{L^d} \int \rho^2 dr + R(\chi N, L, f, \rho), \tag{2.11}$$

by extracting the leading (quadratic) interaction term, and focus on learning, via neural networks, the remainder or residual term R, which contributes to polymer entropy, but

not enthalpy. In Eq. (2.11), d is the spatial dimension. The rationale for splitting H is simply that we know the exact expression for the enthalpic part of H, and it is both local and economical to compute while the entropic part is not known in closed form and is computationally expensive. We present numerical results in Appendix A.1 that show learning only the entropic part R produces superior results for the predicted density field ρ^* than those obtained by learning the full functional H.

Note that in an auxiliary field theory the effective Hamiltonian H, which coincides with the free energy at the SCFT saddle point, is a functional of one or more potential-chemical-like fields and one pressure-like field. The ρ dependence is generally integrated out using Gaussian integral identities and ρ is instead evaluated from these auxiliary fields. Here, we seek instead to learn directly an approximation of H, or more precisely of R, as a function of ρ.

Both Problem 1 and Problem 2 are challenging for the following reasons. First, the training data set for machine learning is generated from SCFT simulations, which produce physically meaningful information only at the saddle point. Thus, we only know H at the saddle point solutions, a reduced set of a much larger complex manifold. But ultimately, we would like to find a minimizer of H over all possible ρ fields, including both saddle points and non-saddle points. Therefore, additional assumptions about the map are necessary. Second, the input is very high dimensional; the density field ρ becomes a large vector whose components correspond to values of ρ at regular mesh points in physical space. Consequently, even with a successfully learned map in Problem 1, Problem 2 is still a formidable optimization problem due to its high dimensionality. Moreover, H can have many local minima, thus finding a global minimum is a difficult task. Finally, the learned map $\tilde{H}$ needs to be invariant under translations (and rotations in 2D and 3D) and this poses an additional challenge to the map-learning process.

In this work, we propose to learn a map from $(\chi N, L, f, \rho)$ to the free energy, using

SCFT-generated data, which is a good approximation of H and its gradient around saddle points and is invariant under spatial shifts of the density field. This is achieved by constructing a novel deep neural network in Sobolev space. Then, we pre-screen candidates for minimizers of the learned map to find, via gradient descent, predictors of new SCFT saddle points. In this chapter, we focus on one-dimensional (in physical space) systems to allow for a thorough exploration and development of the proposed methodology. Extension to higher dimensional systems are discussed in Chapter 3.

2.4 The Neural Network Model

We describe in this section the deep neural network trained in Sobolev space that we propose to accurately describe H and its gradient in the vicinity of field-theoretic saddle points.

2.4.1 Methodology

Henceforth, we use $x = (\chi N, L, f, \rho)$ to denote all parameters of the model, which for concreteness is an incompressible diblock copolymer melt. Here, ρ is a vector whose components are the values of the local average density field ρ at the spatial grid points. We need to learn a map $(\chi N, L, f, \rho) \longmapsto \tilde{H}(\chi N, L, f, \rho)$ which approximates the effective Hamiltonian H of the field theory at the saddle points.

Suppose $M = \{(x_1, H_1), (x_2, H_2), ..., (x_{N_T}, H_{N_T})\}$ is the training set of size N_T generated by SCFT simulations, where $x_i = (\chi N_i, L_i, f_i, \rho_i)$ represents the i_{th} training point and H_i is the corresponding H value (free energy) for the i_{th} training point.

As mentioned in the introduction, we start by rewriting H as

$$H(x) = \chi N f - \frac{\chi N}{L} \int \rho^2 dr + R(x), \tag{2.12}$$

and learn the entropic remainder function R. An appropriate model for this task is one that matches both the functional value of R and its gradient (first variation in the continuum case) because at a saddle point ρ^*, necessarily $\frac{\delta H}{\delta \rho}|_{\rho=\rho^*} = 0$. We will call NN the approximation to R obtained via a neural network approach that we specify in detail below. Hence, at the end, the approximate effective Hamiltonian map has the expression

$$\tilde{H}(x) = \chi N f - \frac{\chi N}{L} \int \rho^2 dr + NN(x). \tag{2.13}$$

In a neural network, there are many parameters in the form of weights and biases. These parameters are determined by minimizing a cost or loss function, which is defined based on the desired properties of the neural network. For our problem, we first choose a basic version of cost function as

$$C(\alpha) = \sum_{i=1}^{N_T} (\tilde{H}(x_i) - H_i)^2 + \beta \sum_{i=1}^{N_T} \|\nabla_\rho \tilde{H}(x_i)\|^2, \tag{2.14}$$

where α refers to the all the parameters in the neural network (hidden in the structure of $\tilde{H}$) and β is a parameter that controls the size of the penalty term $\sum_{i=1}^{N_T} \|\nabla_\rho \tilde{H}(x_i)\|^2$, which favors vanishing gradients at training points and adds smoothness to the learned map and prevents overfitting. More importantly, this regularizing term effectively enforces that the gradient of the learned $\tilde{H}$ approximates the zero vector at the training points (all training points are saddle points of H). It is important to emphasize that the learned map will ultimately match accurately both the function value of H and its vanishing gradient at training points. This property of the model represents a new approach to predict SCFT saddle points.

Note that for each data point (x_i, H_i), in view of (2.12) and (2.13), NN should match

$$R_i = H_i - \chi N_i f_i + \frac{\chi N_i}{L_i} \int \rho_i^2 dr. \tag{2.15}$$

Moreover, after discretizing in space we have

$$\nabla_\rho \tilde{H}(x_i) = -\frac{2\chi N_i}{L_i} \Delta r \, \rho_i + \nabla_\rho NN(x_i), \tag{2.16}$$

where Δr is the spatial mesh size. Then, we can rewrite the cost function (2.14) in terms of the residual map NN as

$$C(\alpha) = \sum_{i=1}^{N_T} (NN(x_i) - R_i)^2 + \beta \sum_{i=1}^{N_T} \|\nabla_\rho NN(x_i) - \frac{2\chi N_i}{L_i} \Delta r \, \rho_i\|^2. \tag{2.17}$$

From Eq. (7), the cost function we seek to minimize is just the distance between the predicted map, NN, and the actual map we are trying to approximate, R, in the sense of the Sobolev norm. The existence of a neural network to approximate a map to a desired accuracy has been established for different activation functions [9, 10]. Thus, a natural question is: does there exist a neural network that approximates both the map and its gradient, i.e. in the sense of Sobolev norm? The answer is yes and our approach is built on this solid theoretical foundation.

A single hidden layer feedback network constructs functions of the form

$$g(x) = \sum_{j=1}^{q} \omega_j \, G(\theta_j \cdot \tilde{x}), \tag{2.18}$$

where G is an activation function (e.g. sigmoid, softmax, ReLU, etc.), $\tilde{x} = (1, x)$, ω_j represents hidden-to-output layer weights and the vectors θ_j (of dimension equal to the dimension of x plus 1) represent input-to-hidden layer weights. Collectively, these pa-

rameters are the vector α appearing in the cost or loss function (2.17). If this class of functions is dense in a functional space F under some specific metric, then for any $f \in F$, there is a neural network of the form (2.18) that approximates f to a given accuracy in that metric. Hornik, Stinchcombe, and White [11] proved this is the case with the Sobolev norm. This result guarantees the existence of a single-hidden-layer neural network that approximates both a functional and its functional gradient simultaneously and provides the theoretical foundation for our proposed cost function (2.17). Moreover, there also exists, as proved by Czarnecki, Osindero, Jaderberg, Swirszcz, and Pascanu [12], a single layer neural network that matches both the functional and its functional gradient with *zero training loss*. That is, theoretically, there exists α^* such that $C(\alpha^*) = 0$. A more technical summary of these important theorems is presented in the Appendix A.2.

We use stochastic gradient descent methods, employing back propagation and the Adam method, to find the network's parameters by minimizing (2.17). We designed a deep neural network with 6 hidden layers to learn NN; the architecture of this deep neural network is shown in Table 2.1. The width and depth of neural network, which control the generalization power and convergence rate, are tunable hyperparameters. We selected these hyperparameters guided by an ablation study, which is summarized in Appendix A.3.

Hidden layer	1	2	3	4	5	6
Number of neurons	60	180	180	180	180	60

Table 2.1: Architecture of the deep neural network: number of neurons (cells) in each hidden layer.

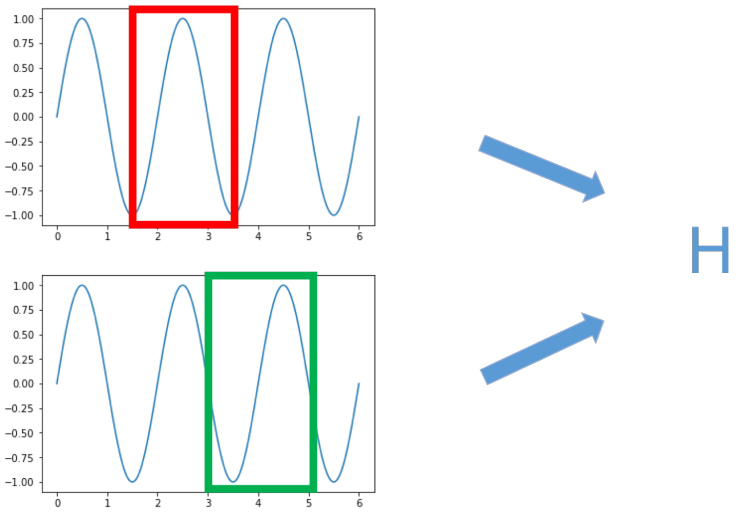


Figure 2.1: Density field in any period window generates the same effective Hamiltonian. This global shift-invariance is required for the learned map.

2.4.2 Neural Network with Global Shift-Invariance

In the SCFT model, with periodic boundary conditions, the effective Hamiltonian is invariant under shifts in ρ. Hence, our approximation $\tilde{H}$ should have the property

$$\tilde{H}(x) = \tilde{H}(T_s x), \tag{2.19}$$

where $T_s x = (\chi N, L, f, T_s \rho)$ and $T_s \rho$ is a spatial shift of the periodic density field ρ, i.e., $T_s \rho(r) = \rho(r + s)$. Equivalently, $\tilde{H}(\chi_N, L, f, [\rho_1, \rho_2, ...]) = \tilde{H}(\chi_N, L, f, [\rho_{1+s}, \rho_{2+s}, ...\rho_1, ..., \rho_s])$. This is illustrated in Figure 2.1: any period-window of ρ generates the same effective Hamiltonian. This is of course also the case for the entropic remainder term R. Machine learning techniques with global shift-invariance are not yet common in the literature. Some popular deep learning architectures, such as the convolutional neural network (CNN), have *local* shift-invariance. However, it is easy to construct examples in which a small change in input can result in a huge change of the output in a CNN [13]. Our goal here is to construct a deep neural network with global

shift-invariance. This can be achieved by combining data augmentation and the addition of a penalty or regularization term in the loss function. Data augmentation consists in adding (cyclically) shifted data points to training set so that the neural network can learn the pattern of periodic shift-invariance directly from the data.

Suppose $M = \{(x_i, H_i), 1 \leq i \leq N_T\}$ is the original training set, the new training set after augmentation is defined as:

$$\tilde{M} = \{(T_s x_i, H_i), 1 \leq i \leq N_T, 1 \leq s \leq N_s\},$$

where N_s is number of possible shifts. We also modify the cost or loss function with an additional penalty term as follows

$$\begin{aligned} C(\alpha) = &\sum_{s=1}^{N_s}\sum_{i=1}^{N_T}(NN(T_s x_i) - \tilde{H}_i)^2 \\ &+ \beta \sum_{s=1}^{N_s}\sum_{i=1}^{N_T} \|\nabla_\rho NN(T_s x_i) - \frac{2\chi N_i}{L_i}\Delta r T_s \rho_i\|^2 \\ &+ \gamma \sum_{s=1}^{N_s}\sum_{i=1}^{N_T}(NN(x_i) - NN(T_s x_i))^2. \end{aligned} \tag{2.20}$$

The last term penalizes the differences of NN at shifted points. It is worth noting that while global shift-invariance is essential for many physical models, the machine learning literature on that topic is scarce. Global shift-invariance is a challenging problem because it results in a much smaller subspace of the general neural network approximation space. Therefore, the existence of a neural network, in the global shift-invariant space, that approximates accurately a functional in the Sobolev norm is an important and non-trivial question. Indeed, if the subspace is not large enough, the existence of a network to approximate accurately any given functional is not guaranteed. Fortunately, we are able to prove the existence of a neural network that approximates a given functional under

the Sobolev norm and has global shift-invariance at the same time. In more precise terms, we have the following result.

Theorem 1 *There exists a neural network $NN(x)$ with a ReLU (or a leaky ReLU) activation function such that the neural network has zero training loss with loss function (2.20).*

Proof: Let $f(x) = R(x)$, i.e. the true remainder of the effective Hamiltonian and $g(\rho) = \frac{2\chi_N}{L}\Delta r\rho$, the true gradient of the remainder. Then, we can take any extension of $g(\rho)$, say $\tilde{g}(x)$, such that $\tilde{g}(x)$ matches $g(\rho)$ on all the coordinates corresponding to ρ. By Theorem 6 in the Appendix, there is a neural network such that $NN(x)$ matches $f(x)$ and $NN_x(x)$ matches $\tilde{g}(x)$ exactly, on the training set. Since we only need the gradient with respect to ρ, by the definition of $\tilde{g}(x)$, NN_ρ is equal to $g(\rho)$ on the training set. This directly makes the first two terms vanish in cost function (2.20). For the third term, noting that

$$NN(T_s x_i) = R(T_s x_i) = R(x_i) = NN(x_i),$$

for all i, s, we get $NN(x_i) - NN(T_s x_i) = 0$. Thus, there exists a neural network $NN(x)$ that leads to a zero training loss with the cost function (2.20). In other words, it is possible to achieve simultaneously the desired approximation in the Sobolev norm and global shift-invariance with one neural network. ■

2.4.3 Searching for Saddle Points

As described above, the map $\tilde{H}$ is trained in Sobolev space. Once this is achieved, we can use local minimizers of $\tilde{H}$, obtained via gradient descent, as predictors of the saddle point density fields of the effective Hamiltonian.[1] To solve $\nabla_\rho \tilde{H} = 0$ we employ

[1]Henceforth, we call the local minimizer of $\tilde{H}$ a saddle point predictor or a predicted saddle point.

the gradient descent method because it is simple and fast to evaluate the learner and its gradient at any x.

Given arbitrary $(\chi N^*, L^*, f^*)$, we eliminate these variables in $\tilde{H}(x)$ so that $\tilde{H}$ can be viewed as a function of ρ alone. Then, we proceed with gradient descent method to find a local minimizer $\widehat{\rho}^{NN}$ as an approximation to a saddle point ρ^* of the effective field-theoretic Hamiltonian:

$$\rho^{n+1} = \rho^n - \epsilon \nabla_\rho \tilde{H}(x^n), \tag{2.21}$$

where $x^n = (\chi N^*, L^*, f^*, \rho^n)$ and ϵ is the step size. Note that, as expressed in (2.16),

$$\nabla_\rho \tilde{H}(x) = -\frac{2\chi_N}{L} \Delta r \, \rho + \nabla_\rho NN(x), \tag{2.22}$$

where $\nabla_\rho NN(x)$ can be efficiently evaluated by the Chain Rule applied to the neural network, e.g. $\nabla_\rho g(x) = \sum_{j=1}^{q} \omega_j \nabla_\rho \, G(\theta_j \cdot \tilde{x})$ for a single-layer neural network. This is easy to compute even for a multiple-layer neural network because a computation graph is automatically generated for the neural network to gather gradient information for backward propagation. Again, here Δr is the spatial mesh resolution. The gradient descent iteration (3.18) is terminated when $\|\nabla_\rho \tilde{H}\|$ decreases below a target specified small value.

One important component of the gradient descent method is the initial iterate ρ^0. Here we propose a selection strategy for ρ^0 that fully uses the information of the training set. For a given $(\chi N^*, L^*, f^*)$, we scan all the density fields in the training set to find the one that generates the smallest $\|\nabla_\rho \tilde{H}\|$ at $(\chi N^*, L^*, f^*)$. Since a neural network is just a composition of a series of linear functions and simple (nonlinear) activation functions, this evaluation is extremely fast in comparison with the evaluation of the full SCFT

model.

2.4.4 Strategies to Tune Hyperparameters

Hyperparameter tuning is an important procedure in deep learning. The weights and biases in the network, denoted collectively as α above, are parameters. There are multiple hyperparameters in a neural network model, such as the strengths of the penalization terms β and γ in the loss function, the hyperparameters defining the architecture of the network, such as the number of layers and the number of nodes per layer, and the learning rate (step size) used in the training process.

Following standard practice, we split the dataset into three parts: a training set, a validation set, and a test set. For various combinations of different values of hyperparameters, the neural network is trained on the training set and the results are compared with the data in the validation set. After selecting the optimal combination of hyperparameters based on the performance on the validation set, we evaluate the best model on the test set to check the error. This comparison requires the specification of a metric. For the given validation set with parameters $(\chi N_i^*, L_i^*, f_i^*)$ where $1 \leq i \leq N_V$, we define the metric or loss function on the validation set as follows

$$\begin{aligned} C_V(\alpha) = &\sum_{s=1}^{N_s}\sum_{i=1}^{N_V}(NN(T_s x_i^*) - R_i)^2 \\ &+ \beta_V \sum_{s=1}^{N_s}\sum_{i=1}^{N_V} \|\nabla_\rho NN(T_s x_i^*) - \frac{2\chi N_i^*}{L_i^*}\Delta r T_s \rho_i^*\|^2 \\ &+ \gamma_V \sum_{s=1}^{N_s}\sum_{i=1}^{N_V}(NN(x_i^*) - NN(T_s x_i^*))^2 \\ &+ \theta_V \sum_{i=1}^{N_V} \|\widehat{\rho}_i^{NN} - \rho_i^*\|^2, \end{aligned} \tag{2.23}$$

where $x_i^* = (\chi N_i^*, L_i^*, f_i^*, \rho_i^*)$ and $\widehat{\rho}_i^{NN}$ is the predicted density field corresponding to

$(\chi N_i^*, L_i^*, f_i^*)$. The first two terms provide a measure of the accuracy of the learner and its gradient. The third term measures the shift-invariance. The fourth term measures the deviation of predicted minimizers $\widehat{\rho}^{NN}$ from the saddle points. Thus, with this metric on the validation set we can find hyperparameters that lead to a neural network that not only approximates the effective Hamiltonian and its gradient but also predicts the saddle point after gradient descent searching. In the implementation, we can adjust the weights (β_V, γ_V and θ_V) to each of the terms in C_V to prioritize the approximation of NN, its gradient, its shift-invariance, or a saddle point, local density field or to balance the scaling.

2.4.5 The Algorithm

In training deep networks it is common to add a regularization term to the loss function that penalizes the size of weights and biases and provides smoothness to the model. Here, we follow that practice and modify our loss function by adding the term $\lambda\|\alpha\|_2^2$, i.e.

$$\begin{aligned} C(\alpha) = &\sum_{s=1}^{N_s}\sum_{i=1}^{N_T}(NN(T_s x_i) - R_i)^2 \\ &+ \beta\sum_{s=1}^{N_s}\sum_{i=1}^{N_T}\|\nabla_\rho NN(T_s x_i) - \frac{2\chi N_i}{L_i}\Delta r T_s \rho_i\|^2 \\ &+ \gamma\sum_{s=1}^{N_s}\sum_{i=1}^{N_T}(NN(x_i) - NN(T_s x_i))^2 + \lambda\|\alpha\|_2^2. \end{aligned} \tag{2.24}$$

Algorithm 1 provides the sequence of steps for deep learning H and to obtain predictions of SCFT saddle points. Training of the neural network is done under the Sobolev norm to obtain accurate predictions of both H and its gradient. With this network, after selecting a good, data-based initial guess, the gradient descent method is used to find

accurate approximations of the saddle point.

Algorithm 1: Neural network method to learn the effective Hamiltonian and to obtain a saddle point density field prediction.

Input: Training set M, validation set M_V, test set M_T

1. Hyperparameter tuning:

for *hyperparameter* λ_i, β_j, γ_k **do**

  use Adam method to search for $\alpha_{ijk} = \arg\min_\alpha C(\alpha)$

  evaluate $C_V(\alpha_{ijk})$ on M_V

end

$\lambda, \beta, \gamma = \arg\min_{\lambda_i,\beta_j,\gamma_k} C_V(\alpha_{ijk})$

2. Prediction of effective Hamiltonian on M_T:

for $x = (\chi N, L, f, \rho)$ **do**

  $\tilde{H}(x) = \chi N f - \frac{\chi N}{L} \int \rho^2 dr + NN(x)$

end

3. Prediction of saddle point corresponding to $\chi N^*, L^*, f^*$:

for ρ_i *from* M **do**

  Evaluate $\tilde{H}(\chi N^*, L^*, f^*, \rho_i)$, $\nabla_\rho \tilde{H}(\chi N^*, L^*, f^*, \rho_i)$

end

$\rho^0 = \arg\min_{\rho_i} \|\nabla_\rho \tilde{H}(\chi N^*, L^*, f^*, \rho_i)\|$

while $\|\nabla_\rho \tilde{H}(\chi N^*, L^*, f^*, \rho_i)\| > \delta$ **do**

  $\rho^{n+1} = \rho^n - \epsilon \nabla_\rho \tilde{H}(x^n)$

end

$\rho^n \to \tilde{\rho}^{NN}$ which is the estimated saddle point corresponding to $\chi N^*, L^*, f^*$

2.5 Exploring a Simpler Data-Based Learner

As we show in Section 2.6, our deep NN-based approach is remarkably accurate and efficient in building a shift-invariant surrogate for the field-theoretic effective Hamiltonian in the vicinity of saddle points. There is however a vast number of other data-based learners that one could potentially use within our proposed framework to accelerate the exploration of parameter space in polymer SCFT. In fact, the first learner we tried, inspired by

the pioneering work of Snyder, Rupp, Hansen, Blooston, Müller, and Burke [14, 15, 16], was Kernel Ridge Regression (KRR).

In KRR, one constructs the approximate map $\tilde{H}$ from an expansion of the form

$$\tilde{H}(x) = \sum_{j=1}^{N_T} \alpha_j K(x, x_j), \tag{2.25}$$

where, as before, $x = (\chi N, L, f, \rho)$, x_i represents i_{th} training point, K is a fixed function known as the kernel (e.g. the Gaussian function), and N_T is the training set size. Because the number of parameters α_j is not fixed and depends on the size N_T of the data , KRR is a simple data-adaptive regression approach.

To use KRR as the data-based learner in our proposed, accelerated SCFT framework we need 1) to train KRR (i.e. determine $\alpha_1, \alpha_2, ..., \alpha_{N_T}$), in Sobolev space to approximate simultaneously H and its gradient, and 2) to constrain the KRR learner to be shift-invariant. While we could solve 1) explicitly, we were unable to find a satisfactory solution for 2). Although this is a serious limitation of KRR in the context of polymer SCFT, the extension of the KRR to Sobolev space training could be useful for other applications that do not require the global invariance. With this in mind, we present below this extension and compare this approximation with that produced by the deep NN learner.

Specifically, to train KRR in Sobolev space and determine $\alpha_1, \alpha_2, ..., \alpha_{N_T}$, we minimize the cost function

$$C(\alpha) = \sum_{i=1}^{N_T} (\tilde{H}(x_i) - H_i)^2 + \beta \sum_{i=1}^{n} \|\nabla_\rho \tilde{H}(x_i)\|^2 + \lambda\, \alpha^T K \alpha, \tag{2.26}$$

where $\nabla_\rho \tilde{H}(x_i)$ stands for the gradient of $\tilde{H}$ with respect to ρ evaluated at x_i, $\alpha = (\alpha_1, \alpha_2, ..., \alpha_{N_T})^T$, and K is a matrix with entries $K_{ij} = K(x_i, x_j)$. The regularization

term $\lambda\,\alpha^T K\alpha$ penalizes the size of the coefficients α in the metric induced by the kernel and limits overfitting. The gradient term in $C(\alpha)$ favors approximations with a vanishing gradient, consistent with SCFT saddle points.

We now derive an explicit expression for α in our Sobolev space KRR model. We employ the Gaussian kernel

$$K(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right), \tag{2.27}$$

where

$$\|x_i - x_j\| = \sqrt{c_1(\chi N_i - \chi N_j)^2 + c_2(L_i - L_j)^2 + c_3(f_i - f_j)^2 + \|\rho_i - \rho_j\|_2^2}, \tag{2.28}$$

$\|\cdot\|_2$ is the l_2 norm, and c_1, c_2, c_3 are positive hyperparameters.

Theorem 2 *The Sobolev space-trained Kernel Ridge Regression (2.25) with Gaussian kernel (2.27) , which minimizes (2.26), has coefficients α that satisfy linear system*

$$(K^T K + \beta\tilde{K} + \lambda K)\alpha = K^T H, \tag{2.29}$$

where $\tilde{K} = \sum_{i=1}^{N_T} \tilde{K}_i$ and $\tilde{K}_i$ is a matrix with $(\tilde{K}_i)_{jm} = \frac{1}{\sigma^4} K(x_i, x_j)K(x_i, x_m)\langle\rho_j - \rho_i, \rho_m - \rho_i\rangle$.

Proof: Since $\tilde{H}(x) = \sum_{j=1}^{N_T} \alpha_j K(x, x_j)$, we have

$$\begin{aligned}
\nabla_\rho \tilde{H}(x) &= \sum_{j=1}^{N_T} \frac{\rho_j - \rho}{\sigma^2} \alpha_j K(x, x_j), \\
\|\nabla_\rho \tilde{H}(x_i)\|^2 &= \langle \nabla_\rho \tilde{H}(x_i), \nabla_\rho \tilde{H}(x_i) \rangle \\
&= \langle \sum_{j=1}^{N_T} \frac{\rho_j - \rho_i}{\sigma^2} \alpha_j K(x_i, x_j), \sum_{m=1}^{N_T} \frac{\rho_m - \rho_i}{\sigma^2} \alpha_m K(x_i, x_m) \rangle \\
&= \sum_{j=1}^{N_T} \sum_{m=1}^{N_T} \alpha_j \alpha_m \frac{1}{\sigma^4} K(x_i, x_j) K(x_i, x_m) \langle \rho_j - \rho_i, \rho_m - \rho_i \rangle \\
&= \alpha^T \tilde{K}_i \alpha,
\end{aligned}$$

where $\tilde{K}_i$ is a matrix with $(\tilde{K}_i)_{jm} = \frac{1}{\sigma^4} K(x_i, x_j) K(x_i, x_m) \langle \rho_j - \rho_i, \rho_m - \rho_i \rangle$. Thus,

$$\sum_{i=1}^{N_T} \|\nabla_\rho \tilde{H}(x_i)\|^2 = \sum_{i=1}^{N_T} \alpha^T \tilde{K}_i \alpha = \alpha^T \tilde{K} \alpha, \tag{2.30}$$

where $\tilde{K} = \sum_{i=1}^{N_T} \tilde{K}_i$. Note that both $\tilde{K}_i$ and $\tilde{K}$ are symmetric matrices. The cost function (2.26) can now be rewritten as

$$\begin{aligned}
C(\alpha) &= \sum_{i=1}^{N_T} (\tilde{H}(x_i) - H_i)^2 + \beta\, \alpha^T \tilde{K} \alpha + \lambda\, \alpha^T \tilde{K} \alpha \\
&= (H - K\alpha)^T (H - K\alpha) + \beta\, \alpha^T \tilde{K} \alpha + \lambda\, \alpha^T K \alpha,
\end{aligned} \tag{2.31}$$

Therefore

$$\frac{\partial C}{\partial \alpha} = 2(-K^T)(H - K\alpha) + \beta(\tilde{K} + \tilde{K}^T)\alpha + \lambda(K + K^T)\alpha = 0 \tag{2.32}$$

and consequently, α is the solution of the linear system

$$(K^T K + \beta \tilde{K} + \lambda K)\alpha = K^T H. \tag{2.33}$$

■

This result is an attractive feature of KRR; there is an explicit expression for α, i.e. the learner is easy to train.

There are 6 hyperparameters in our KRR method: c_1, c_2, c_3, λ, σ, and β. They could either be determined by the methods in Section 2.4.4 or by k-fold cross validation. Once the KRR learner is obtained, we proceed as in the deep neural network case, to minimize $\tilde{H}$, via gradient descent, to find saddle point predictors

$$\rho^{n+1} = \rho^n - \epsilon \nabla_\rho \tilde{H}(x^n), \tag{2.34}$$

where the gradient is evaluated explicitly:

$$\nabla_\rho \tilde{H}(x) = \sum_{j=1}^{N_T} \frac{\rho_j - \rho}{\sigma^2} \alpha_j \, K(x, x_j). \tag{2.35}$$

2.6 Results

We conducted several numerical experiments to validate the proposed SCFT-deep learning framework. We summarize our results in this section.

We considered two systems: an AB diblock copolymer melt and an AB_3 star copolymer melt, both in one spatial dimension. There were two stages in the numerical experiments. First, we trained a machine learning architecture (the deep neural network

here, but other machine learning methods could be used in this step) to predict the effective Hamiltonian H and its gradient from the parameters, χN, L, f and average monomer (A) density field ρ. The deep neural network learner is equipped with global shift-invariance obtained through a data augmentation technique and the addition of a penalty term in the loss function. Second, we selected an initial guess for the density field from the training set and searched by gradient descent for a density field that minimizes the learned map $\tilde{H}$.

2.6.1 AB Diblock Copolymer with Low-to-Moderate χN

In this experiment, we first test the algorithms on an AB diblock system for χN in the range $[20, 35]$ for A-block volume fractions $f \in \{0.3, 0.4\}$. The training, validation, and test data for $f = 0.3$ are as follows:

- The **training set** consists of the combination of $\chi N \in \{n \in \mathbb{N} : 20 \leq n \leq 35\}$, $L \in \{n + 0.2, n + 0.5, n + 0.8 : 3 \leq n \leq 6\}$ and $f \in \{0.3\}$. The size of the training set is $16 * 3 * 4 = 192$.

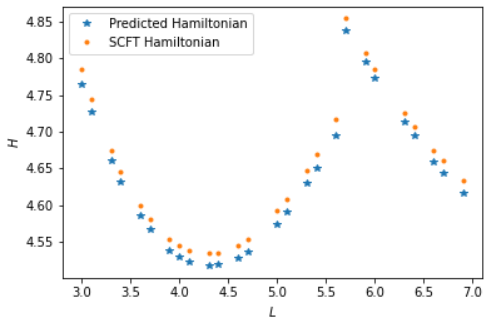
- The **validation** and **test sets** consist of the combination of $\chi N \in \{n, n+0.5 : 20 \leq n \leq 35, n \in \mathbb{N}\}$, $L \in \{n, n+0.1, n+0.3, n+0.4, n+0.6, n+0.7, n+0.9 : 3 \leq n \leq 6\}$ and $f \in \{0.3\}$. Note that this larger set has no overlap with the training set. We randomly take $192/3 = 64$ points by uniform distribution from this set as our validation set and apply the rest as the test set.

All the data points are obtained by numerically solving the SCFT model for each corresponding set of parameters. The modified diffusion equations were solved using periodic boundary conditions and pseudo-spectral collocation in space with 64 mesh points in 1D. 100 contour steps were made using second-order operator splitting [17, 18]. Auxiliary fields, initialized with smooth fields with a fixed number of periods, were relaxed

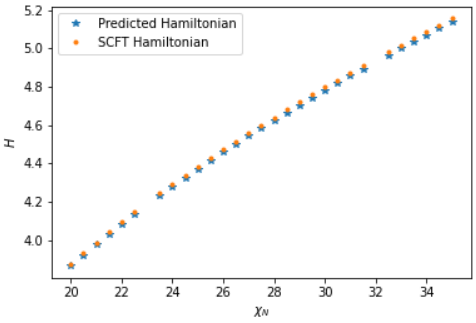
to saddle-point configurations using the semi-implicit Seidel iteration [5]. L was varied rather than set to the value that minimizes the effective Hamiltonian in building the datasets. This gave more richness to the training set by including stressed configurations. However, as L increases, solutions with an increased number of periods become feasible; the algorithm selects the density field that produces the smallest effective Hamiltonian among these solutions with different periods. We use a similar strategy to generate the data for the case $f = 0.4$. The deep neural network is trained separately for $f = 0.3$ and $f = 0.4$.

Figure 2.2 shows the excellent accuracy of the predicted $\tilde{H}$ as a function of χN and L. $\tilde{H}$ coincides with the SCFT mean-field free energy H to two digits of accuracy. As mentioned above, global shift-invariance is an important property the SCFT model solutions. Figure 2.3a displays the output of the deep network for all the possible periodic shifts (cyclic permutations) of the density array input (from 0 to 63) for some representative cases. As this figure demonstrates, the proposed deep network preserves with good accuracy (2–3 digits) the desired, global shift-invariance.

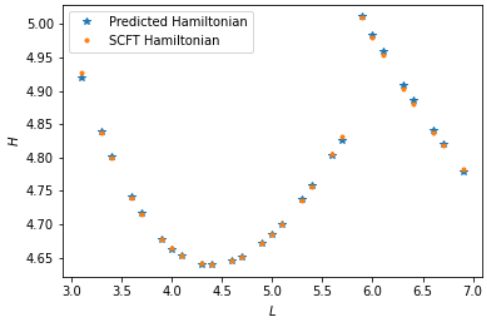
We now test the ability of our deep learning model to predict saddle point SCFT densities. As mentioned earlier this is done using gradient descent of the learned map. The search process starts from the density field in the training set that generates a gradient of the learner $\tilde{H}$ closest to 0. Then, in each iteration, we evaluate the gradient of the learner by summing up the gradient of the leading order term and the gradient of the deep-learned remainder R, computed by the Chain Rule. Because the training is done in Sobolev space, a minimizer of the learner $\tilde{H}$ yields an accurate approximation of the true SCFT saddle point density field. Figure 2.4 presents four examples of the predicted density field and compares them with the corresponding SCFT solution. The accuracy of the deep learning model predictions is outstanding. Table 2.2 lists the specific errors of the predictions on the test set for both the AB diblock and the AB_3 star copolymer.



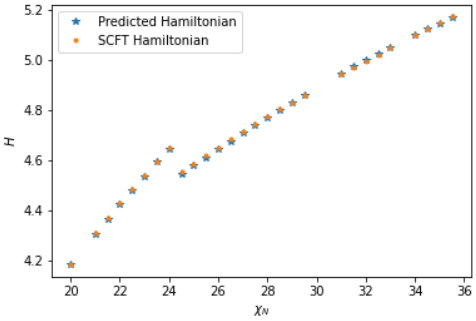
(a) $\chi N = 27.5$, $3 \leq L \leq 6.9$, and $f = 0.3$.



(b) $L = 3.6$, $20 \leq \chi N \leq 35.5$, and $f = 0.3$.



(c) $\chi N = 28.5$, $3 \leq L \leq 6.9$, and $f = 0.4$.

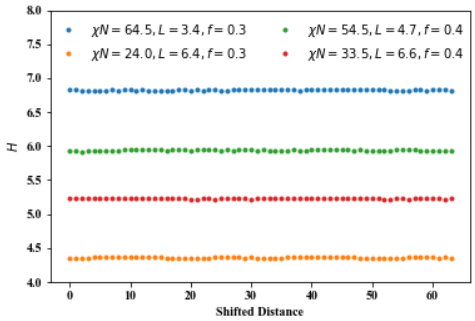


(d) $L = 5.6$, $20 \leq \chi N \leq 35.5$, and $f = 0.4$.

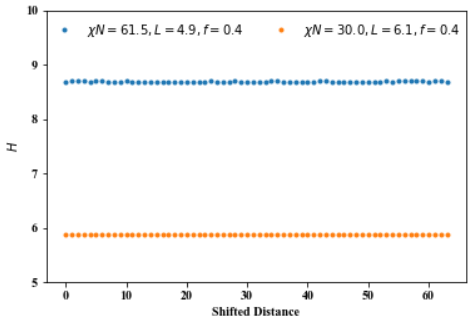
Figure 2.2: AB diblock copolymer with low-to-moderate χN: comparison of the deep learned effective Hamiltonian with the SCFT effective Hamiltonian. In (d), there is a jump because the optimal density field switches from two periods to one period at that point and this leads to a smaller effective Hamiltonian.

χN	Structure	f	E_H	E_G	E_I
low χN	AB diblock	0.3	0.017	0.018	0.006
		0.4	0.006	0.023	0.005
	AB_3 star	0.4	0.013	0.035	0.005
high χN	AB diblock	0.3	0.011	0.050	0.006
		0.4	0.012	0.046	0.017
	AB_3 star	0.4	0.017	0.065	0.012

Table 2.2: Performance of the Sobolev space-trained deep neural network on the test set (root mean square error): E_H is the error of predicted effective Hamiltonian, E_G is the error of predicted effective Hamiltonian gradient, E_I is the difference of predicted effective Hamiltonian of shifted density fields. Low χN refers to $\chi N \in [20, 35.5]$ for all cases. High χN refers to $\chi N \in [50, 65.5]$ for the AB diblock with $f = 0.3$ and for the AB_3 star system, and to $\chi N \in [45, 60.5]$ for the AB diblock with $f = 0.4$.

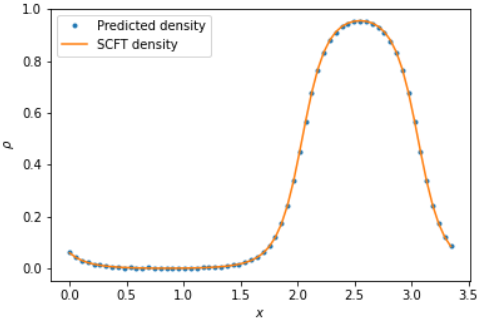


(a) AB diblock copolymer

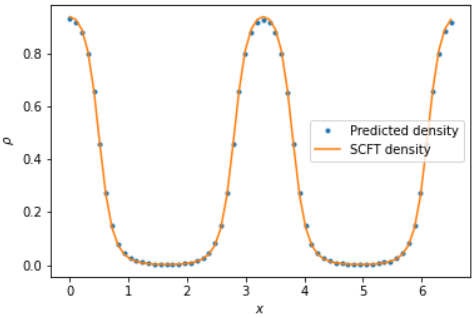


(b) AB_3 star copolymer

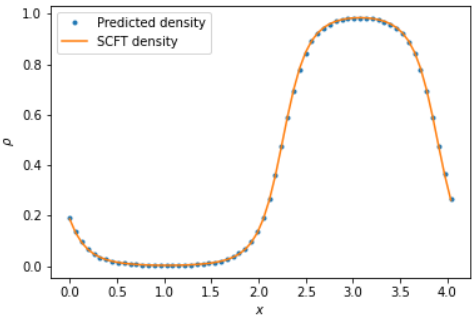
Figure 2.3: Global shift-invariance validation for some representative cases: (a)low-to-moderate χN and high χN cases for AB diblock system, (b)low-to moderate and high χN cases for AB_3 star system.



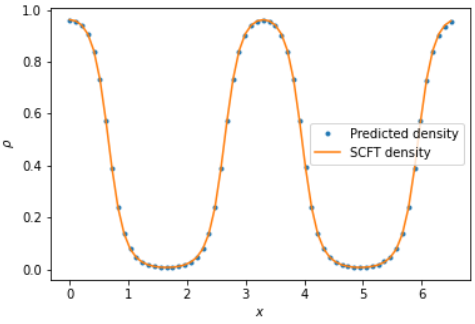
(a) $\chi N = 33.5$, $L = 3.4$, and $f = 0.3$.



(b) $\chi N = 30.5$, $L = 6.6$, and $f = 0.3$.



(c) $\chi N = 26.5$, $L = 4.1$, and $f = 0.4$.



(d) $\chi N = 24.0$, $L = 6.6$, and $f = 0.4$.

Figure 2.4: AB diblock copolymer with low-to-moderate χN: predicted and SCFT saddle density fields.

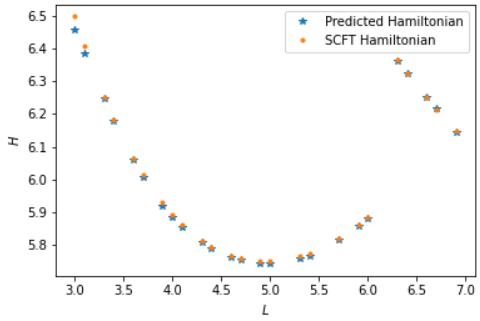
We are using a relatively small training set but get fairly accurate approximations. Increasing the number of data points would generally improve the performance. As proved by Chen, Jiang, Liao, and Zhao [19], there exists a deep ReLU architecture such that the mean squared error of the approximation converges at the rate of $O(n^{-\frac{2(s+\alpha)}{2(s+\alpha)+d}} \log^3 n)$ when n points are sampled to approximate a Hölder function in $\mathcal{H}^{s,\alpha}$ supported on a d-dimensional Riemannian manifold isometrically embedded in $\mathbb{R}^D$ with sub-Gaussian noise and a data intrinsic dimension of d.

2.6.2 AB Diblock Copolymer with High χN

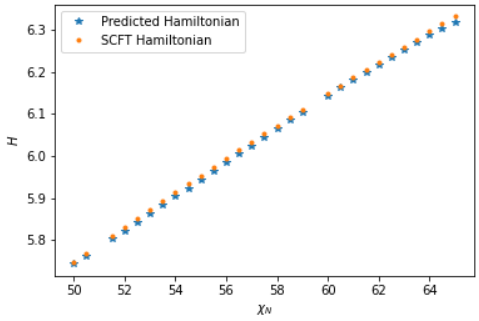
We now consider the case of high χN, i.e. strong segregation. This is a notoriously difficult case from the SCFT computational point of view. High χN SCFT simulations usually require high spatial resolution and are numerically stiff, taking hundreds or thousands of iterations to converge. Here, we select χN in the range $[50, 65]$ for $f = 0.3$ and χN in the range $[45, 60]$ for $f = 0.4$, using a similar procedure with same number of spatial and contour grid points as Section 2.6.1 to generate the dataset.

Figure 2.5 shows representative predictions of the effective Hamiltonian map H from $(\chi N, L, f, \rho)$, and Figure 2.3a displays representative results for global shift-invariance. As in the low-to-intermediate χN cases, the deep learner produces excellent predictions. Table 2.2 demonstrates that we still have several digits of accuracy in H, its gradient, and in preserving global shift-invariance. However, as expected, the error in the gradient is larger for the high χN cases because of the sharper interfaces.

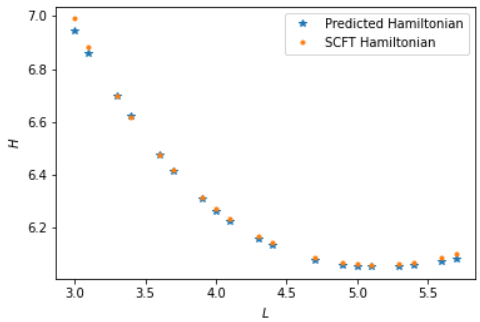
Predictions of the SCFT saddle point density field for a given $(\chi N^*, L^*, f^*)$ are shown in Figure 2.6. The deep learner in combination with gradient descent is able to predict the density profiles for these higher values of χN almost as accurately as for the lower χN values. We emphasize that high χN SCFT computations are computationally much



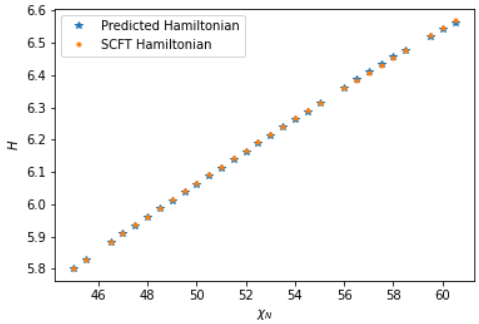
(a) $\chi N = 50.5$, $3 \leq L \leq 6.9$, and $f = 0.3$.



(b) $L = 4.6$, $50 \leq \chi N \leq 65.5$, and $f = 0.3$.



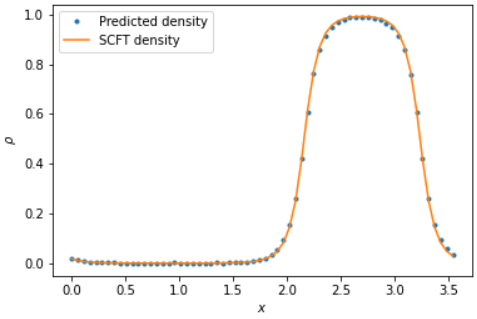
(c) $\chi N = 58.5$, $3 \leq L \leq 5.8$, and $f = 0.4$.



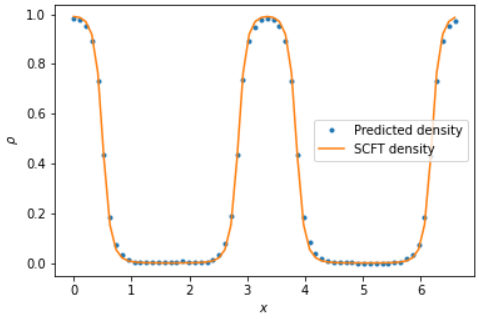
(d) $L = 3.6$, $45 \leq \chi N \leq 60.5$, and $f = 0.4$.

Figure 2.5: AB diblock copolymer with high χN: comparison of the learned and SCFT Hamiltonian.

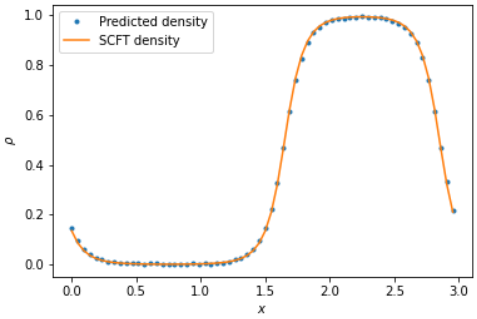
more expensive than those for smaller χN due to the requirement of higher numerical resolution and slower convergence of the saddle point iterations. In contrast, with the deep learning method we can obtain an accurate prediction of the saddle point density field for high χN fast, at the same cost as that for obtaining a density prediction for low-to-moderate χN case; in both cases, it is just several evaluations of a neural network, which is a combination of linear functions and activation functions as shown in Eq. (2.18). After generating hundreds of training data points and a one-time training, the Sobolev space-trained neural network becomes a valuable, fast computational tool to predict accurately the effective Hamiltonian from density fields on any dataset, for example a large scale dataset with tens of thousands of data points to evaluate. No additional



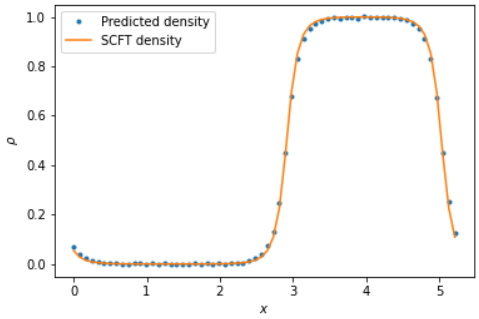
(a) $\chi N = 59.5$, $L = 3.6$, and $f = 0.3$.



(b) $\chi N = 60.0$, $L = 6.7$, and $f = 0.3$.



(c) $\chi N = 52.0$, $L = 3.0$, and $f = 0.4$.



(d) $\chi N = 57.5$, $L = 5.3$, and $f = 0.4$.

Figure 2.6: AB diblock copolymer with high χN: predicted and SCFT saddle density field.

training or fine-tuning is needed during the subsequent polymer phase discovery process. To illustrate the running time difference between the predictions by the numerical solution of SCFT and the new deep neural network model, we take 200 samples of $(\chi N, L, f)$ with $61 \leq \chi N \leq 65$, to compare both approaches. The running times are shown in Table 2.3. The SCFT CPU and the neural network CPU times are from the same machine (MacBook Pro, 2.2 GHz Intel Core i7 Processor, 16 GB 1600 MHz DDR3 Memory). The neural network GPU time is the running time on a Tesla P100 GPU. The same stopping criterion was used CPU and GPU neural network experiments. The superiority of the deep neural network model is clear and the computational savings would even more dramatic in 2D and 3D. Finding saddle point, local density fields with large 3D SCFT computations

can take several hours whereas we should expect the proposed deep learning approach to accomplish the same task in seconds. In addition, as the size of sample set to be predicted increases, the neural network approach becomes more efficient because it is implemented based on tensor operations and the running time increases slowly as the size of input set increases.

Model	SCFT CPU	neural network CPU	neural network GPU
Prediction time	$1123s$	$5.97s$	$3.27s$

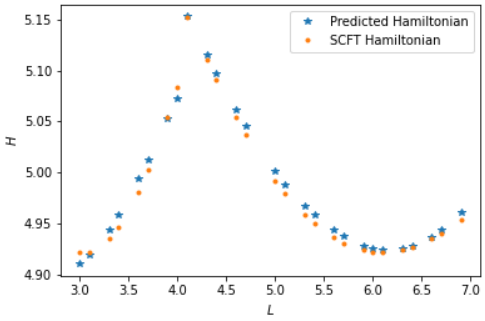
Table 2.3: Comparison of the direct SCFT and the deep neural network approach in terms for 200 samples.

2.6.3 AB_3 Star Copolymer

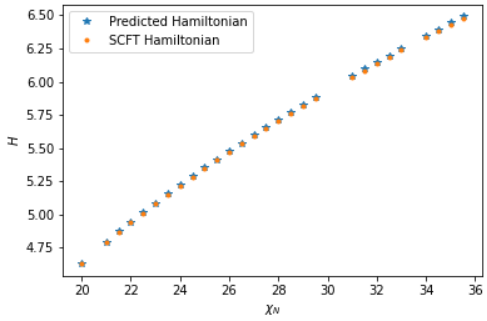
To show the generalizability of our model, we now test our deep learning model for an AB_3 star copolymer melt, which has a different molecular architecture than the AB diblock melt. In an AB_3 star system, three B blocks are attached at a point to the A block terminus. In both systems, strand lengths (including degeneracy factor) sum to 1. In the AB diblock case, $f_A + f_B = 1$ while $f_A + 3f_B = 1$ for the AB_3 star system.

We employ the same technique for the AB_3 star system as used for the AB diblock melt, where we wrote the effective Hamiltonian as the sum of the enthalpic term (explicitly extracting the quadratic interaction) and a remainder that contains the polymer entropy and is deep learned in Sobolev space. Experiments are run on both a low-to-moderate χN case and a high χN case, and in both cases accurate results are obtained.

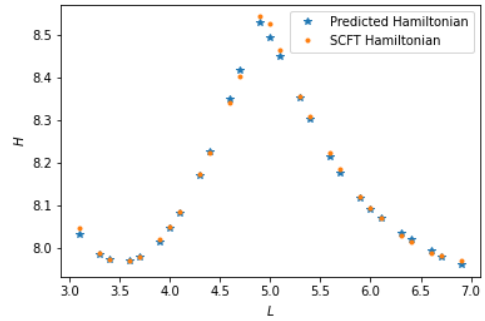
The predicted map from $(\chi N, L, f, \rho)$ to effective Hamiltonian H is shown in Fig. 2.7 and a validation of the global shift-invariance is presented in Fig. 2.3b. Just as for the AB diblock, the predictions for the AB_3 star copolymer are, as Table 2.2 quantifies, very



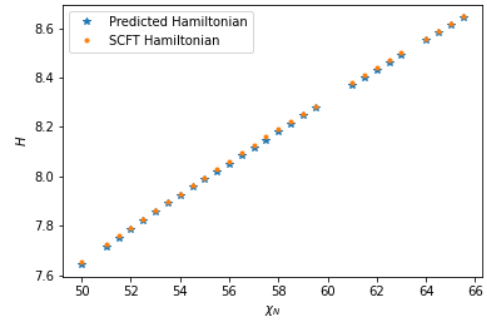
(a) $\chi N = 22$, $3 \leq L \leq 6.9$, and $f = 0.4$.



(b) $L = 5.6$, $20 \leq \chi N \leq 35.5$, and $f = 0.4$.



(c) $\chi N = 58.5$, $3 \leq L \leq 6.9$, and $f = 0.4$.



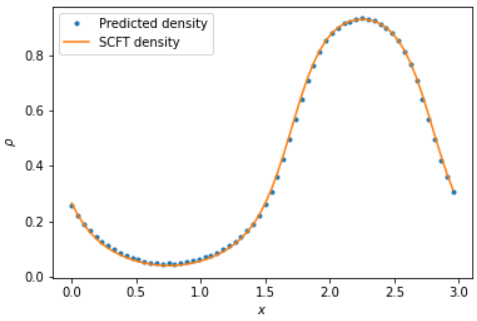
(d) $L = 5.6$, $50 \leq \chi N \leq 65.5$, and $f = 0.4$.

Figure 2.7: AB_3 star copolymer: comparison of predicted and SCFT effective Hamiltonian.

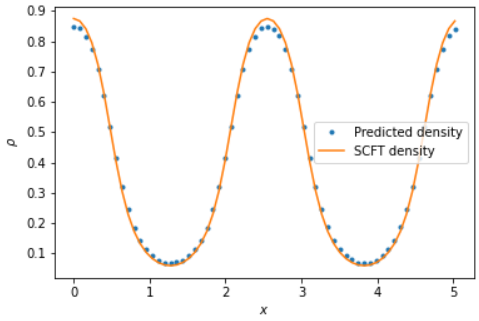
accurate. The density profile predictions are also excellent for both low and high χN as Fig. 2.8 demonstrates.

2.6.4 Comparison with the Kernel Ridge Regression Learner

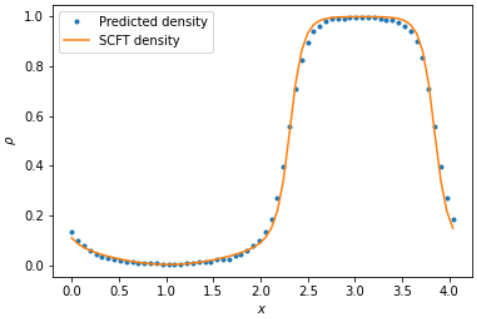
We implemented the Sobolev space-trained Kernel Ridge Regression (KRR) learner introduced in Section 2.5 for the AB diblock copolymer and the AB_3 star copolymer. Even though the KRR does not comply with the shift invariance constraint, a comparison with the deep NN-based method offers information on the capability of the Sobolev space-trained KRR to approximate simultaneously H and its gradient, which might be useful for other applications.



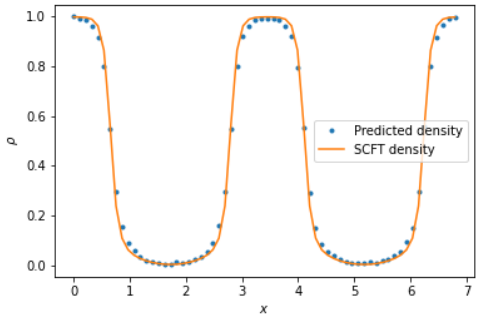
(a) $\chi N = 24.5$, $L = 3.0$, and $f = 0.4$.



(b) $\chi N = 23.0$, $L = 5.1$, and $f = 0.4$.



(c) $\chi N = 59.5$, $L = 4.1$, and $f = 0.4$.



(d) $\chi N = 64.5$, $L = 6.9$, and $f = 0.4$.

Figure 2.8: AB_3 star copolymer: predicted and SCFT saddle density fields.

Table 2.4 shows a comparison of the accuracy of the deep NN and the KRR for approximating H and its gradient, for both the AB block copolymer and the AB_3 star copolymer. The accuracy in H is slightly better for the deep NN but the KRR yields a more accurate approximation of the gradient. The latter is not surprising given the additional shift-invariance requirement in the deep NN which effectively reduces the approximating space. It is in fact remarkable that the deep NN yields comparable accuracy with that of the constraint-free KRR. This underlines the generalization power of the deep NN while handling additional learning constraints. On the other hand, for systems that do not require a global shift (and/or rotational) invariance, the Sobolev space-trained KRR could provide a simple, accurate, and explicitly trained learner.

Structure	learner	f	E_H	E_G	E_I
AB diblock	NN	0.3	**0.017**	0.018	**0.006**
	KRR	0.3	0.021	**0.006**	–
AB_3 star	NN	0.4	**0.017**	0.065	**0.012**
	KRR	0.4	0.020	**0.007**	–

Table 2.4: Comparison of the NN and the KRR learners trained in Sobolev space for the AB diblock and the AB_3 star system with low χN. E_H is the error of predicted effective Hamiltonian, E_G is the error of predicted effective Hamiltonian gradient, and E_I is the difference of predicted effective Hamiltonian of shifted density fields. The KRR learner does not have this approximate shift invariance.

2.7 Conclusions

In this chapter, we presented a deep learning framework to accelerate the exploration of parameter space for block copolymer systems based on field theoretic models. The central idea is to use data sets obtained from SCFT simulations to train *in Sobolev space* and learn the effective Hamiltonian of the system as a function of the relevant average monomer density field and the model parameters. The proposed neural network learner is built from rigorous universal approximation results in Sobolev space and accurately preserves global shift-invariance. Once this learning process is done, one can expeditiously find, via gradient descent method, an accurate prediction for a saddle point density field. Moreover, we can potentially combine any global optimizer with this neural network approach to search for a global minimum.

The proposed SCFT-deep learning approach could also be used to accelerate the solution of the *inverse design problem*: given target properties of the system, find the parameters and composition that generate those properties. This could be done by optimizing a suitable fitness function, which measures deviations from the target system, and whose evaluation can be expeditiously done through a deep-learned functional.

There are, of course, several other machine learning models beside deep neural networks. As we show in Section 2.5, we could replace the learner by other learner in this

framework and they too have great performance.

The focus of this chapter has been on systems in one spatial dimension to allow us properly develop and test the proposed framework. Work on 2D is shown in chapter 3. In the higher-dimensional case we need global shift invariance along all axes, as well as rotational invariance. Both can be incorporated with the data enhancement and regularization approach proposed here. We will do some modifications on the architecture and methodology to improve the performance in high dimensional cases.

Chapter 3

Deep Learning and Self-Consistent Field Theory: 2D Polymer Phase Discovery

3.1 Introduction

In Chapter 2, we introduced deep learning methodologies for polymer self-consistent field theory (SCFT) in one spatial dimension. In this chapter, we discuss this new marriage of polymer SCFT and deep learning in two spatial dimensions for polymer phase discovery. There are two main challenges: (1) the size of the input, density field space increases exponentially as the spatial dimensions increase and (2) the Hamiltonian must be invariant under both translational and rotational spatial transformations of the density field.

Following the framework developed in the one-dimensional case, we train a machine learner in Sobolev space and employ it to approximate the effective Hamiltonian and to ultimately predict saddle point density fields. In order to tackle the aforementioned

challenges and to ensure the algorithms work efficiently, we upgrade our methodologies as follows:

1. We view the density fields as 2D images and train a Convolutional Neural Network (CNN) [20] in Sobolev space. This ensures that the neural network approximates accurately and efficiently the effective Hamiltonian and its gradient in the vicinity of saddle points. CNN's have proved to be a powerful tool for multiple image-related problems [21].

2. We use a rotation-invariant representation of the polymer cell parameters and design a CNN architecture based on circular padding to achieve strong rotation and translation invariance.

3. We propose to use a Generative Adversarial Network (GAN) [22] to generate initial saddle point density fields without memorizing the training set. This saves enormous computer memory/computation time, and makes it more feasible to generate uncommon saddle point density fields.

The rest of the chapter is organized as follows. Section 3.2 details the two-dimensional problem. Section 3.4 introduces the Convolutional Neural Network trained in Sobolev space, which approximates the effective Hamiltonian and its gradient, and preserves strong rotation/translation invariance. Section 3.4 introduces the developments in the methodology to predict saddle point density fields by generating initial saddle point density fields via a GAN and then fine-tuning the saddle point density fields with the Sobolev space-trained CNN. Section 3.5 presents experimental results of effective Hamiltonian prediction and saddle density fields prediction on the two-dimensional AB diblock polymer system. Finally, concluding remarks about the results and the proposed methodologies are given in Section 3.6.

3.2 Problem Definition and General Strategy

To formulate the problem and illustrate the methodology, we use the concrete example of a 2D incompressible AB diblock copolymer melt. The relevant parameters are χN (χ is the Flory parameter and N is the copolymer degree of polymerization), which measures the strength of segregation of the two components, cell vectors $\vec{a}_1$ and $\vec{a}_2$, which correspond to the adjacent sides of the cell parallelogram, in units of the unperturbed radius of gyration R_g, and f, the volume fraction of component A. Let $\rho = \rho_A$ be the local average monomer density of blocks A. As in the one-dimensional case, we formulate two problems to solve:

- **Problem 1**: Learn a map $(\chi N, \vec{a}_1, \vec{a}_2, f, \rho) \longmapsto \tilde{H}(\chi N, \vec{a}_1, \vec{a}_2, f, \rho)$, where $\tilde{H}$ is an accurate approximation of the field-theoretic intensive Hamiltonian H, the Helmholtz free-energy per chain at saddle points.

- **Problem 2**: For specific values χN^*, $\vec{a}_1^*$, $\vec{a}_2^*$, f^*, find accurately and efficiently a density field ρ^* that minimizes $\tilde{H}$.

The map $\tilde{H}$ is trained to approximate the Hamiltonian near the saddle points in Problem 1. This surrogate functional can thus be evaluated much more efficiently than through direct SCFT computation and provides an expedited way to screen for the density field saddle point candidates in Problem 2.

As the one-dimensional case, we recast effective Hamiltonian H as

$$H(\chi N, \vec{a}_1, \vec{a}_2, f, \rho) = \chi N f - \frac{\chi N}{|\vec{a}_1 \times \vec{a}_2|} \int \rho^2 dr + R(\chi N, \vec{a}_1, \vec{a}_2, f, \rho), \tag{3.1}$$

by extracting the leading quadratic interaction term and focus on learning the remainder of Hamiltonian which contributes to polymer entropy but not enthalphy, where $\times$ is

the cross product thus the denominator of the second term is the cell volume. In this way, we directly evaluate the main part which is cheap to compute and approximate the expensive part by neural networks. The advantages of learning the remainder R instead of the entire Hamiltonian H is discussed in Appendix A.1.

The two-dimensional setting shares the same challenges as the one-dimensional one. For instance, we only have data of effective Hamiltonian at saddle points, a reduced set of a much larger complex manifold. Thus we need make assumptions on the data manifold to guarantee that local minimizers of the learned Hamiltonian, search among saddle and non-saddle points, are accurate approximations of the actual SCFT saddle points. In addition, the exponential increase of in the size of input (discretized) density fields compounds the challenge to accurately approximate H and to predict saddle density fields. The translation invariance also involves more possible shifts in 2D and rotation invariance adds another non-trivial challenge.

To handle these issues, we upgrade the methodology introduced in Chapter 2. To solve Problem 1, instead of training a fully connected feedforward neural network in Sobolev space, we train a CNN in Sobolev space to learn the effective Hamiltonian. In order to implement the CNN on the density fields, we view these as images as use them as input for the CNN. We use the rotation-invariant representation of parameters $\vec{a}_1$ and $\vec{a}_2$ to make the method invariant to rotations. Taking advantage of the local shift invariance of CNN, we design the architecture of the CNN to make it have strongly global shift invariance instead of the weakly global shift invariance in one-dimensional case. To solve problem 2, we add an additional step: use generative adversarial network to generate a initial saddle point density field pool. In 1D case, our initial density fields come from training set. With the help of GAN, it is unnecessary to memorize the training set and dramatically saves memory since the training set is large in high dimensional case. It is unnecessary to evaluate the training set to get the initial saddle

density fields, which reduces time complexity. In addition, the randomness of the GAN increases the possibility to generate unseen initial density fields and hence it enhances the capability of the proposed methodology for the discovery of new polymer phases. Then, we select the initial density fields with the smallest magnitude of gradients by the Sobolev space-trained CNN in Problem 1 and fine-tune the density fields by gradient descent methods.

3.3 Methodology: Approximation of the Effective Hamiltonian by a Convolutional Neural Network

In this section, we describe the methodology to approximate the effective Hamiltonian from the relevant physocal parameters and density fields by a Sobolev space-trained CNN. The 2D density field ρ can be represented as a matrix and thus viewed as image. Then, the density field is processed by several convolutional layers, concatenated with the parameters. Finally, an approximation to the effective Hamiltonian emerges after going through several fully connected layers.

3.3.1 Convolutional Neural Networks to Learn the Entropic Remainder

CNN's are an important class of neural networks based on local convolutions. They have become one of the dominant machine learning techniques for a variety of computer vision tasks [23]. A typical CNN consists of convolutional layers, pooling layers, activation layers and fully connected layers[1]. Generally, the convolutional layer utilizes local convolution to perceptive local signals and pooling layers to extract local features and

[1]Deatils on CNN and animation of these layers could be found at `https://cs231n.github.io/convolutional-networks/`.

achieve dimensionality reduction. A CNN learns spatial features by the combination of these layers. The parameters of a CNN, such as kernel (convolution) weights, are determined by backpropagation.

In each convolutional layer, a local kernel or filter is applied to the input tensor, where the element-wise products of the kernel and a same-size neighbourhood of the tensor are computed and added together. Multiple kernels might be applied to the input tensor to generate multiple channels and the kernel weights are neural network parameters determined by stochastic gradient descent methods. Figure 3.1 shows a schematics of how a convolutional layer works.

A pooling layer is usually a downsampling operation that reduces the dimension of the feature map [23]. Common pooling layers include the Max pooling, where the maximal value in each neighbourhood is taken, and the Average pooling, where the average value in each neighbourhood is selected. There are also a Global Maximal Pooling layer and a Global Average Pooling layer, where global maximal/average is taken.

Activation layers increase the non-linearity and the learning ability of the CNN. After several blocks of convolution, pooling, and activation layers, we can send the feature map, into fully connected layers to generate the final output.

Considering CNN's astonishing power for making predictions based on image inputs, we use it as the main tool to approximate the effective Hamiltonian in 2D. According to Equation 3.1,

Going back to the central problem of constructing a surrogate functional for H, we again split the effective Hamiltonian H into its enthalpic and entropic parts and use a CNN to learn the latter, which here we refer to as the remainder. Our proposed

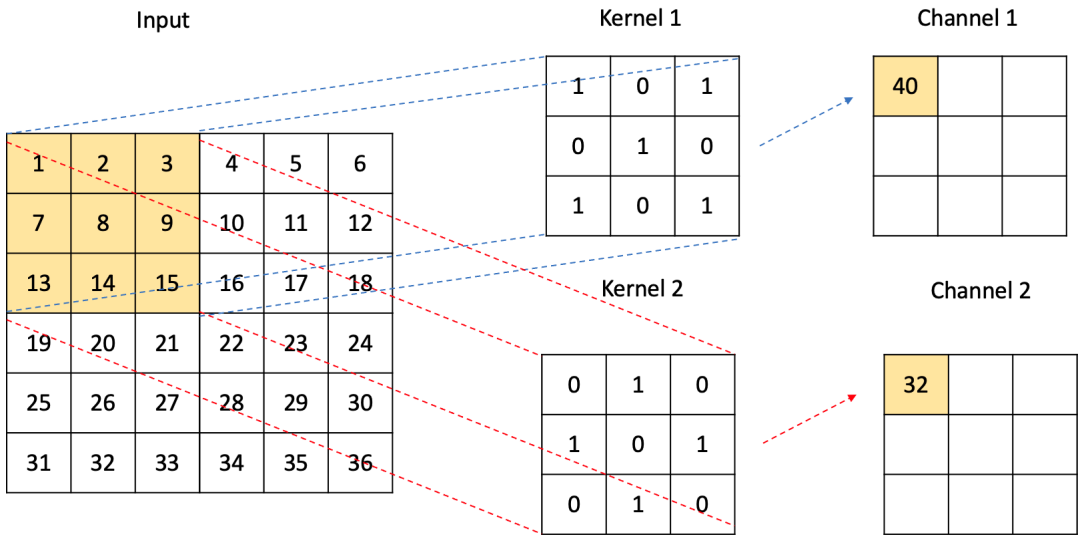


Figure 3.1: Illustration of convolutional layer

approximation has the form

$$\tilde{H}(x) = \chi N f - \frac{\chi N}{|\vec{a}_1 \times \vec{a}_2|} \int \rho^2 dr + NN(x), \tag{3.2}$$

where $x = (\chi N, \vec{a}_1, \vec{a}_2, f, \rho)$ and $NN(\cdot)$ is a CNN learner that takes density fields as image inputs and process the information of parameters in the fully connected layer. The specific architecture of the proposed CNN is provided in Section 3.3.2.

3.3.2 Rotation and Translation Invariance

The main goal of Problem 1 is to produce an accurate approximation of H which is invariant under both translations and rotations of the density field. Rotation invariance and translation invariance is illustrated in Figure 3.2, where picture (c) results from the rotation of picture (a), and picture (b) results from the translation of picture (a). All three share the same effective Hamiltonian. We opt to represent the 2D density fields by the matrices corresponding to the values of H at equi-spaced grids. Mathematically,

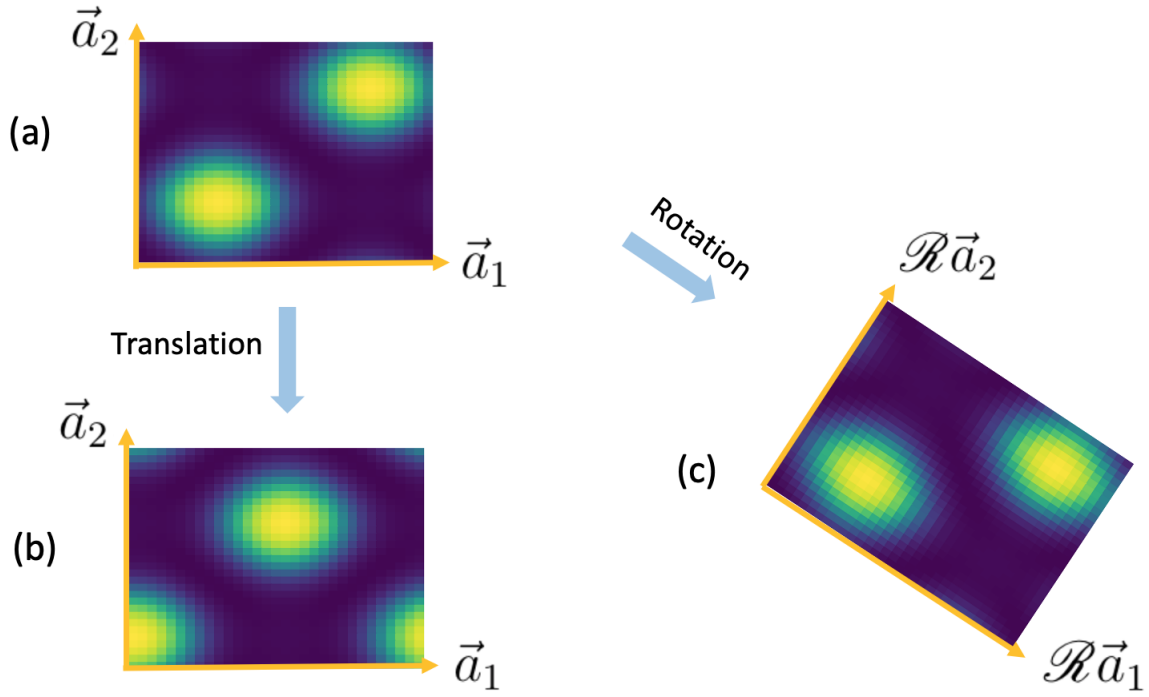


Figure 3.2: Rotation invariance and translation invariance.

setting the input parameters and the density field as $x = (\chi N, \vec{a}_1, \vec{a}_2, f, \rho)$, the rotation of density fields could be represented by

$$\mathscr{R}x = (\chi N, \mathscr{R}\vec{a}_1, \mathscr{R}\vec{a}_2, f, \mathscr{R}\rho) = (\chi N, \mathscr{R}\vec{a}_1, \mathscr{R}\vec{a}_2, f, \rho),$$

which means that the rotation of the density field is equivalent to rotate the two cell vectors, given the fixed arrangement of density field matrix. Thus, the rotation invariance requirement can be succinctly expressed as

$$\tilde{H}(\mathscr{R}x) = \tilde{H}(x). \tag{3.3}$$

The main part of $\tilde{H}$, given in Equation 3.2, is invariant under rotation because $|\mathscr{R}\vec{a}_1 \times \mathscr{R}\vec{a}_2| = |\vec{a}_1 \times \vec{a}_2|$. Therefore, we seek a learner with the property

$$NN(\mathscr{R}x) = NN(x). \tag{3.4}$$

To equip the learner with rotation invariance, we employ a rotation-invariant representation of ($\vec{a}_1$, $\vec{a}_2$) which keeps all the necessary information of the adjacent vectors. This is the natural representation, (l_1, l_2, θ), where $l_1 = \|\vec{a}_1\|$, $l_2 = \|\vec{a}_2\|$ and θ is the angle between $\vec{a}_1$ and $\vec{a}_2$. This new representation is evidently invariant under rotation, i.e., $\|\vec{a}_i\| = \|\mathscr{R}\vec{a}_i\|, i = 1, 2$ and $\theta(\vec{a}_1, \vec{a}_2) = \theta(\mathscr{R}\vec{a}_1, \mathscr{R}\vec{a}_2)$, thus using this representation as input ensures the output of the CNN preserves strongly rotation invariance. Henceforth, we set $x = (\chi N, l_1, l_2, \theta, f, \rho)$, which is equivalent to the previous definition but has rotation invariance. Equation 3.2 can now be rewritten as,

$$\tilde{H}(x) = \chi N f - \frac{\chi N}{|l_1 l_2 \cos\theta|} \int \rho^2 dr + NN(x). \tag{3.5}$$

We now address the constraint of translation invariance. Mathematically, translation invariance is described as

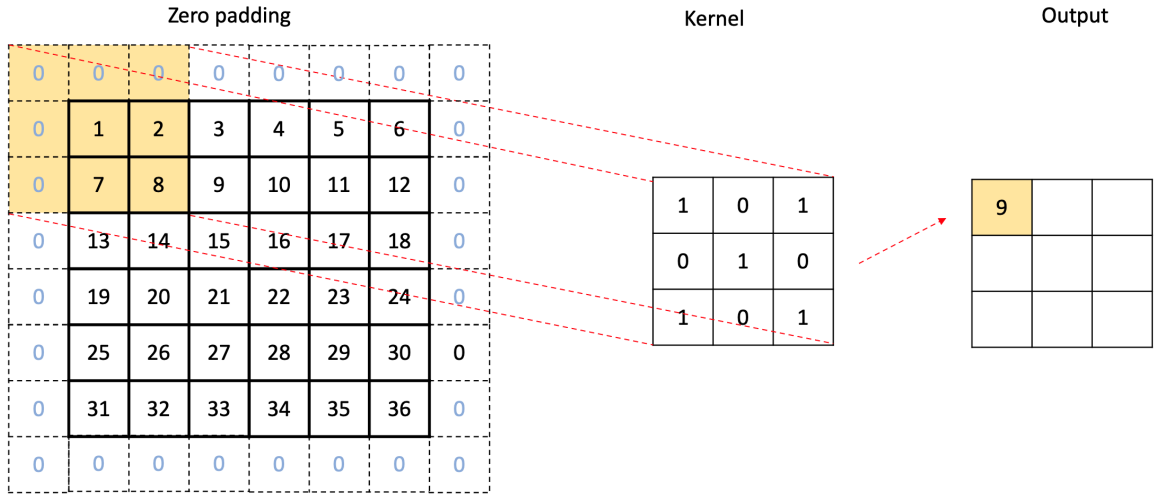
$$\tilde{H}(\mathscr{T}x) = \tilde{H}(x), \tag{3.6}$$

where $\mathscr{T}x = (\chi N, l_1, l_2, \theta, f, \mathscr{T}\rho)$ and $\mathscr{T}\rho(i_0+i, j_0+j) = \mathscr{T}\rho(i_0, j_0)$ for some (i, j) and all (i_0, j_0). To build a CNN with the global invariance property, we design the network using circular padding. In the convolutional layer and the pooling layer of neural networks, the element-wise product of the kernel and the input near the boundary is not well-defined because the boundary element does not have neighbours outside of the boundary. The most common solution is to use zero padding, where we assume that there are neighbours with values zero near the boundary to compute the convolution, as shown in Figure 3.3a.

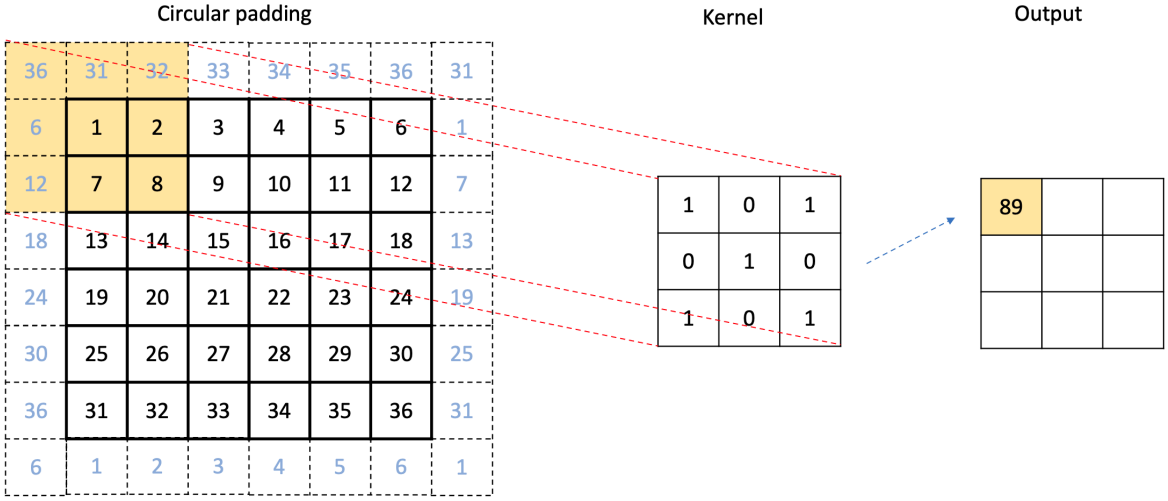
However, convolution with zero padding unfortunately does not have the translation invariance. Therefore, we endow the CNN with circular padding, which is similar to periodic boundary conditions in PDEs, where we append the values from the opposite boundary of the cell to each boundary. As shown in Figure 3.3b, the periodic boundary allows the output tensor to have the same elements with a shift. Note that, in order to make the operator commutative to the translation operator, the convolutional layer and pooling layer should have stride 1. We denote a convolution with size k, stride 1 and circular padding as $Conv^c_{k,1,p(k)}()$, where $p(k)$ is the size of padding which equals to $(k-1)/2$ when k is odd and $k/2$ when k is even. Similarly, we denote the Average Pooling layer with size k, stride 1 and circular padding as $AvgP^c_{k,1,p(k)}()$, the Maximal Pooling layer with size k, stride 1 and circular padding as $MaxP^c_{k,1,p(k)}()$, global average layer as $GlA()$, global maximal layer as $GlM()$, and element-wise activation function layer as $AcF()$, (e.g. Relu as $Relu()$, Sigmoid function as $Sigmoid()$,....), fully connected layers as $FC()$.

Lemma 3 *$Conv^c_{k,1,p(k)}()$, $AvgP^c_{k,1,p(k)}()$, $MaxP^c_{k,1,p(k)}()$ and $AcF()$ all commute with the translation $\mathscr{T}$. $GlM()$ and $GlA()$ are invariant under translation.*

Proof: Suppose the input tensor is I, a spatial translation of I is $\mathscr{T}I$. Then after the convolutional layer with stride 1 and circular padding, $Conv^c_{k,1,p(k)}(\mathscr{T}I)$ only differs from $Conv^c_{k,1,p(k)}(\mathscr{T}I)$ by a shift because they have the same set of neighbourhood batches and only the starting batch changes. So $Conv^c_{k,1,p(k)}(\mathscr{T}I) = \mathscr{T}Conv^c_{k,1,p(k)}(I)$. Similarly, $MaxP^c_{k,1,p(k)}(\mathscr{T}I) = \mathscr{T}MaxP^c_{k,1,p(k)}(I)$, $AvgP^c_{k,1,p(k)}(\mathscr{T}I) = \mathscr{T}AvgP^c_{k,1,p(k)}(I)$. Since AcF is element-wise operation, $AcF(\mathscr{T}I) = \mathscr{T}AcF(I)$. Trivially, $GlM(\mathscr{T}I) = GlM(I)$ and $GlA(\mathscr{T}I) = GlA(I)$. ■



(a) Zero padding



(b) Circular padding

Figure 3.3: Zero padding and circular padding.

We design the CNN with the following architecture:

$$\rho \to \underbrace{Conv^{c}_{5,1,2}() \to Relu() \to ... \to Conv^{c}_{5,1,2}() \to Relu()}_{\times 5} \to GlA() \\ \to \text{concat with } (\chi N, l_1, l_2, \theta, f) \underbrace{\to FC()}_{\times 3}, \tag{3.7}$$

where "concat with $(\chi N, l_1, l_2, \theta, f)$" means the concatenation of the numbers derived in each channel and the parameters. After $GlA()$ or $GlM()$, the output is in the form of several channels and each channel only has one single number (global maximum or global average), say $(c_1, c_2, ...c_m)$ where m is the number of channels. So we could concatenate the vector of all the channels with the parameters to get $(c_1, c_2, ...c_m, \chi N, l_1, l_2, \theta, f)$, and put them into several fully connected layers to make the final predictions.

Theorem 4 *The convolutional neural network in Equation 3.7 has shift invariance with respect to ρ.*

Proof: From Lemma 3,

$$GlA \circ (AcF \circ Conv^c_{k,1,p(k)})^n(\mathscr{T}\rho) = GlA \circ (\mathscr{T} \circ (AcF \circ Conv^c_{k,1,p(k)})^n(\rho)) \tag{3.8}$$

$$= GlA \circ (AcF \circ Conv^c_{k,1,p(k)})^n(\rho), \tag{3.9}$$

for any n, where $\circ$ stands for composition of operators. Since each channel outputs only one single number in the end, the fully connected layers preserves the translation invariance. Our case is a special case with $AcF = Relu$, $k = 5$ and $n = 5$, which inherits the translation invariance. ■

Note that we could also add the pooling layers with circular padding into the neural network, e.g. $AvgP^c_{k,1,p(k)}()$, $MaxP^c_{k,1,p(k)}()$, and the network would still preserve translation invariance according to Lemma 3. Since the architecture designed in Equation 3.7 without adding the pooling layers after convolutional layers already works very well, we did not include it for computation efficiency.

Based on the considerations above, we have effectively constructed an approximation $\tilde{H}$ in Equation 3.2 with $NN(x)$ in Equation 3.7, and $\tilde{H}$ keeps rotation and translation

invariance, i.e.,

$$\tilde{H}(\mathscr{R}x) = \tilde{H}(x),$$
$$\tilde{H}(\mathscr{T}x) = \tilde{H}(x).$$

3.3.3 Convolutional Neural Network in Sobolev Space

In Section 3.3.2, we developed the convolutional neural network to approximate effective Hamiltonian H with the rotation/translation invariance. In this section, we propose to train the CNN in Sobolev space to fully solve Problem 1 and thus obtain a powerful tool to solve Problem 2.

As in the one-dimensional case, all the training points are saddle points which should have vanishing Hamiltonian gradients on the training set. We encode this valuable information into the objective function to train an approximation $\tilde{H}$ which matches both the Hamiltonian and its gradient. This approach has many advantages, for instance, it makes the prediction more accurate by imitating the gradient behaviour of the true system, and it ensures the approximation has vanishing gradients on saddle points which is significant for the predicted field in Problem 2.

Considering the vanishing gradient at saddle points, we select parameters of the CNN to minimize the obejctive function,

$$C(\alpha) = \sum_{i=1}^{N_T} (\tilde{H}(x_i) - H_i)^2 + \beta \sum_{i=1}^{N_T} \|\nabla_\rho \tilde{H}(x_i)\|^2, \tag{3.10}$$

where β controls the importance of the penalty terms on gradients and favours vanishing gradients on saddle points and also serves as a regularization for smoothness.

For each data point (x_i, H_i), according to Equation 3.1 and Equation 3.2, $NN(x_i)$

should match

$$R_i = H_i - \chi N_i f_i + \frac{\chi N_i}{|l_{1i} l_{2i} \cos\theta_i|} \int \rho_i^2 dr. \tag{3.11}$$

Based on the spatial discretization,

$$\nabla_\rho \tilde{H}(x_i) = -\frac{2\chi N_i}{l_{1i} l_{2i}} \Delta l_{1i} \, \Delta l_{2i} \, \rho_i + \nabla_\rho NN(x_i), \tag{3.12}$$

where Δl_1 and Δl_2 are the spatial mesh size of the l_1 direction and l_2 direction. Then we rewrite the cost function in Equation 3.10 as

$$C(\alpha) = \sum_{i=1}^{N_T} (NN(x_i) - R_i)^2 + \beta \sum_{i=1}^{N_T} \|\nabla_\rho NN(x_i) - \frac{2\chi N_i}{l_{1i} l_{2i}} \Delta l_{1i} \, \Delta l_{2i} \, \rho_i\|^2, \tag{3.13}$$

which is in the form of typical Sobolev norm, where we aim to train a map which matches both the functional and functional gradients by minimizing the objective function. Since the neural network is trained in Sobolev space, it is expected to predict the Hamiltonian and its gradient simultaneously. At the same time, with the architecture described in Equation 3.7, the approximation preserves rotation and translation invariance. With all the pieces together, $\tilde{H}$ is able to approximate the effective Hamiltonian in Problem 1 and also its gradient, which is useful for the saddle point density predictions in Problem 2.

3.4 Methodology: Saddle Point Density Field Prediction by Generative Adversarial Networks

In Section 3.3, we developed a Sobolev space-trained CNN with strong rotation and translation invariance, i.e. we fully solved Problem 1. In this section, we focus on the solution to Problem 2 by employing a generative adversarial network (GAN) and the Sobolev space-trained CNN of Problem 1.

3.4.1 GAN, cGAN and DCGAN

Our methodology to predict saddle points is built by merging GAN, cGAN (Conditional GAN) and DCGAN (Deep Convolutional GAN). In this section, we briefly introduce the three architectures and the motivation to apply it on saddle points prediction.

A GAN [22] is a network where a Generator (G) is trained to generate, from random noise, images similar to the images in the training set. A GAN has also a Discriminator (D) which is trained to estimate the probability that the input image is from the training set rather than from the Generator. After several rounds of training, G is expected to generate images which could fool D and D is expected to tell the difference from the true image and the fake image. In the end, G takes random noise as input and could generate images that imitate the training set. For instance, if the training set are pictures of human faces, G is expected to generate pictures of human faces because it learned how to extract the latent features to the type of images. The loss function for a GAN is

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))]. \tag{3.14}$$

Based on the loss function, D is trained to maximize the probability that $D(x) = 1$ for all the x in the training set and the probability that $D(G(z)) = 0$ for all the images generated by G. G is trained to minimize the probability that D identifies the fake images and fool D by driving $D(G(z))$ to 1.

Conditional GAN (cGAN) was proposed in [24] to send labels to the Generator to generate the images corresponding to the label, for example the cGAN trained on MNIST[2], which is a dataset containing the images of 60K handwritten digits. The architecture and training process is similar to GAN, with a difference that the Generator has an extra input, the labels. Thus, the Generator has two types of inputs, random noise and labels of

[2] `http://yann.lecun.com/exdb/mnist/`

the digits. After training, the Generator is expected to generate the digits corresponding to the true label.

DCGAN [25] is an extended version of GAN, with an architecture where D is mainly made up of convolutional layers and G is mostly composed of transposed convolutional layers. A transposed convolutional layers is an upsampling layer, whose output has larger dimension size than the input. A detailed introduction of transposed convolutional layer is can be found here[3]. Since the transposed convolutional layers have the ability to expand dimensions, we could see the noise vector of dimension n as an image with n channels and size 1×1 in each channel. Then by repeating transposed convolutions, each channel could be expanded into the image size, e.g. 32×32. With the transposed convolutional layers in the Generator and convolutional layers in the Discriminator, DCGAN could generate higher quality images.

3.4.2 ScftGAN

As described above, GAN-based models try to learn the relationship between the latent feature space and the images with common styles, then generate the similar-style images from a random noise with fixed dimensions, thus lying in a subspace considered as latent feature space. In our Problem 2, we can regard all the saddle point density fields as points on a high dimensional manifold. Could GAN learn the map from a latent feature space equivalent to the saddle point manifold? If so, we could use the Generator to generate density fields from the saddle density field manifold. In order to predict saddle density fields corresponding to the specific combination of parameters, (χN^*, l_1^*, l_2^*, θ^*, f^*), we could pass the parameters to the Generator (G) as done in a cGAN. After training with the structure of saddle point density field manifold together with the relationship between parameters and saddle point density fields, G is expected to

[3] https://towardsdatascience.com/what-is-transposed-convolutional-layer-40e5e6e31c11

generate saddle point density fields corresponding to the parameters from random noise, drawn from the latent space of the saddle point manifold.

The new GAN architecture we propose for saddle points density fields, which we henceforth call ScftGAN, is shown in Figure 3.4. G generates "fake" density fields from random noise and the parameters. The fake density fields together with the parameters (shown as Information in Fig. 3.4) are sent into Discriminator (D). These examples are labeled "fake" to train D. The true density fields are also sent into the D, labeled as "true" during the training process. With the education from the true examples and fake examples, D is trained to accurately classify the examples. In turn, the results of D are utilized to update G in a stochastic gradient descent method, by maximizing the probability that D identifies as "true" the fake examples (generated by the G). To educate the D with the relationship between parameters and the corresponding density fields, we also send the true density fields with the shuffled Information (parameters), i.e. assigning a random Information (parameters) from the pool to the density fields, and penalizing the probability that D classify these as true pairs, to avoid the risk that D makes decision without considering the parameters. The knowledge that only saddle point density fields corresponding to the input parameters should be generated, is enhanced in training. The loss function for our proposed ScftGAN is

$$\min_G \max_D V(D,G) = E_{\rho\sim p_{data}(\rho)}[\log D(\rho|y)] + E_{z\sim p_z(z)}[\log(1-D(G(z|y)|y))] \\ + E_{\rho\sim p_{data}(\rho)}[\log(1-D(\rho|S(y)))], \quad (3.15)$$

where y is the tuple of parameters (χN, l_1, l_2, θ, f) and $S(y)$ means a random shuffle of all the Information (parameters) in the data set. Compared with loss function defined in Equation 3.14 of the classical GAN, parameter information is sent to both G and D, and the additional term enforces D to output "0" for the input of wrong pairs of parameters

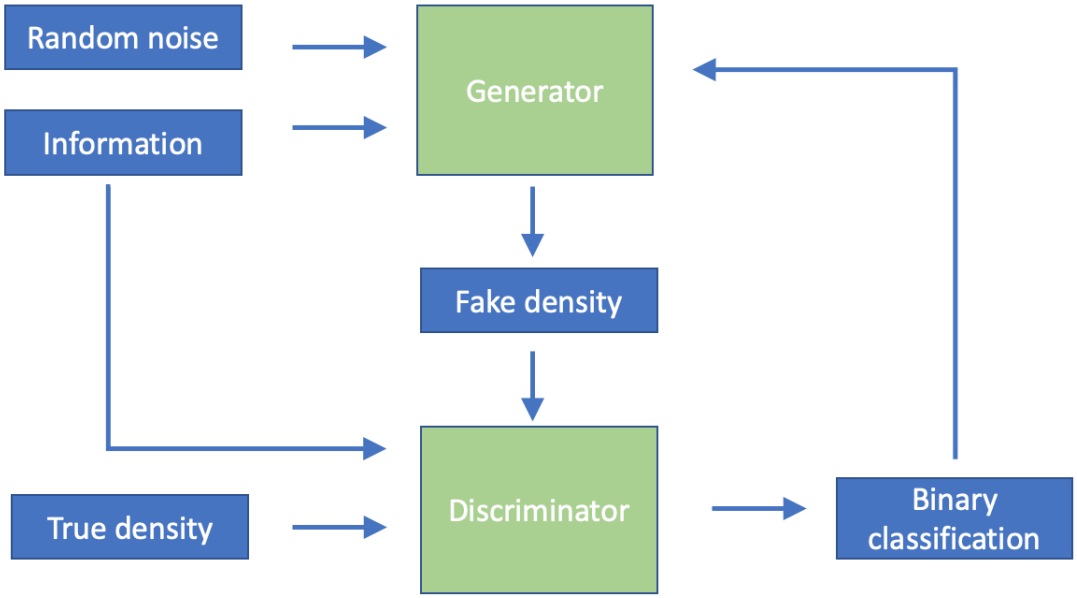


Figure 3.4: ScftGAN, generative adversarial networks for SCFT. The Information block represents parameters, i.e., (χN, l_1, l_2, θ, f).

and density fields, so that D could learn the correct relationship between parameters and saddle density fields.

The architecture of G is :

$$
\begin{aligned}
z \rightarrow \text{concat with } y &\rightarrow ConvT_{4,1,0}() \rightarrow Bnorm() \rightarrow Relu() \\
&\rightarrow \text{concat with } y \underbrace{\rightarrow ConvT_{4,2,1}() \rightarrow Bnorm() \rightarrow Relu()}_{\times 2} \\
&\rightarrow \text{concat with } y \rightarrow ConvT_{4,2,1}() \rightarrow Bnorm() \rightarrow Relu() \\
&\rightarrow \text{concat with } y \underbrace{\rightarrow ConvT_{5,1,2}() \rightarrow Bnorm() \rightarrow Relu()}_{\times 6} \\
&\rightarrow ConvT_{5,1,2}() \rightarrow Sigmoid()
\end{aligned} \tag{3.16}
$$

where $ConvT_{m,n,k}()$ stands for transposed Convolution of size m, stride n and zero padding of size k, $Bnorm()$ represents batch normalization, "concat with y" refers to

the operation that treats each parameter in y has a constant channel and append the parameter channels to the input tensor. We frequently repeat inputting the parameters to the neural networks to stress the importance of parameters in saddle density fields prediction.

The architecture of D is:

$$\begin{aligned}
&\rho \to \text{concat with } y \to Conv_{3,1,1}() \to LeakyRelu() \\
&\underbrace{\to ConvT_{4,2,1}() \to Bnorm() \to LeakyRelu()}_{\times 3} \\
&\to Conv_{4,1,0}() \to Sigmoid(),
\end{aligned} \tag{3.17}$$

the output is a number between 0 and 1 which measures the probability that the input density field ρ is the saddle density fields of the corresponding parameters y.

After training, given specific parameters (χN^*, l_1^*, l_2^*, θ^*, f^*), GAN could generate a batch of candidates for saddle density fields by taking various inputs of random noise. The result is shown in Section 3.5. Then we could use the Sobolev space-trained CNN in Section 3.3 to select the optimal density field, fine-tune the density field and predict the corresponding Hamiltonian.

3.4.3 Saddle Density Field Prediction

In Section 3.4.2, we introduced the ScftGAN to generate predictions of saddle density fields corresponding to the given parameters. Using various random noise, we could generate a pool of saddle density fields. In this section, we combine the predictions from ScftGAN with the Sobolev space-trained CNN to finalize the saddle density field prediction.

Since we do Sobolev space-training for the approximation of the Hamiltonian, the

saddle point density fields will yield small or vanishing Hamiltonian gradients. Therefore, we plug all the saddle point density fields generated by ScftGAN into $\tilde{H}$ and compute the gradients $\nabla_\rho \tilde{H}$. We select the density field in the pool with the smallest $\|\nabla_\rho \tilde{H}\|$ as the initial density field. Notice that in the one-dimensional work, we proposed to use the density field in training set with the smallest $\|\nabla_\rho \tilde{H}\|$ as initial density field. There are several advantages using this GAN-based generation of initial density field. First, we do not need to memorize the training set for the saddle point density fields prediction, which saves an enormous amount of computer memory. Second, the strong dependence of performance on the training set is weaken so that the algorithm is more robust. Third, the ScftGan could generate multiple candidates given different noise, which increases the diversity of the candidate pool and consequently the probability to discover new polymer phases.

After selecting the initial density field, we use gradient descent to fine-tune the density field to get a local minimizer of $\tilde{H}$, which is an approximation of a saddle point density field of the effective Hamiltonian. The gradient of $\tilde{H}$, i.e., which is an approximation to the true Hamiltonian gradient, can be computed explicitly by Equation 3.12, where $\nabla_\rho NN$ is easily computed by backward propagation. In each step of gradient descent, the saddle point density field prediction is updated by,

$$\rho^{n+1} = \rho^n - \epsilon \nabla_\rho \tilde{H}(x^n). \tag{3.18}$$

After the gradient descent method, we get the finalized prediction of the density field. The experimental results are shown in Section 3.5.

3.5 Results

In this section, we present results of numerical experiments for a two-dimensional AB diblock system to validate the efficiency of the proposed methods. The data are generated for the common range of parameters, i.e., $\chi N = 16$, $l_1 \in [3, 5]$, $l_2 \in [3, 5]$, $\theta \in [\pi/2, 5\pi/6]$, $f \in [0.3, 0.5]$. We sample data points on equidistributed nodes in the given interval of each parameter by running the direct SCFT solver to compute the corresponding saddle point density fields and the effective Hamiltonian. The modified diffusion equations are solved using periodic boundary conditions and pseudo-spectral collocation in space with 32×32 mesh points in 2D. Auxiliary fields, initialized with smooth fields with a fixed number of periods, are relaxed to saddle-point configurations using the semi-implicit Seidel iteration [5]. l_1, l_2 are varied rather than set to the value that minimizes the effective Hamiltonian in building the datasets. After building the dataset, we randomly pick 70% of the data as training set, 15% as the validation set, and 15% as the test set. In the generated dataset there are both hexagonal cylinders (HEX) and lamellar phase (LAM). Our experimental results show the capability of the Sobolev space-trained CNN to accurately approximate the effective Hamiltonian for both phases. In other words, we do not need to train separate learners for different phases.

3.5.1 Approximation of the Effective Hamiltonian

In this section, we show the performance of the Sobolev space-trained CNN to approximate the effective Hamiltonian and its gradient. Table 3.1 lists the mean square error for these quantities. The Sobolev space-trained CNN achieves 3 digits of accuracy on H prediction and 2 digits of accuracy on its gradient.

We randomly pick 30 data points form the test set and plot the predicted Hamiltonian and SCFT effective Hamiltonian. Figure 3.5 shows that the CNN-predicted prediction is

	Hamiltonian Error	Hamiltonian Gradient Error
Training set	0.0060	0.0129
Test set	0.0063	0.0128

Table 3.1: (Mean Square) Error of the CNN

accurate throughout the entire range of parameters.

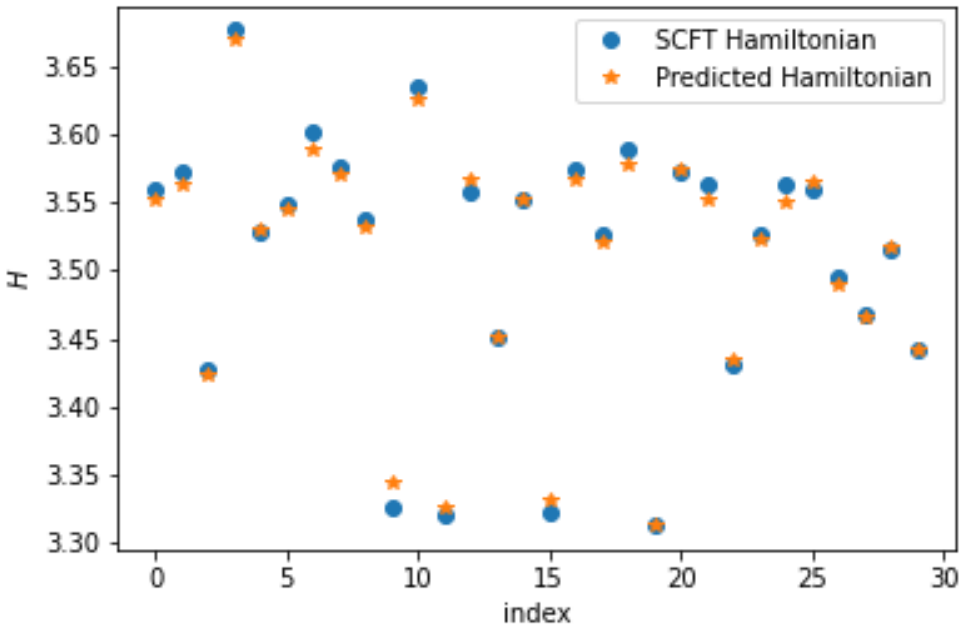


Figure 3.5: Comparison of effective Hamiltonian and the CNN approximation for a random set of 30 data points.

3.5.2 Saddle Point Density Field Predictions

In this section, we demonstrate the performance of our proposed ScftGAN for saddle point density prediction. At the start we train the ScftGAN, which takes random noise and parameter (χN^*, l_1^*, l_2^*, θ^*, f^*) as input. We show the performance of the ScftGAN on the validation set during the training process in Figure 3.6. For each subplot, the left columns are saddle density fields computed by direct SCFT simulation and the right

column are the saddle density fields predicted by the ScftGAN. In the beginning (epoch 0), the ScftGAN predicts random density fields but the predictions dramatically improve with the number of training epochs.

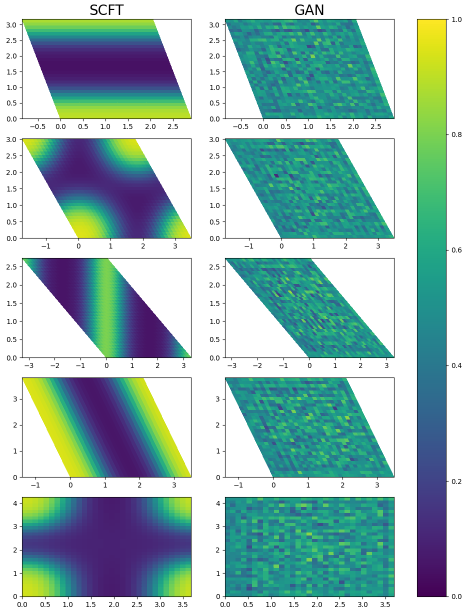
After training ScftGAN, for each combination of parameters, (χN^*, l_1^*, l_2^*, θ^*, f^*), we generate 20 saddle point density fields candidates based on the same number of random noise input. Then, we select the density field with smallest magnitude of gradient of the the Sobolev space-trained CNN approximation $\tilde{H}$. Finally, we employ the gradient descent method to fine-tune the density field prediction as shown in Equation 3.18. After the fine-tuning, the new density field yields a density field with a reduced Hamiltonian which suggest that this fine-tuning procedure leads to a more accurate minimizer.

The comparison of the Hamiltonian from the fine-tuned-density field and initial density field is displayed in Figure 3.7. The full procedures of saddle point density prediction is illustrated in Figure 3.8.

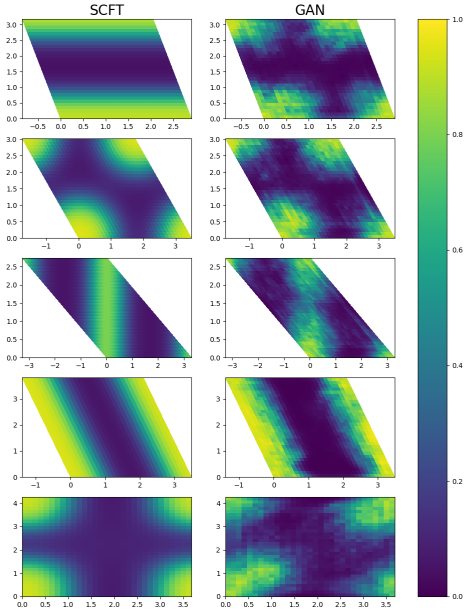
Finally, we take some representative cases from the test set to check the ultimate prediction of saddle point density fields. The results are summarized in Figure 3.9. The left column contains the saddle points density fields computed by direct SCFT and the right column has the density fields predicted by the ScftGAN with the final fine-tuning. This completes the solution to Problem 2.

3.6 Conclusions

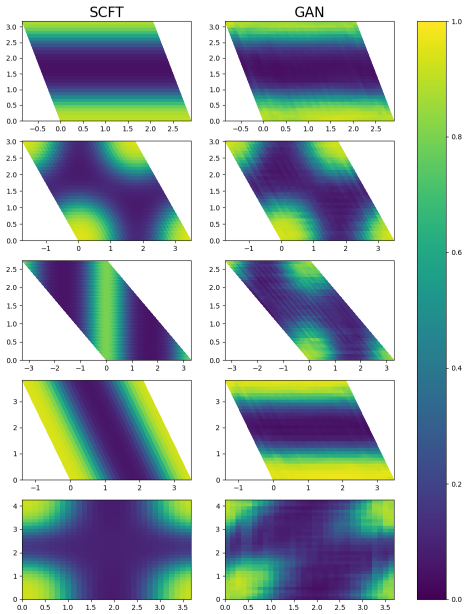
In this chapter, we solved two problems on the two-dimensional phase discovery. First, we developed a CNN in Sobolev space, which approximates the effective Hamiltonian and its gradient from the relevant physical parameters and corresponding saddle point density fields. This CNN approximation also preserves rotation invariance and translation invariance, which are necessary in SCFT theory. Second, we developed an algorithm



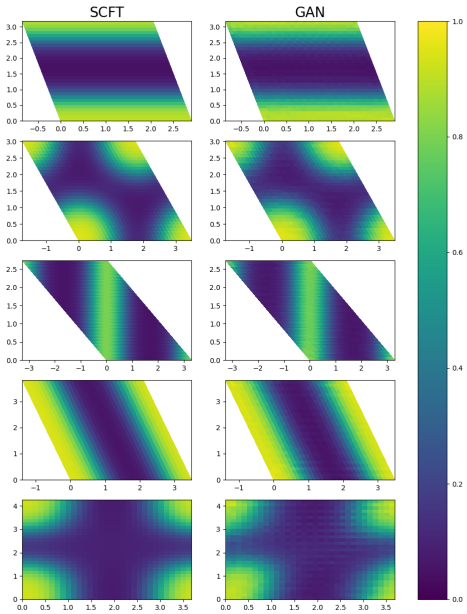
(a) epoch 0



(b) epoch 13



(c) epoch 121



(d) epoch 4999

Figure 3.6: Predictions of saddle point density fields on the validation set during the training of ScftGAN.

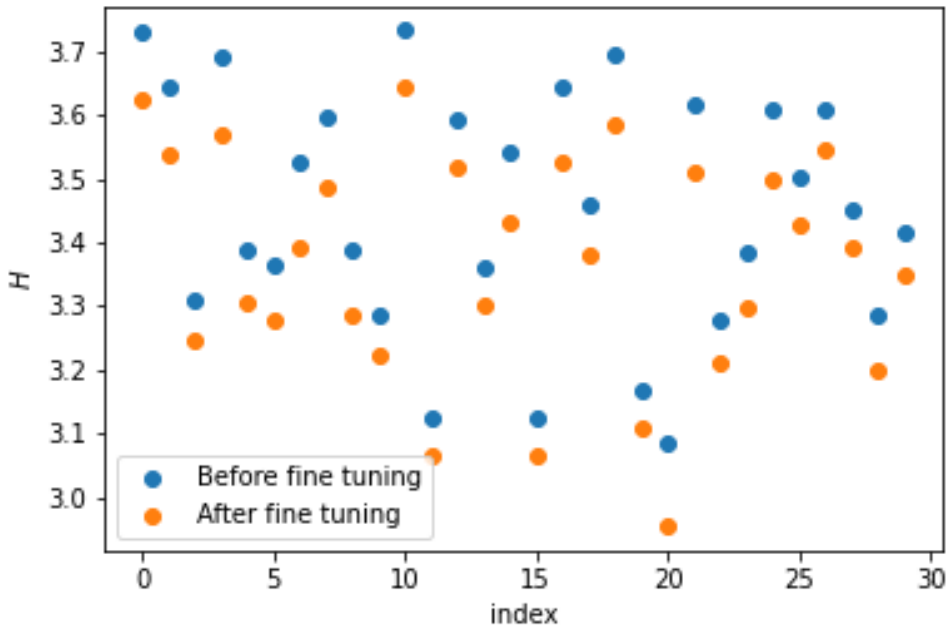


Figure 3.7: Comparison of the effective Hamiltonian from the fine-tuned-density field and initial density field

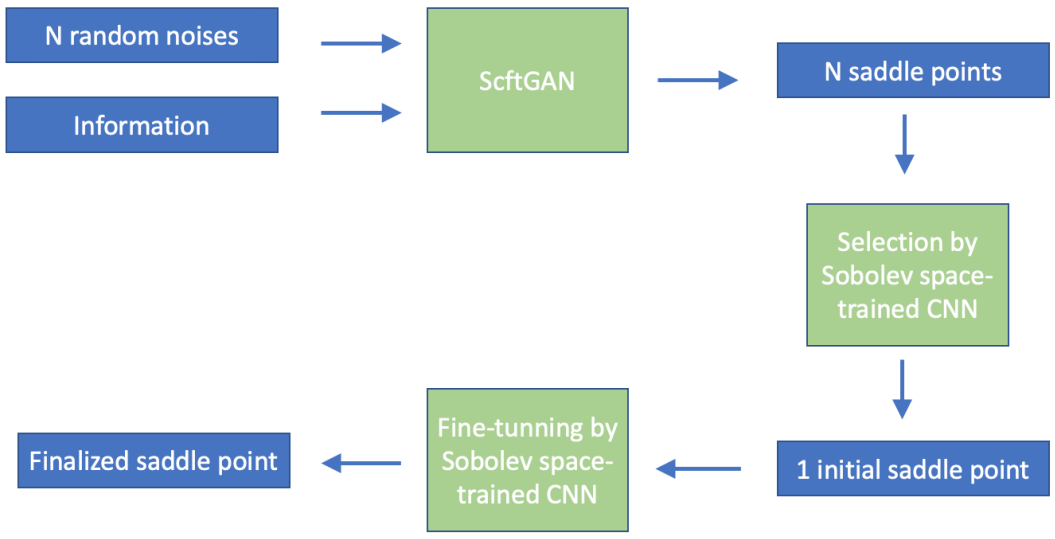


Figure 3.8: Procedures of saddle point density fields prediction. The Information block represents parameters, i.e., (χN, l_1, l_2, θ, f).

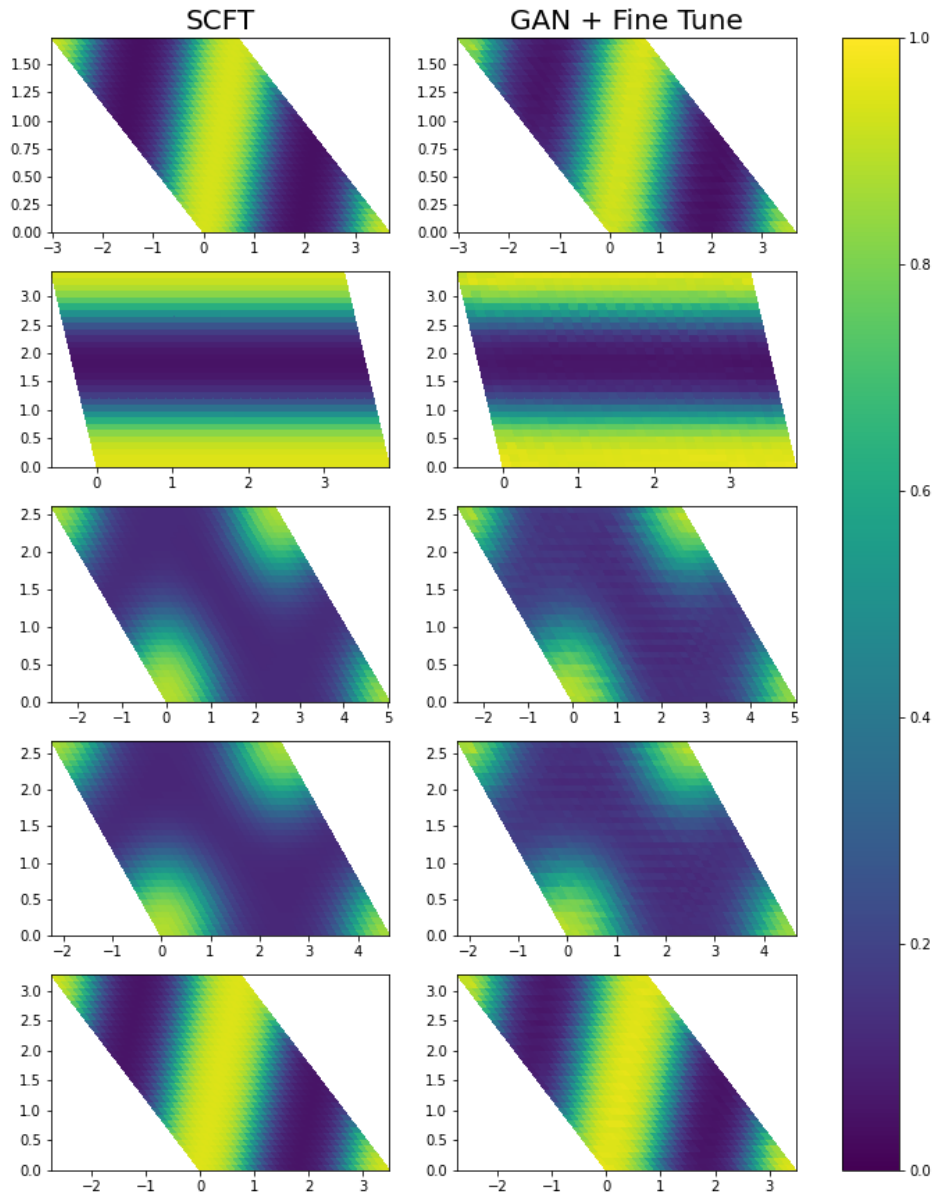


Figure 3.9: Saddle point density fields predictions for representative cases in the test set.

to generate saddle density fields from the physical parameters by using GAN's and a fine-tuning process of the density fields using the Sobolev space-trained CNN and the gradient descent method. It is important to note that the Sobolev space-trained CNN yields an accurate approximation of H (ands its gradient) which is fast to evaluate. In fact, we could also equip the trained-CNN with other screening methods to enhance the prediction of saddle point density fields, such as Markov chain Monte Carlo or the Hamiltonian Monte Carlo algorithms.

Chapter 4

Optimal Policies for a Pandemic: A Stochastic Game Approach and a Deep Learning Algorithm

4.1 Introduction

The pandemic of coronavirus disease 2019 (COVID-19) has brought a huge impact on our lives. Based on the CDC Data Tracker, as of early December 2020, there have been more than 15 million confirmed cases of infection and more than 290 thousand cases of death in the United States. Needless to say, the economic impact has also been catastrophic, resulting in unprecedented unemployment and the bankruptcy of many restaurants, recreation centers, shopping malls, etc.

In a classic, compartmental epidemiological model each individual is assigned a label, *e.g.*, **S**usceptible, **E**xposed, **I**nfectious, **R**emoved, **V**accinated. The labels' order shows the flow patterns between the compartments (SIR, SEIR, SIRV models). Other approaches include network models, which explicitly include the interaction of individuals,

in addition to the modeling of each individual's dynamics, and agent-based models that are useful in informing decision making when accurately calibrated. Moreover, the consideration of pharmaceutical and/or non-pharmaceutical intervention policies naturally couples game theory to epidemiological models by controlling when and how the game is played in such models. For example, in some early studies [26, 27], one can use non-repeated games to incorporate game theory into modeling at the individual level, where individuals (known as "players" in the game theory) maximize their gain by weighing the costs and benefits of different strategies. We refer to the review paper [28] and the references therein for more details.

Differential games, initiated by [29], as an offspring of game theory and optimal control, provide modeling and analysis of conflict in the context of a dynamical system. They have been intensively employed across many disciplines, including management science, economics, social science, biology, military, etc. One of the core objectives in differential games is to compute Nash equilibria that refer to strategies by which no player has an incentive to deviate. However, a major bottleneck comes from the notorious intractability of N-player games, and the direct computation of Nash equilibria is extremely time-consuming and memory demanding. In a series of recent works by [30, 31, 32, 33], the deep fictitious play (DFP) theory and algorithms were developed for stochastic differential games (SDG) with a large number of heterogeneous players. The DFP framework embeds the fictitious play idea, introduced by [34, 35], into designed architectures of deep neural networks to produce accurate and parallelizable algorithms with convergence analysis, and resolve the intractability issue (curse of dimensionality) caused by the complex modeling and underlying high-dimensional space in SDG.

Building from the DFP theory and algorithms for computing Nash equilibria in SDG, we propose here to strengthen the classical SEIR model by taking into account the social and health policies issued by multiple region planners. We call this new model a stochastic

multi-region SEIR model because it couples the stochastic differential game theory with the SEIR model, making it more realistic and powerful. The computational challenge introduced by the high-dimensionality of the multi-region solution space is addressed by generalizing the deep fictitious algorithm proposed by [31]. This new approach leads to an enhanced deep fictitious play algorithm to overcome the curse of dimensionality and further reduce the computational complexity. To showcase the performance of the proposed model and algorithm, we apply them to a case study of the COVID-19 pandemic in three states: New York (NY), New Jersey (NJ), and Pennsylvania (PA). We present the optimal lockdown policy corresponding to the Nash equilibrium of the multi-region SEIR model. We remark that our work is not to predict a pandemic, but to provide a game-themed framework, a deep learning algorithm and possible outcomes for competitive region planners. We hope that information can provide some qualitative guidance for policymakers on the impact of certain policies. The parameters used in the numerical experiments are based on the current knowledge of the Coronavirus, which is still under development. In practice, at the beginning of the Coronavirus, a governor might not be able to trace the infections fully, and infected cases may not be fully identified. All these may lead to the inaccuracy of parameter estimations despite our using of the best available data. Therefore, it may be hard to match the predicted results with practical observation.

The rest of the paper is organized as follows. In Section 4.2, we propose a novel multi-region epidemiological model and explain how social and health policies issued by region planners are integrated into dynamics, leading to a game feature of the problem. We explain the numerical challenge in Section 4.3 and propose an enhanced deep fictitious play algorithm for memory and computational efficiency. Section 4.4 focuses on a NY-NJ-PA case study with detailed discussions on parameter choices and the resulted optimal policies. We present some concluding remarks in Section 4.5 and provide technical details

in the appendices.

4.2 Mathematical modeling: A multi-region SEIR model

We consider a pandemic spreading in N geographical regions, and each planner controls the loss of her region by implementing some policies. We aim to study how the region planners' policies affect each other, and the equilibrium policies.

Let us start with a modified version of the very-known epidemic SEIR model (cf. [36]), where each region's population is assigned to compartments with four labels: **S**usceptible, **E**xposed, **I**nfectious, and **R**emoved. Individuals with different labels denote S: those who are not yet infected; E: who have been infected but are not yet infectious themselves; I: who have been infected and are capable of spreading the disease to those in the susceptible category, and R: who have been infected and then removed from the disease due to recovery or death. The region planners can issue certain policies to mitigate the pandemic, for instance, policies that can help reduce the transmission rates and death rates. Mathematically, denote by $S_t^n, E_t^n, I_t^n, R_t^n$ the *proportion* of population in the four compartments of region n at time t. We consider the following stochastic multi-region SEIR model:

$$\mathrm{d}S_t^n = -\sum_{k=1}^{N} \beta^{nk} S_t^n I_t^k (1-\theta\ell_t^n)(1-\theta\ell_t^k)\,\mathrm{d}t - v(h_t^n) S_t^n\,\mathrm{d}t - \sigma_{s_n} S_t^n\,\mathrm{d}W_t^{s_n}, \tag{4.1}$$

$$\mathrm{d}E_t^n = \sum_{k=1}^{N} \beta^{nk} S_t^n I_t^k (1-\theta\ell_t^n)(1-\theta\ell_t^k)\,\mathrm{d}t - \gamma E_t^n\,\mathrm{d}t + \sigma_{s_n} S_t^n\,\mathrm{d}W_t^{s_n} - \sigma_{e_n} E_t^n\,\mathrm{d}W_t^{e_n}, \tag{4.2}$$

$$\mathrm{d}I_t^n = (\gamma E_t^n - \lambda(h_t^n) I_t^n)\,\mathrm{d}t + \sigma_{e_n} E_t^n\,\mathrm{d}W_t^{e_n}, \tag{4.3}$$

$$\mathrm{d}R_t^n = \lambda(h_t^n) I_t^n\,\mathrm{d}t + v(h_t^n) S_t^n\,\mathrm{d}t, \quad n \in \mathcal{N} := \{1, 2, \ldots, N\}, \tag{4.4}$$

where $\boldsymbol{\ell}_t \equiv (\ell_t^1, \ldots, \ell_t^N)$ and $\boldsymbol{h}_t \equiv (h_t^1, \ldots, h_t^N)$ are policies chosen by the region planners at time t. Each planner n seeks to minimize its region's cost within a period $[0, T]$:

$$J^n(\boldsymbol{\ell}, \boldsymbol{h}) := \mathbb{E}\bigg[\int_0^T e^{-rt} P^n\big[(S_t^n + E_t^n + I_t^n)\ell_t^n w + a(\kappa I_t^n \chi + p I_t^n c)\big] + e^{-rt}\eta(h_t^n)^2 \,\mathrm{d}t\bigg]. \quad (4.5)$$

We now give detailed description of this model (4.1)–(4.5):

S: β^{nk} denotes the average number of contacts per person per time. The transition rate between S^n and E^n due to contacting infectious people in the region k is proportional to the fraction of those contacts between an infectious and a susceptible individual, which result in the susceptible one becoming infected, *i.e.*, $\beta S_t^n I_t^k$. Although some regions may not be geographically connected, the transmission between the two is still possible due to air travels but is less intensive than the transmission within the region, *i.e.*, $\beta^{nk} > 0$ and $\beta^{nn} \gg \beta^{nk}$ for all $k \neq n$.

$\ell_t^n \in [0, 1]$ denotes the decision of the planner n on the fraction of population being locked down at time t. We assume that those in lockdown cannot be infected. However, the policy may only be partially effective as essential activities (food production and distribution, health, and basic services) have to continue. Here we use $\theta \in [0, 1]$ to measure this effectiveness, and the transition rate under the policy $\boldsymbol{\ell}$ thus become $\beta^{nk} S_t^n I_t^k (1 - \theta\ell_t^n)(1 - \theta\ell_t^k)$. The case $\theta = 1$ means the policy is fully effective.

$h_t^n \in [0, 1]$ denotes the effort the planner n decide to put into the health system, which we refer as *health policy*. It will influence the vaccination availability $v(\cdot)$ and the recovery rate $\lambda(\cdot)$ of this model.

$v(h_t^n)$ denotes the vaccination availability of region n at time t. Once vaccinated, the susceptible individuals $v(h_t^n)S_t^n$ become immune to the disease, and join the

removed category R_t^n. We model it as an increasing function of h_t^n, and if the vaccine has not been developed yet, we can define $v(x) = 0$ for $x \leq \overline{h}$.

E: γ describes the latent period when the person has been infected but not infectious yet. It is the inverse of the average latent time, and we assume γ to be identical across all regions. The transition between E^n and I^n is proportional to the fraction of exposed, *i.e.*, γE_t^n.

I: $\lambda(\cdot)$ represents the recovery rate. For the infected individuals, a fraction $\lambda(h^n)I^n$ (including both death and recovery from the infection) joins the removed category R^n per time unit. The rate is determined by the average duration of infection D. We model the duration (so does the recovery rate) related to the health policy h_t^n decided by its planner. The more effort put into the region (*i.e.*, expanding hospital capacity, creating more drive-thru testing sites), the more clinical resources the region will have and the more resources will be accessible by patients, which could accelerate the recovery and slow down death. The death rate, denoted by $\kappa(\cdot)$, is crucial for computing the cost of the region n; see the next item.

Cost: Each region planner faces four types of cost. One is the economic activity loss due to the lockdown policy, where w is the productivity rate per individual, and P^n is the population of the region n. The second one is due to the death of infected individuals. Here κ is the death rate which we assume for simplicity to be constant, and χ denotes the cost of each death. The hyperparameter a describes how planners weigh deaths and infections comparing to other costs. The third one is the inpatient cost, where p is the hospitalization rate, and c is the cost per inpatient day. The last term $\eta(h_t^n)^2$ is the grants putting into the health system. We choose a quadratic form to account for diminishing marginal utility (view it from $\eta(h_t^n)^2$ to h_t^n). All costs are discounted by an exponential function e^{-rt}, where r is the risk-free interest

rate, to take into account the time preference. Note that region n's cost depends on all regions' policies $(\boldsymbol{\ell}, \boldsymbol{h})$, as $\{I^k, k \neq n\}$ appearing in the dynamics of S^n. Thus we write is $J^n(\boldsymbol{\ell}, \boldsymbol{h})$.

The choices of epidemiological parameters will be discussed in Section 4.4.1. Next, we summarize the key assumptions in the above model:

1. The dynamics of an epidemic are much faster than the vital (birth and death) dynamics. So vital dynamics are omitted in the above model.

2. The planning is of a short horizon and will be adjusted frequently as the epidemic develop. For simplicity, we assume this is no migration between regions over the time $[0, T]$.

3. Individuals who once recovered from the disease, are immune and free of lockdown policy.

4. The dynamics obeys the conservation law: $S_t^n + E_t^n + I_t^n + R_t^n = P^n$. This means that the process R^n is redundant.

5. The dynamics of S, E and I are subjected to random noise, to account for the noise introduced during data recording, false-positive/negative test results, exceptional cases when recovered individuals become susceptible again, minor individual differences in the latent period, etc.

6. Individuals who are not under lockdown have the same productivity, no matter their categories. We assume this for simplicity remark that this can be improved by assigning different productivity to individuals with or without symptoms.

The above modeling and objectives can be viewed as a stochastic differential game between N players[1]. Here, we view the whole problem as a non-cooperative game, as

[1]Henceforth, we shall use *planner* and *player* interchangeably.

many regions make decisions individually and indeed even compete for scarce resources (frontline workers, personal protective equipment, etc.) during the outbreak. Each player n controls her states (S^n, E^n, I^n, R^n) through her strategy (ℓ^n, h^n) in order to minimize the associated cost J^n. The optimizers then are interpreted as the optimal lockdown policy and optimal effort putting into the health system.

For a non-cooperative game, one usually refers to Nash equilibrium as a notion of optimality. For completeness, we review the definition here.

Definition 4.2.1 *A Nash equilibrium is a tuple* $(\boldsymbol{\ell}^*, \boldsymbol{h}^*) = (\ell^{1,*}, h^{1,*}, \ldots, \ell^{N,*}, h^{N,*}) \in \mathbb{A}^N$ *such that*

$$\forall n \in \mathcal{N}, \text{ and } (\ell^n, h^n) \in \mathbb{A}, \quad J^n(\boldsymbol{\ell}^*, \boldsymbol{h}^*) \leq J^n((\boldsymbol{\ell}^{-n,*}, \ell^n), (\boldsymbol{h}^{-n,*}, h^n)), \tag{4.6}$$

where $\boldsymbol{\ell}^{-n,*}$ *represents strategies of players other than the* n*-th one:*

$$\boldsymbol{\ell}^{-n,*} := [\ell^{1,*}, \ldots, \ell^{n-1,*}, \ell^{n+1,*}, \ldots, \ell^{N,*}] \in \mathbb{A}^{N-1}, \tag{4.7}$$

$\mathbb{A}$ *denotes the set of admissible strategies for each player and* $\mathbb{A}^N$ *is the produce of* N *copies of* $\mathbb{A}$*. For simplicity, we have assumed all players taking actions in the same space.*

In the sequel, to fix the notations, we shall use

- a regular character with a superscript n for an object from player n;
- a boldface character for a collection of objects from all players, *i.e.*, $\boldsymbol{S}_t \equiv [S_t^1, \ldots, S_t^N]^{\mathrm{T}}$;
- a boldface character with a superscript $-n$ for a collection of objects from all players except n, *i.e.*, $\boldsymbol{S}_t^{-n} \equiv [S_t^1, \ldots, S_t^{n-1}, S_t^{n+1}, \ldots, S_t^N]^{\mathrm{T}}$.

A Markovian Nash equilibrium is a Nash equilibrium defined above with $\mathbb{A}$ being the set of Borel measurable functions: $(\ell, h) : [0, T] \times \mathbb{R}^{3N} \to [0, 1]^2$. In other words,

the policies (ℓ_t^n, h_t^n) at time t are functions of the time t and the current values of all players' state processes $(\boldsymbol{S}_t, \boldsymbol{E}_t, \boldsymbol{I}_t)$. We omit the dependence on $\boldsymbol{R}_t$ as it is redundant as a consequence of the conservation law.

We derive below the Hamilton-Jacobi-Bellman (HJB) equations characterizing the Markovian Nash equilibrium. To simplify the notation, we first rewrite the dynamics of $(\boldsymbol{S}_t, \boldsymbol{E}_t, \boldsymbol{I}_t)$ defined in (4.1)–(4.3) into a vector form

$$\boldsymbol{X}_t \equiv [\boldsymbol{S}_t, \boldsymbol{E}_t, \boldsymbol{I}_t]^{\mathrm{T}} \equiv [S_t^1, \cdots, S_t^N, E_t^1, \cdots, E_t^N, I_t^1, \cdots, I_t^N]^{\mathrm{T}} \in \mathbb{R}^{3N}.$$

Again, we shall drop the redundant process $\boldsymbol{R}_t$. The dynamics of $\boldsymbol{X}_t$ reads:

$$\mathrm{d}\boldsymbol{X}_t = b(t, \boldsymbol{X}_t, \boldsymbol{\ell}(t, \boldsymbol{X}_t), \boldsymbol{h}(t, \boldsymbol{X}_t))\,\mathrm{d}t + \Sigma(\boldsymbol{X}_t)\,\mathrm{d}\boldsymbol{W}_t, \tag{4.8}$$

where b, Σ are deterministic functions in $\mathbb{R}^{3N}$ and $\mathbb{R}^{3N\times 2N}$, and $\{\boldsymbol{W}_t\}_{0\le t\le T}$ is a $2N$-dimensional standard Brownian motion. Each player n aims to minimize the expected running cost

$$\mathbb{E}\left[\int_0^T f^n(t, \boldsymbol{X}_t, \ell^n(t, \boldsymbol{X}_t), h^n(t, \boldsymbol{X}_t))\,\mathrm{d}t\right]. \tag{4.9}$$

We defer the specific definitions of $b, \Sigma, \boldsymbol{W}$, and f^n to Appendix B.1.1 to facilitate the exposition. We now define the value function of player n by

$$V^n(t, \boldsymbol{x}) = \inf_{(\ell^n, h^n)\in\mathbb{A}} \mathbb{E}\left[\int_t^T f^n(s, \boldsymbol{X}_s, \ell^n(s, \boldsymbol{X}_s), h^n(s, \boldsymbol{X}_s))\,\mathrm{d}s | \boldsymbol{X}_t = \boldsymbol{x}\right]. \tag{4.10}$$

By dynamic programming, it solves the following HJB system

$$\begin{cases} \partial_t V^n + \inf_{(\ell^n,h^n)\in[0,1]^2} H^n(t,\boldsymbol{x},(\boldsymbol{\ell},\boldsymbol{h})(t,\boldsymbol{x}),\nabla_{\boldsymbol{x}}V^n) + \frac{1}{2}\mathrm{Tr}(\Sigma(\boldsymbol{x})^{\mathrm{T}}\mathrm{Hess}_{\boldsymbol{x}}V^n\Sigma(\boldsymbol{x})) = 0, \\ V^n(T,\boldsymbol{x}) = 0, \quad n\in\mathcal{N}, \end{cases} \tag{4.11}$$

where H^n is the usual Hamiltonian defined by

$$H^n(t,\boldsymbol{x},\boldsymbol{\ell},\boldsymbol{h},\boldsymbol{p}) = b(t,\boldsymbol{x},\boldsymbol{\ell},\boldsymbol{h})\cdot\boldsymbol{p} + f^n(t,\boldsymbol{x},\ell^n,h^n), \tag{4.12}$$

∂_t denotes the time derivative, $\nabla_{\boldsymbol{x}}V$ and $\mathrm{Hess}_{\boldsymbol{x}}V$ denote the gradient and the Hessian of the function V with respect to $\boldsymbol{x}$, respectively, and Tr stands for the trace of a matrix.

Finding the optimal policies for N regions is equivalent to solving N-coupled $3N+1$ dimensional nonlinear equations (4.11). For example, when $N=3$, each PDE is 10-dimensional and conventional methods start to lose their efficiency. The recently proposed deep learning algorithm in [31], known as deep fictitious play (DFP), has shown excellent numerical performance in solving high-dimensional, coupled HJB equations with convergence analysis [32]. In the next section, we will first review and then propose an enhanced version of DFP to tackle some new issues.

4.3 Numerical methodology: Enhanced deep fictitious algorithm

We first briefly review the deep fictitious play (DFP) algorithm for solving equation (4.11) and we refer readers to [31] for full details. With the idea of fictitious play, DFP recasts the N-player game into N decoupled optimization problems, which are solved repeatedly stage by stage. Each individual problem is solved by the deep BSDE method

[37, 38]. The algorithm starts with some initial guess $(\boldsymbol{\ell}^0, \boldsymbol{h}^0)$, where the superscript 0 stands for stage 0. At the $(m+1)^{th}$ stage, given the optimal policies $(\boldsymbol{\ell}^m, \boldsymbol{h}^m)$ at the previous stage, the algorithm solves the following PDEs

$$\begin{cases} \partial_t V^{n,m+1} + \inf_{(\ell^n,h^n)\in[0,1]^2} H^n(t,\boldsymbol{x},(\ell^n,\boldsymbol{\ell}^{-n,m},h^n,\boldsymbol{h}^{-n,m})(t,\boldsymbol{x}),\nabla_{\boldsymbol{x}}V^{n,m+1}) \\ \qquad\qquad +\frac{1}{2}\mathrm{Tr}(\Sigma(\boldsymbol{x})^{\mathrm{T}}\mathrm{Hess}_{\boldsymbol{x}}V^{n,m+1}\Sigma(\boldsymbol{x})) = 0, \\ V^{n,m+1}(T,\boldsymbol{x}) = 0, \quad n\in\mathcal{N}, \end{cases} \tag{4.13}$$

and obtains the $(m+1)^{th}$ stage's optimal strategy by:

$$(\ell^{n,m+1},h^{n,m+1})(t,\boldsymbol{x}) = \underset{(\ell^n,h^n)\in[0,1]^2}{\arg\min}\ H^n(t,\boldsymbol{x},(\ell^n,\boldsymbol{\ell}^{-n,m},h^n,\boldsymbol{h}^{-n,m})(t,\boldsymbol{x}),\nabla_{\boldsymbol{x}}V^{n,m+1}(t,\boldsymbol{x})). \tag{4.14}$$

Here, $(\boldsymbol{\ell}^{-n,m},\boldsymbol{h}^{-n,m})$ stands for others' optimal policies from the m^{th} stage and are considered to be fixed functions when solving the PDE at the current stage. In the sequel, to simplify notations we omit the stage label m in the superscript when there is no risk of confusion. To solve (4.13) at each stage, it is first rewritten in the DFP as

$$\partial_t V^n + \frac{1}{2}\mathrm{Tr}(\Sigma(\boldsymbol{x})^{\mathrm{T}}\mathrm{Hess}_{\boldsymbol{x}}V^n\Sigma(\boldsymbol{x})) + \mu^n(t,\boldsymbol{x};\boldsymbol{\ell}^{-n},\boldsymbol{h}^{-n})\cdot\nabla_{\boldsymbol{x}}V^n \tag{4.15}$$

$$+ g^n(t,\boldsymbol{x},\Sigma(\boldsymbol{x})^{\mathrm{T}}\nabla_{\boldsymbol{x}}V^n;\boldsymbol{\ell}^{-n},\boldsymbol{h}^{-n}) = 0, \tag{4.16}$$

with some functions μ^n and g^n. The solution is then approximated by solving the equivalent BSDE $(\boldsymbol{X}_t^n, Y_t^n, Z_t^n)\in\mathbb{R}^{3N}\times\mathbb{R}\times\mathbb{R}^{2N}$:

$$\begin{cases} \boldsymbol{X}_t^n = \boldsymbol{x}_0 + \int_0^t \mu^n(s,\boldsymbol{X}_s^n;(\boldsymbol{\ell}^{-n},\boldsymbol{h}^{-n})(s,\boldsymbol{X}_s^n))\,\mathrm{d}s + \int_0^t \Sigma(\boldsymbol{X}_s^n)\,\mathrm{d}\boldsymbol{W}_s, & (4.17) \\ Y_t^n = \int_t^T g^n(s,\boldsymbol{X}_s^n,Z_s^n;(\boldsymbol{\ell}^{-n},\boldsymbol{h}^{-n})(s,\boldsymbol{X}_s^n))\,\mathrm{d}s - \int_t^T (Z_s^n)^{\mathrm{T}}\,\mathrm{d}\boldsymbol{W}_s, & (4.18) \end{cases}$$

in the sense of (*cf.* [39, 40, 41])

$$Y_t^n = V^n(t, \boldsymbol{X}_t^n) \quad \text{and} \quad Z_t^n = \Sigma(\boldsymbol{X}_t^n)^{\mathrm{T}} \nabla_{\boldsymbol{x}} V^n(t, \boldsymbol{X}_t^n). \tag{4.19}$$

The high-dimensional BSDE (4.17)–(4.18) is tackled by the deep BSDE method proposed in [37, 38].

In [31], the algorithm solves the BSDE by parametrizing $V^n(t, \boldsymbol{x})$ using neural networks (NN) and then obtains the approximate optimal policy by plugging the NN outputs into (4.14). For memory efficiency, the algorithm only stores the NNs' parameters at the current and the one-step previous stages. This strategy works well for games like the linear-quadratic game, but it would be ineffective if $\boldsymbol{\ell}^{-n}$ or $\boldsymbol{h}^{-n}$ explicitly appears in the minimizer in (4.14). In this case, when evaluating others' strategy $\boldsymbol{\ell}^{-n}$ or $\boldsymbol{h}^{-n}$ at stage m, it does not only need NNs at stage m but also at stages $m-1$, $m-2$, ..., 0. This means one needs to store NNs' parameters for all the previous stages from $1, \ldots, m$, and evaluate the associated output. Therefore, the time complexity of evaluating $(\boldsymbol{\ell}^{-n}, \boldsymbol{h}^{-n})(s, \boldsymbol{X}_s^n)$ up to stage m is $\mathcal{O}(m^2)$ and the memory complexity is $\mathcal{O}(m)$. This is infeasible in practice, as for real problems it hundreds of stages are needed. To overcome this significant problem, we propose an enhanced version of the original algorithm which reduces the time complexity to $\mathcal{O}(m)$ and the memory complexity to $\mathcal{O}(1)$. We present this new, enhanced algorithm (with pseudocode).

4.3.1 Algorithm

In order to reduce the computational complexity of evaluating $(\boldsymbol{\ell}^{-n}, \boldsymbol{h}^{-n})(s, \boldsymbol{X}_s^n)$ in the situation when $\boldsymbol{\ell}^{-n}$ or $\boldsymbol{h}^{-n}$ explicitly appears in the minimizer in (4.14), we propose the *Enhanced Deep Fictitious Play* which parametrizes both $V^n(t, \boldsymbol{x})$ and policy $(\ell^n, h^n)(t, \boldsymbol{x})$ by NNs. For simplicity, we state the algorithm based on a generic stochas-

tic differential game, where (possibly high-dimensional) controls are denoted by $\alpha^n(t, \boldsymbol{x})$ for player n.

In each stage of the *Enhanced Deep Fictitious Play*, for each planner n, the loss that our algorithm aims to minimize consists of two parts: (1) the loss related to solving (4.17)–(4.18) and (2) the error of approximating the optimal strategy α^n within some hypothesis spaces. The resulted approximation $\tilde{\alpha}^n$ will be used in the next stage of fictitious play:

$$\begin{aligned}
&\inf_{Y_0^n, \tilde{\alpha}^n, \{Z_t^n\}_{0\le t\le T}} \mathbb{E}(|Y_T^n|^2 + \tau \int_0^T \|\alpha^n(s, \boldsymbol{X}_s^n) - \tilde{\alpha}^n(s, \boldsymbol{X}_s^n)\|_2^2 \, \mathrm{d}s) \\
\text{s.t. } &\boldsymbol{X}_t^n = \boldsymbol{x}_0 + \int_0^t \mu^n(s, \boldsymbol{X}_s^n; \tilde{\boldsymbol{\alpha}}^{-n}(s, \boldsymbol{X}_s^n)) \, \mathrm{d}s + \int_0^t \Sigma(\boldsymbol{X}_s^n) \, \mathrm{d}\boldsymbol{W}_s, \\
&Y_t^n = Y_0^n - \int_0^t g^n(s, \boldsymbol{X}_s^n, Z_s^n; \tilde{\boldsymbol{\alpha}}^{-n}(s, \boldsymbol{X}_s^n)) \, \mathrm{d}s + \int_0^t (Z_s^n)^{\mathrm{T}} \, \mathrm{d}\boldsymbol{W}_s, \\
&\alpha^n(s, \boldsymbol{X}_s^n) = \arg\min_{\beta^n} H^n(s, \boldsymbol{X}_s^n, (\beta^n, \tilde{\boldsymbol{\alpha}}^{-n})(s, \boldsymbol{X}_s^n), Z_s^n),^2
\end{aligned} \tag{4.20}$$

where $\|\cdot\|_2$ denotes the 2-norm, $\tilde{\boldsymbol{\alpha}}^{-n}$ denotes the collection of approximated optimal controls from the previous stage except player n, and τ is a hyperparameter denoting the weight between two terms in the loss function. As detailed in Section 4.3.2, the hypothesis space for which we search $\tilde{\alpha}^n$ is characterized by another NN, in addition to the one to approximate Y_0 and $\{Z_t^n\}_{0\le t\le T}$. Although representing $\tilde{\alpha}^n$ with a neural network introduces approximation errors, it allows us to efficiently access the proxy of the optimal strategy $\boldsymbol{\alpha}^{-n}$ in the last stage by calling corresponding networks, instead of storing and calling all the previous strategies $\boldsymbol{\alpha}^{-n,m-1}, \ldots, \boldsymbol{\alpha}^{-n,1}$ due to the recursive dependence.

Numerically we solve a discretized version of (4.20). Given a partition π of size N_T on the time interval $[0, T]$, $0 <= t_0 < t_1 < .... < t_{N_T} = T$, the algorithm reads (to ease

[2]Here we have assumed that the Hamiltonian H^n depends on $\nabla_{\boldsymbol{x}} V$ through $\Sigma^{\mathrm{T}} \nabla_{\boldsymbol{x}} V$.

the notation, we replace the subscript t_k by k):

$$\inf_{\psi_0 \in \mathcal{N}_0^{n'}, \left\{\phi_k \in \mathcal{N}_k^n, \xi_k \in \mathcal{N}_k^{n''}\right\}_{k=0}^{N_T-1}} \mathbb{E}\{|Y_T^{n,\pi}|^2 + \tau \sum_k \|\alpha_k^{n,\pi} - \tilde{\alpha}_k^{n,\pi}(\boldsymbol{X}_k^{n,\pi})\|_2^2 \, \Delta t_k\} \tag{4.21}$$

$$\text{s.t. } \boldsymbol{X}_0^{n,\pi} = \boldsymbol{X}_0, \quad Y_0^{n,\pi} = \psi_0\left(\boldsymbol{X}_0^{n,\pi}\right), \quad Z_k^{n,\pi} = \phi_k\left(\boldsymbol{X}_k^{n,\pi}\right), \quad \tilde{\alpha}_k^{n,\pi}(\boldsymbol{X}_k^{n,\pi}) = \xi_k(\boldsymbol{X}_k^{n,\pi}), \tag{4.22}$$

$$\alpha_k^{n,\pi} = \arg\min_{\beta^n} H^n(t_k, \boldsymbol{X}_k^{n,\pi}, (\beta^n, \tilde{\boldsymbol{\alpha}}_k^{-n,\pi})(\boldsymbol{X}_k^{n,\pi}), Z_k^{n,\pi}), \quad k = 0, \ldots, N_T - 1 \tag{4.23}$$

$$\boldsymbol{X}_{k+1}^{n,\pi} = \boldsymbol{X}_k^{n,\pi} + \mu^n\left(t_k, \boldsymbol{X}_k^{n,\pi}; \tilde{\boldsymbol{\alpha}}_k^{-n,\pi}(\boldsymbol{X}_k^{n,\pi})\right) \Delta t_k + \Sigma\left(t_k, \boldsymbol{X}_k^{n,\pi}\right) \Delta \boldsymbol{W}_k, \tag{4.24}$$

$$Y_{k+1}^{n,\pi} = Y_k^{n,\pi} - g^n\left(t_k, \boldsymbol{X}_k^{n,\pi}, Z_k^{n,\pi}; \tilde{\boldsymbol{\alpha}}_k^{-n,\pi}(\boldsymbol{X}_k^{n,\pi})\right) \Delta t_k + (Z_k^{n,\pi})^{\mathrm{T}} \Delta \boldsymbol{W}_k, \tag{4.25}$$

where $\Delta t_k = t_{k+1} - t_k$, $\Delta \boldsymbol{W}_k = \boldsymbol{W}_{t_{k+1}} - \boldsymbol{W}_{t_k}$, and $\mathcal{N}_0^{n'}$, $\{\mathcal{N}_k^n\}_{k=0}^{N_T-1}$, $\{\mathcal{N}_k^{n''}\}_{k=0}^{N_T-1}$ are hypothesis spaces for player n, which will be specified later through neural network structures.

The expectation in (4.21) is further approximated by Monte Carlo samples of (4.24)-(4.25). The parameters in the hypothesis spaces are determined by stochastic gradient descent (SGD) algorithms such that the approximated expectation is minimized, which in turn gives the optimal deterministic functions $(\psi_0^*, \phi_k^*, \xi_k^*)$. We expect that $(\psi_0^*, \phi_k^*, \xi_k^*)$ will approximate $(V^n, \nabla_{\boldsymbol{x}} V^n, \alpha^n)$ well when this proxy of (4.21) is small. Particularly, $\{\xi_k^*\}_{k=0}^{N_T-1}$ serves as an efficient tool to evaluate the optimal policy at the current stage for finding Nash equilibrium. Implementation details and the full algorithm are presented in Section 4.3.2. Note that when $\tau = 0$ and $\tilde{\alpha}$ are replaced by α in the above algorithm, the *Enhanced Deep Fictitious Play* degenerates to the *Deep Fictitious Play* proposed in [31].

4.3.2 Implementation

Here we provide some detail to implement the methodology in Section 4.3.1. First, we specify the hypothesis spaces for neural networks $\mathcal{N}_0^{n'}$,$\{\mathcal{N}_k^n\}_{k=0}^{N_T-1}$, $\{\mathcal{N}_k^{n''}\}_{k=0}^{N_T-1}$, corresponding to $V^n, \nabla_{\boldsymbol{x}} V^n, \alpha^n$ (the superscript m is dropped again for simplicity). $V^n(t,x)$ is parametrized directly by a neural network $\mathrm{NN}(t,\boldsymbol{x})$. Corresponding map $\Sigma(\boldsymbol{X})^{\mathrm{T}}\nabla_{\boldsymbol{x}} V^n(t,\boldsymbol{X})$ that defines Z_t^n in the optimization problem (4.20) could be parametrized by $\Sigma(\boldsymbol{x})\nabla_{\boldsymbol{x}}\mathrm{NN}(t,\boldsymbol{x})$. Naturally, $\Sigma(\boldsymbol{x})\nabla_{\boldsymbol{x}}\mathrm{NN}(t_k,\boldsymbol{x})$ is a hypothesis function in $\mathcal{N}_k^n$. Under this parametrization rule, the hypothesis functions in $\mathcal{N}_0^{n'}$ and $\{\mathcal{N}_k^n\}_{k=0}^{N_T-1}$ share the same set of parameters. The policy function $\alpha^n(t,\boldsymbol{x})$ is parametrized by another neural network $\widetilde{\mathrm{NN}}(t,\boldsymbol{x})$ and then $\widetilde{\mathrm{NN}}(t_k,\boldsymbol{x})$ plays the role of a hypothesis function in $\mathcal{N}_k^{n''}$. In other words, $\{\mathcal{N}_k^{n''}\}_{k=0}^{N_T-1}$ share the same set of neural networks. In a stochastic game of N players, there are $2N$ neural networks in total, with N neural networks corresponding to $V^n(t,\boldsymbol{x})$ and N neural networks corresponding to $\alpha^n(t,\boldsymbol{x})$. At stage m, the N V-networks are trained to approximate the solution of PDE (4.15) and the N α-networks are trained to approximate the current optimal policy computed by (4.14) using the optimal strategies in the last stage. The updated neural networks at stage m would be used at stage $m+1$ to simulate paths $\{X_k^{n,\pi}\}_{k=0}^{N_T-1}$ and optimal strategies by (4.14). In this work, fully connected neural networks with three hidden layers are used.

Second, at each stage, the $2N$ neural networks could be decoupled to N pairs of V-network and α-network based on players. Then, the N pairs of neural networks could be trained in parallel, which dramatically reduces computational time. As [31] and [42] pointed out, it is not necessary to solve the individual control problem accurately in each stage; the parameters at each stage are updated starting from the optimal parameters in the last stage without re-initialization. This requires only a moderate number of epochs for the stochastic gradient descent at each stage.

The full implementation of *Enhanced Deep Fictitious Play* is shown in Algorithm 2. For simplicity, we state the algorithm based on a generic stochastic differential game.

Algorithm 2: Enhanced Deep Fictitious Play for Finding Markovian Nash Equilibrium

Require: $N = \#$ of players, $N_T = \#$ of subintervals on $[0, T]$, $M = \#$ of total stages in fictitious play, $N_{\text{sample}} = \#$ of sample paths generated for each player at each stage of fictitious play, $N_{\text{SGD\_per\_stage}} = \#$ of SGD steps for each player at each stage, $N_{\text{batch}} =$ batch size per SGD update, $\boldsymbol{\alpha}^0$: the initial policies that are smooth enough
Initialize N deep neural networks to represent $V^{n,0}$ and N deep neural networks to represent $\alpha^{n,0}, n \in \mathcal{N}$
for $m \leftarrow 1$ *to* M **do**
  for ***all*** $n \in \mathcal{N}$ ***parallelly*** **do**
    Generate N_{sample} sample paths $\{\boldsymbol{X}_k^{n,\pi}\}_{k=0}^{N_T}$ according to (4.24) and the realized approximate optimal policies $\tilde{\boldsymbol{\alpha}}^{-n,m-1}(t_k, \boldsymbol{X}_k^{n,\pi})$ (Remark: $\tilde{\boldsymbol{\alpha}}$ represents social and health policies in the multi-region SEIR model)
    for $e \leftarrow 1$ *to* $N_{SGD\_per\_stage}$ **do**
      Update the parameters of the n^{th} V-neural network and α-neural network one step with N_{batch} paths using the SGD algorithm (or its variant), based on the loss function (4.21)
    Obtain the approximate optimal policy $\tilde{\alpha}^{n,m}$ represented by the latest policy neural network
  Collect the approximate optimal policies at stage m: $\tilde{\boldsymbol{\alpha}}^m \leftarrow (\tilde{\alpha}^{1,m}, \ldots, \tilde{\alpha}^{N,m})$
return The approximate optimal policy $\tilde{\boldsymbol{\alpha}}^M$

Due to page limits, the exact choice of NN architectures will be detailed in Appendix B.2.1. To determine the total stages of fictitious play M, we monitor the relative changes of α^n and V^n, and stop the process when the relative change from stage to stage is below a threshold. Regarding the total number of SGD per stage, as shown in [31, Figure 1], the original DFP is insensitive to the choice of $N_{\text{SGD\_per\_stage}}$. We find the enhanced version sharing the same behavior when apply to the COVID-19 case study. We give more details in Section 4.4.2, and further experiments regarding different choices of M and $N_{\text{SGD\_per\_stage}}$ in Appendix B.2.2.

For problems without analytical solutions, one natural concern is the reliability of

numerical solutions. Theoretically, the quantity (4.21) serves as the indicator of the numerical accuracy. In the original DFP where the second term in (4.21) does not exist, Theorem 3 in [33] ensures the convergence to the true Nash equilibrium under technical assumptions when (4.21) is small enough for each fictitious play stage and with sufficiently large M and small Δt_k. In practice, the quantity in (4.21) is approximated by its Monte Carlo counterpart, which we define as the loss function of our algorithms. Therefore, having small training losses during all stages will ensure convergence. Extending Theorem 3 in [33] to the current setting is beyond the scope of this paper and is left for further work.

4.4 Application on COVID-19

Our case study is based on COVID-19. We focus mainly on the lockdown/travel ban policy between different regions. Therefore, to simplify the presentation, we omit the health policy h in the following discussion and make $v(\cdot) = v$, $\lambda(\cdot) = \lambda$, and $\eta = 0$. Moreover, as vaccines are not available to the population yet, we let $v(\cdot) = v = 0$.

4.4.1 Parameter choices

In single-region SEIR models, the transmission rate, β, is the basic reproductive number divided by the length of time an individual is infectious. In our model, we assume that there is a region-independent constant β that underlies the rate of infections for each population. The transmission rates β^{nk} between regions are related to the underlying transmission rate β, and the amount of travel between regions n and k.

To quantify the size of travel between regions, we assume there is a constant fraction of people from region n that travel to region k, f^{nk}, at any given moment in time. We note that realistically one may expect f^{nk} to depend on time and also on the epidemic

status of regions n and k. However, for simplicity, we will not consider these scenarios in our numerical experiments. We assume that $f^{nn} \gg f^{nk}$, $f^{nn} \gg f^{kn}$, $\forall k \neq n$, meaning that most of the population n resides in region n at any given time, and also that most of the people in region n at a given time are from n and not travelers from another region. We will see later that this implies $\beta^{nn} \gg \beta^{nk}$ $\forall k \neq n$.

To further clarify the transmission of infection from one region due to another, we need to describe the set of parameters $\{\beta^{nk} : n, k \in \mathcal{N}\}$. Here, β^{nk} represents the rate of transmission from region k to region n. Specifically, β^{nk} is the number of infected people in population n per a contactable, infectious individual in population k per day, assuming that 100% of population n is susceptible. This definition of β^{nk} comes from the derivation of the SDE system itself (4.1)–(4.3). Accounting for infection in region n by individuals in region k from travel to both region n and k, we have that

$$\beta^{nk} = \begin{cases} \beta(f^{nk}f^{kk} + f^{kn}f^{nn})\frac{P^k}{P^n}, & \text{if } k \neq n \\ \beta(f^{nn})^2, & \text{if } k = n. \end{cases} \tag{4.26}$$

We defer the detailed derivation of (4.26) to Appendix B.1.3.

Therefore, to specify β^{nk}, we need to provide β and f^{nk}. We will specify f^{nk} for New York (NY), New Jersey (NJ), and Pennsylvania (PA) in the next section. To estimate β, we choose a basic reproductive number $R_0 = 2.2$, which is consistent with [43], and assume that the length of each individual being infectious is 13 days. More precisely, we assume infectious individuals either recover or die in 13 days. Under these assumptions, we obtain $\beta = \frac{2.2}{13} \approx 0.17$, consistent with [44] as a 13 day median time until death from illness onset is used.

The infection fatality rate, or IFR, is the fraction of those infected who died from the infection. We choose the IFR to be 0.65% according to the CDC estimate. This is

also consistent with [45], which suggests a point estimate of 0.68%. The assumptions of an IFR of 0.65% and an infectious period of 13 days determines that the recovery rate (including both recovery and death due to infection) is $\lambda = \frac{1}{13} \approx 0.0769$, and the death rate is $\kappa = \frac{(0.65\%)}{13} = 0.0005$. We choose the latent period to be 5 days according to [46]. This means that the we will have $\gamma = \frac{1}{5}$. Note that this choice has also been used in other models such as [47] and [48]. We assume that the parameters for noise-level $\sigma_{s_n}, \sigma_{e_n}$, $n \in \mathcal{N}$ are all 0.0002, and the extent to which one adheres to the social distancing policy, θ, is either $\theta = 0.9$ or $\theta = 0.99$.

With most of the parameters for the SDE model (4.1)-(4.3) discussed, we now address those specific to defining the cost. Regarding the risk-free-rate r, note that U.S. Treasury yields are historically low and the uncertainty in the current level of inflation. We choose for simplicity that $r = 0$. Also, considering that we are interested in simulations with time periods of less than a year, the discounting is negligible. The parameter w represents the dollar output per individual per day. To estimate w, we use GDP per capita per day, yielding the estimate $w = 172.6$ dollars per person per day. Following [47] and [49], we use the value of a statistical life, χ, to be 20 times GDP per capita. This results in $\chi = 1.95 \cdot 10^6$ dollars per person. According to the CDC summary of U.S. COVID-19 activity, the hospitalization rate was 228.7 per 100,000 population by 11/14/2020. Thus, we set $p = 228.7 \times 10^{-5}$. The cost per inpatient day is $c = 73300/13$ dollars, estimated according to [50]. The attention hyperparameter a takes various values in the case study, and will be specified in Section 4.4.2.

4.4.2 NY-NJ-PA COVID-19 case study

In this section, we apply our model (4.1)–(4.5) to analyze COVID-19 related policy in three adjacent states: New York, New Jersey, and Pennsylvania. This case study is done

over 180 days starting from 03/15/2020, using the Enhanced Deep Fictitious Play algorithm introduced in Section 4.3 (cf. Algorithm 2) and the parameters discussed in 4.4.1. The exact formulas of μ^n and h^n in equation (4.15) are derived in Appendix B.1.2.

We refer to New York State as region 1, New Jersey State as region 2, and Pennsylvania State as region 3. Their respective populations are $P^1 = 19.54$ million, $P^2 = 8.91$ million, and $P^3 = 12.81$ million. Regarding β^{nk}, $\forall n, k = 1, 2, 3$, we assume that: (a) 90% of any state's population is residing in their state at a given time; (b) the remaining population (travelers) visit the other regions in an equal proportion; and (c) there is no travel outside of the considered regions, *i.e.*, the NY-NJ-PA is a closed system. The reasoning for (c) is that, under our model assuming that infection only occurs in the regions considered, (c) is equivalent to allowing people traveling outside the considered regions, but the travelers cannot be affected. For simplicity, we assume this is the case. Under these assumptions, we will have $f^{nn} = 90\%$ for $n = 1, 2, 3$ and $f^{nk} = 5\%$ for $n \neq k$, and obtain the values of β^{nk} through (4.26).

Figure 4.1 presents the equilibrium policy issued by the governors of NY, NJ, and PA when the policy effectiveness is $\theta = 0.99$, *i.e.*, 99% of the population follow the lockdown order. The hyperparameter is $a = 100$, *i.e.*, each governor values people's death 100 times the lockdown cost. In this scenario, the governors take action at an early stage and soon reach the strictest policy. Once the disease is under control, they may relax the policy later. The percentage of Susceptible, Exposed, Infectious, and Removed stays almost constant in the end. As a comparison, Figure 4.2 illustrates how the pandemic gets out of control if governors show inaction or issue mild lockdown policies.

Experiment 1: dependence on a. We further analyze how the planners' view on the death of human beings changes their policies. In reality, economic loss is not the only factor the planners concern about. It is also important to mitigate the infections and deaths within the budget and available resources. Different views and values from the

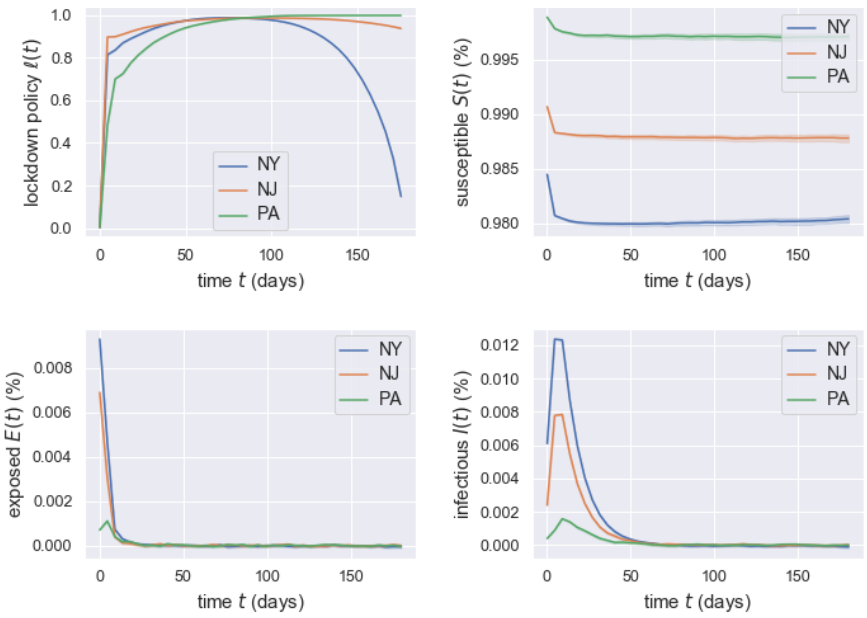


Figure 4.1: Plots of optimal policies (top-left), Susceptibles (top-right), Exposed (bottom-left) and Infectious (bottom-right) for three states: New York (blue), New Jersey (orange) and Pennsylvania (green). The shaded areas depict the mean and 95% confidence interval over 256 sample paths. Choices of parameters are in Section 4.4.1, $a = 100$ and $\theta = 0.99$.

planners will lead to different policies. In this experiment, we consider different attitudes towards the infection, especially death caused by COVID-19. This is reflected by the attention hyperparameter a. Large a implies that planners care more about human beings and are willing to spend more effort or endure more economic loss on lockdown to avoid further infection and death. In comparison, smaller a implies that planners care less about infection and death and instead pay more attention to minimizing the total cost.

The numerical results in Figures 4.3 and 4.4 are consistent with intuition. With a large a (top-left panels), meaning the planners give more consideration to infection and death, they tend to issue a restrict lockdown policy, which helps slow down the disease spreads and reduce the percentage of infected people. As a becomes smaller (top-right panels), planners weigh more the economic loss and spend fewer efforts on lockdown.

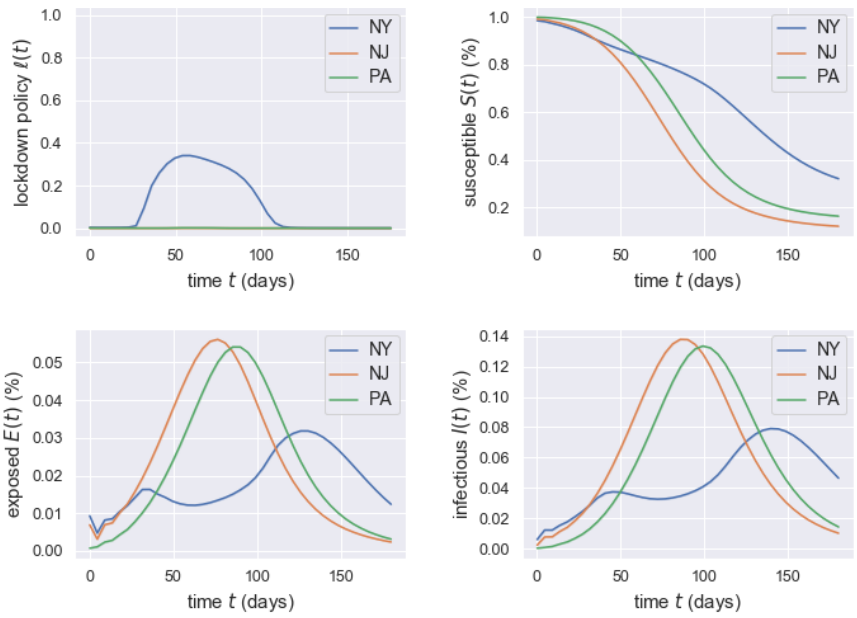


Figure 4.2: An illustration that governors' inaction or mild control leads to disease spreading.

When the attention a is small enough, some states even give up controlling the disease spread due to economic concern (bottom panels). As a result, the pandemic would get out of control by the end of the simulation period. This mild lockdown policy leads to a natural spread of disease (also shown in Figure 4.2).

Experiment 2: dependence on θ. We next analyze how the residents' willingness to comply with the lockdown policy changes the optimal policies and the development of a pandemic. The larger the θ is, the more likely the residents will follow the lockdown policy, and the larger the difference the control makes on the pandemic situation. Conversely, small θ weakens the effect of the lockdown policy. In the extreme case of $\theta = 0$, no matter how strict the lockdown policy is, the pandemic will become a natural spread because the control term in (4.1) disappears. In short, this willingness to policy compliance should be an essential factor in decision-making.

To this end, we compare the optimal policy when $\theta = 0.9$ and $\theta = 0.99$ in Figure

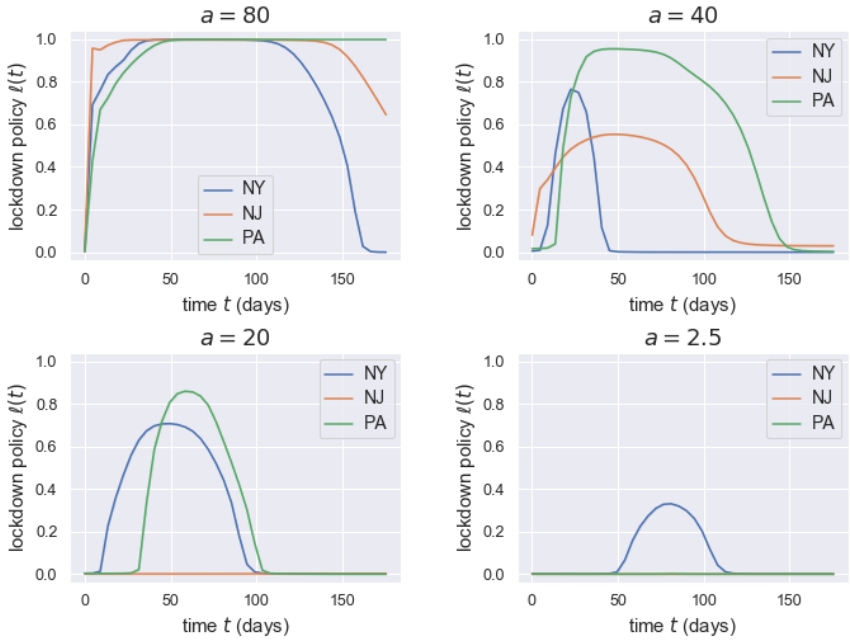


Figure 4.3: Plots of optimal policies with different choice of a for three states: New York (blue), New Jersey (orange) and Pennsylvania (green), when the lockdown efficiency is $\theta = 0.9$.

4.5. Panels (a-d) show the difference of optimal policies $\ell(t)$ and the Susceptible $S(t)$ in the tri-state game under different θ when $a = 50$. In both situations, the pandemic is well-controlled, with the percentage of susceptible people staying stable in the end. Moreover, in the case of $\theta = 0.99$, people are more willing to comply with the policies. Consequently, the planners are allowed to use a less strict lockdown policy as shown in Figure 4.5(b) compared to 4.5(a), which saves the lockdown cost. Figure 4.5 (e-h) shows an interesting case in the comparison of $\theta = 0.9$ and $\theta = 0.99$. In this scenario, with the same attention parameter ($a = 25$), $\theta = 0.9$ leads to a mild lockdown policy, see Figure 4.5(e), while $\theta = 0.99$ provides a possibility to stop the spread of virus, see Figure 4.5(f). We believe that the decision when $\theta = 0.9$ is a compromise as the lockdown is not efficient enough to reduce largely the infection and death loss by paying lockdown cost, and also due to the limited simulation period, *i.e.*, the policies could have been different

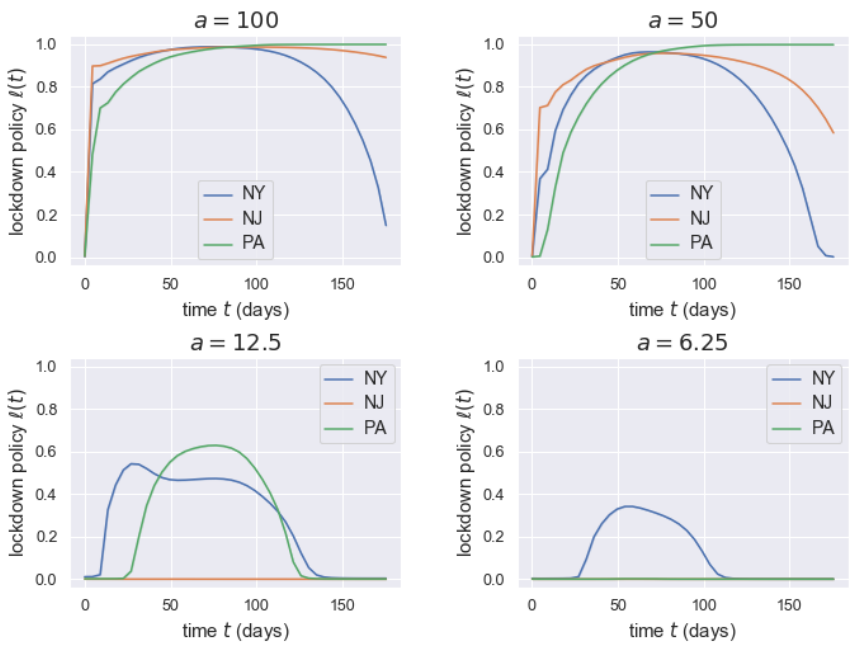


Figure 4.4: Plots of optimal policies with different choice of a for three states: New York (blue), New Jersey (orange) and Pennsylvania (green), when the lockdown efficiency is $\theta = 0.99$.

if we had the simulation until the disease dies out. We also believe that the early give-up by NJ drives NY and PA to lift lockdown policies at a later stage, because even NY and PA issue strict policies, they are still facing severe infections from NJ due to its high infected percentages and the existence of travel between states whatever the policy is. So their interventions are not worth the candle. Figure 4.5(f)(h) further elucidate the importance of residents' support in slowing down the pandemic. Further experiments based on different sets of (a, θ) reveal the possibility of having multiple Nash equilibrium, with more elaboration in Appendix B.2.3.

To summarize, the numerical experiments illustrate that both the balance of economy and infection/death from the view of plan-makers and the willingness of residents to follow the lockdown policy play an important role in decision-making. In reality, all three states issued stay-at-home orders in March, and attempted to reopen in June. By comparing

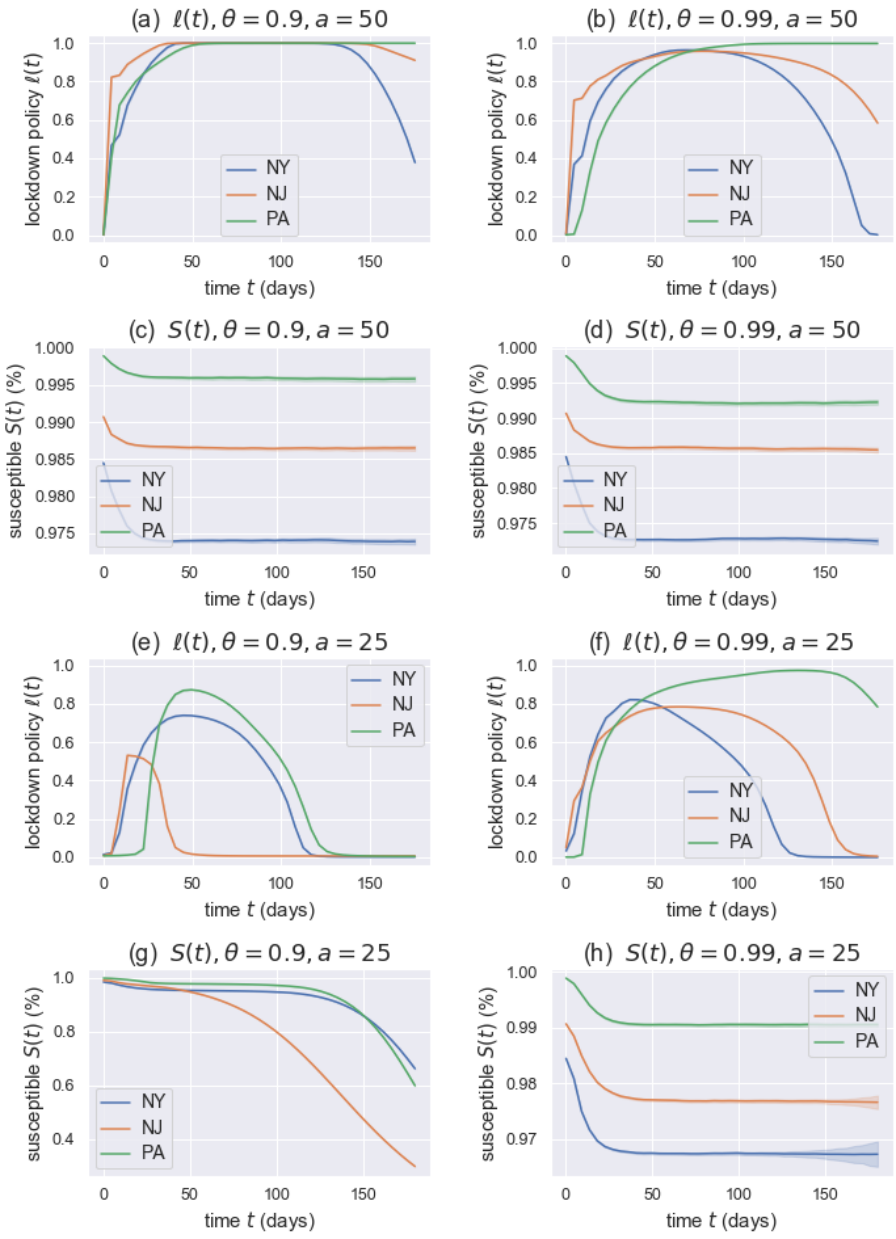


Figure 4.5: Comparison of optimal policies for three states (NY = blue, NJ = orange, PA = green) and their susceptibles between different policy effectiveness θ and hyperparameter a.

real world policies and our simulations of $\boldsymbol{\ell}(x)$, we may infer α and θ for NY, NJ, and PA in our model, *i.e.*, $\theta = 0.99$ and $a = 25$.

4.5 Conclusion

In this paper, we propose a novel multi-region SEIR model to study optimal policies under a pandemic. Our new model, built on game theory, takes into account how the social and health policies issued by multiple region planners affect the progress of infectious diseases. This feature makes the model more realistic and powerful but also introduces a formidable computational challenge due to the high-dimensionality of the solution space and the strong coupling of planners' policies. We propose the enhanced deep fictitious play algorithm to overcome the curse of dimensionality and use the model and algorithm in a case study of the COVID-19 pandemic in three states, New York, New Jersey, and Pennsylvania. The model parameters are estimated from real data posted by the CDC. We are able to show the effect of lockdown/travel ban policy on the spread of COVID-19 for each state and how people's willingness to comply and planners' attitude towards deaths influence the equilibrium strategies as a consequence of the competition between regions. We hope our model can draw more attention to studying optimal interventions in infectious diseases using game theory. Our numerical simulations can shed light on public policies.

In reality, during a pandemic, the planning is usually for short periods and adjusted frequently. This can be modeled using repeated games, and planners may infer other regions' cost functional from past game outcomes. The assumptions that some parameters are identical across different regions can be relaxed and the health policy can also be added for more accurate simulations. These will be left for future work.

Chapter 5

Conclusion

In this thesis, we explore two directions to leverage machine learning with scientific computing to accelerate computations or to solve previously intractable problems in computational mathematics. The first direction is to generate data by traditional numerical methods, and then train a machine learning system to learn the relationship between variables. The second direction is to use a machine learner as an approximation to the solution of a previously unsolvable system, plug the learner in the system, run numerical simulation of the system on the top of the learner, quantify the loss between the the theoretical value of the interested variable and the one derived from the learner after the simulation, then determine the parameters of the learner by minimizing the loss.

In the first direction, we focus on the case study of accelerating polymer phase discovery based on machine learning and self-consistent field theory. We solve both the one-dimensional and the two-dimensional problems. In the one-dimensional case, we generate data from SCFT simulations and train feedforward neural networks in Sobolev space to accurately approximate the effective Hamiltonian and its gradient. We also develop a technique, extendable to other architectures, to equip the neural network with translation invariance. The existence of a neural network in Sobolev space which pre-

serves shift invariance is rigorously proved. We then apply the gradient descent method on the Sobolev space-trained neural network to find saddle point density field predictions. In the two-dimensional case, we train convolutional neural networks (CNN's) in Sobolev space to obtain accurate approximations of the effective Hamiltonian and its gradient. We take advantage of the CNN architecture to equip it with strong rotation and translation invariance. We then develop generative adversarial neural networks (GAN's) to generate a saddle point density field pool, select the optimal one and refine it using the Sobolev space-trained CNN and gradient descent. The framework we propose for saddle point density field prediction could also be extended to a general inverse/optimization problem, where we can train a neural network for fast evaluation, generate initial prediction from the training set or by GAN, then screen and/or fine-tune the candidates to get the finalized prediction.

In the second direction, we take the case study to model the COVID-19 pandemic and determine the optimal policy based on stochastic game strategy. The stochastic game based on multi-region SEIR models introduced a high dimensional coupled HJB systems to solve for Nash Equilibrium. To solve these problems with no traditional solvers, we propose the enhanced deep fictitious algorithm, where we use neural networks to approximate the optimal control and value function, plug them in the systems to do numerical simulations on the corresponding Backward Stochastic Differential Equations, and search for the parameters by using stochastic gradient descent method to minimize the difference at terminal time. This provides the framework to solve previously unsolvable numerical system by machine learning methods.

There are still many other exciting directions to combine computational mathematics with the fast-developing machine learning techniques. For these two projects, polymer phase discovery and Nash equilibrium of stochastic game for a pandemic, there is still plenty of room for improvement. This work aims at providing a stepping stone to broad

and exciting new area of applied mathematics.

Appendix A

Appendix of Chapter 2

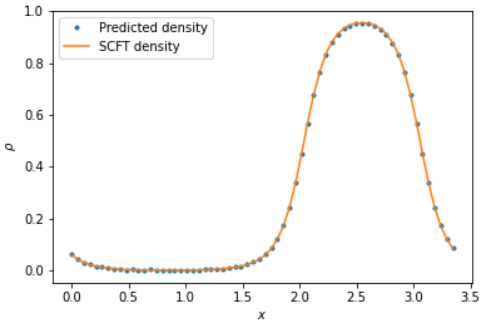
A.1 Learning H Versus Learning R Only

We present here numerical evidence that the proposed splitting of the Hamiltonian to focus on learning the entropic part R only produces superior results for the predicted saddle point density field than those produced by learning the full functional H. To make a fair comparison of the two strategies, we trained the two NN learners with the same architecture in the same set-up. Both methods produce comparable accuracy for H and its gradient and are both capable of enforcing shift invariance accurately but as Fig. A.1 shows, the splitting approach yields significantly more accurate density field predictions.

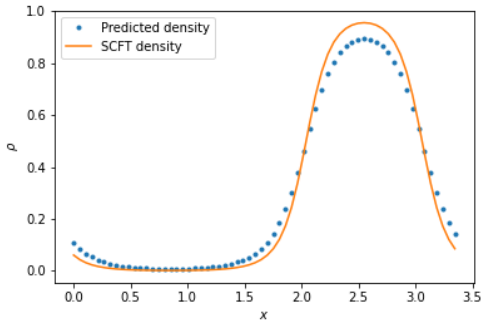
A.2 Universal approximation

Consider the Sobolev space $W_p^m(U) \equiv \{f \in L_{1,loc}(U) | \partial^\alpha f \in L_p(U,\lambda), 0 \leq |\alpha| \leq m\}$. There are two significant results on the existence of neural network approximation neural network in Sobolev space. These are the following theorems.

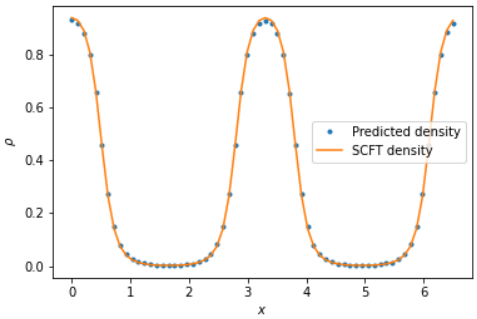
Theorem 5 *[11] If G is l-finite, $0 \leq m \leq l$, U is an open bounded subset of $\mathbb{R}^r$ and*



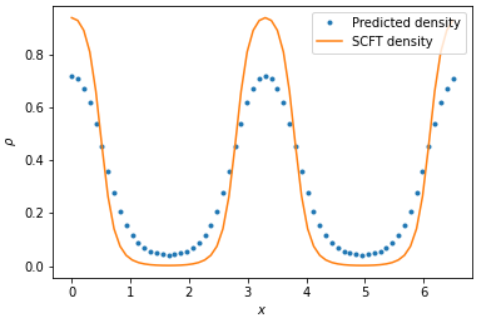
(a) $\chi N = 33.5$, $L = 3.4$, $f = 0.3$: learn the remainder



(b) $\chi N = 33.5$, $L = 3.4$, $f = 0.3$: learn the entire Hamiltonian



(c) $\chi N = 30.5$, $L = 6.6$, $f = 0.3$: learn the remainder



(d) $\chi N = 30.5$, $L = 6.6$, $f = 0.3$: learn the entire Hamiltonian

Figure A.1: Learning H versus learning R only. Predicted saddle point density fields prediction for the AB diblock copolymer with low-to-moderate χN: left panels (a and c) are the predictions based on learning the entropic remainder R only, right panels (b and d) are the predictions based on learning the entire Hamiltonian H.

$C_0^\infty(\mathbb{R}^r)$ is d_p^m-dense in $W_p^m(U)$ then $\Sigma(G)$ is also d_p^m-dense in $W_p^m(U)$.

For a natural number l, G is l-finite if $G \in C^l(\mathbb{R})$ and $0 < \int |D^l G| d\lambda < \infty$. Most commonly used activation functions are l-finite. In Theorem 2 d_p^m is the Sobolev norm up to m_{th} derivative induced, by L^p norm. The condition that $C_0^\infty(\mathbb{R}^r)$ is d_p^m-dense in $W_p^m(U)$ is easy to satisfy. As pointed out by Hornik et al. [11], for all U which is a bounded domain star-shaped with respect to a point O (equivalently, any ray with origin O has a unique intersection with the boundary of U) $C_0^\infty(\mathbb{R}^r)$ is d_p^m-dense in $W_p^m(U)$. This means that for most common subsets of $\mathbb{R}^r$, the aforementioned condition is naturally true. In [11],

the authors also proved several other versions of this universal approximation result for single layer neural networks with respect to the Sobolev norm. These theorems support the existence of such neural networks that simultaneously approximates a functional and its functional gradient.

Theorem 6 *[12] Given any two functions $f : S \to \mathbb{R}$ and $g : S \to \mathbb{R}^d$ and a finite set $U \subset S$, there exits neural network NN with a ReLU (or a leaky ReLU) activation such that $\forall x \in U : f(x) = NN(x)$ and $g(x) = \frac{\partial NN}{\partial x}(x)$.*

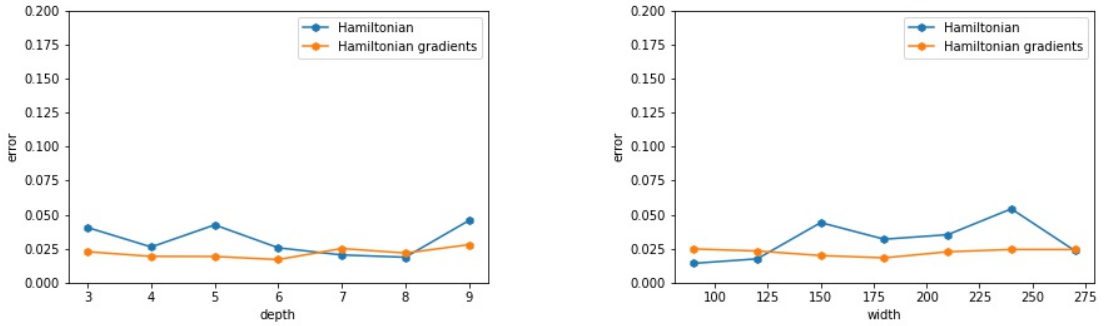
This theorem guarantees the existence of a Sobolev space-trained neural network with 0 training loss.

A.3 Ablation Study

In this section, we summarize results of a study that guided our choice of the network size. We consider the approximation error, in both H and its gradient, as the networks depth (number of hidden layers) and the network width (number of cells per layer) is changed while the training and test sets and all the other hyperparameters are kept fixed. We also examine the behavior of the training and validation loss functions as the number of epochs (iterations on the stochastic gradient descent method) increases.

In the first batch of experiments, all the hyperparameters are fixed except for the network depth. Seven neural networks with different number of hidden layers are trained on the same training set and evaluated on the same test set independently. In the second batch of experiments, the number of cells (network width) in the middle hidden layers are the only variable while all other hyperparameters are fixed. Again, seven neural networks with different widths are trained on *the same training set* and evaluated on the same test set. Figure A.2 shows that the approximation is relatively stable with respect

to the network size (both in depth and width) for the range considered. This suggests that for the relatively modest training set size there is no discernible improvement of the approximation as the network size grows beyond 6-8 levels. In other words, the training set is not big enough to benefit from the use of larger networks. This observation is consistent with the results reported by D'souza, Huang, and Yeh [51] on a convolutional neural network with a small sample size.



(a) error vs. number of hidden layers (b) error vs. number of cells in middle hidden layers

Figure A.2: Relationship between the error on test set (root square error of H and its gradient) and the size of neural network: (a) error versus number of hidden layers (depth), (b) error versus number of hidden cells (width) from hidden layer 2 to hidden layer 5.

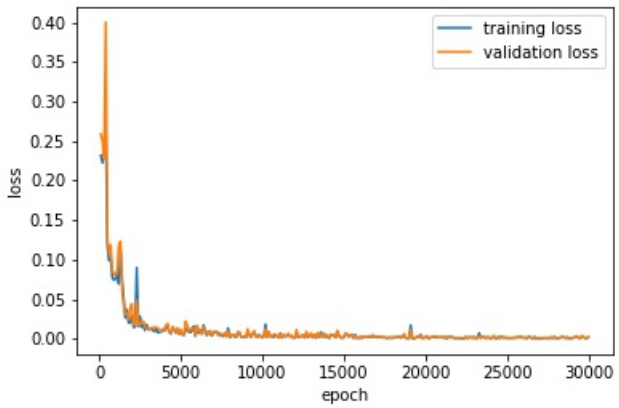


Figure A.3: Training and validation loss functions against the number of epochs (stochastic gradient descent iterations).

We now look at the behavior of the training and validation loss functions as number of epochs (iterations on the stochastic gradient descent method) increases. The training loss functions and the validation loss function (the sum of the loss of the effective Hamiltonian, the effective Hamiltonian gradient and the shift invariance term) after 100 epochs are presented in Fig. A.3. Both training loss function and the validation loss function (not used in training) decrease as expected as the number of epochs increases.

A.4 Hyperparameters Employed in the Numerical Experiments

We list in Table A.1 the hyperparameters we used for the experiments in Section 2.6. We took $\beta = 1$ to stress the equal priority of approximating both the Hamiltonian

χN	Structure	f	learning rate	λ	β	γ
low χN	AB diblock	0.3	0.0001	8.15e-08	1	$N_s/(N_s-1)$
		0.4	0.001	8.44e-09	1	$N_s/(N_s-1)$
	AB_3 star	0.4	0.0001	5.23e-08	1	$N_s/(N_s-1)$
high χN	AB diblock	0.3	0.001	4.97e-10	1	$N_s/(N_s-1)$
		0.4	0.001	2.80e-10	1	$N_s/(N_s-1)$
	AB_3 star	0.4	0.001	4.93e-11	1	$N_s/(N_s-1)$

Table A.1: Hyperparameters employed in the numerical experiments. N_s is the number of possible shifts.

and its gradient also to match the coefficient of the Sobolev norm. We chose $\gamma = N_s/(N_s-1)$, which is close to 1. This choice takes into account that one of the possible shifts, $s = N_s$, is a one-period shift and leads to a 0 difference with $NN(x_i)$ in formula (2.20) (only $N_s - 1$ valid terms in the summation). We employed the hyperparameters-tuning strategies described in Section 2.4.4 to select the optimal learning rate and the regularization coefficient λ. We took $\beta_V = \beta$ and $\gamma_V = \gamma$ in the validation loss (2.23) and $\theta_V = N_s$ to balance the summation. Note that one could also view β and γ as

hyperparameters and tune them as done for the learning rate and λ, which might generate better results. We did not do this because with the more expedited use of fixed values of β and γ the deep NN already produced impressive results.

In Step 2, when we search for saddle density fields by gradient descent, we performed 500 iterations after selecting the initial density fields from training set.

Appendix B

Appendix of Chapter 4

B.1 Technical Details to the Stochastic Multi-region SEIR Model

B.1.1 The dynamics of $\boldsymbol{X}_t$ in Section 4.2

In Section 4.2, for the ease of notations and clarity of the presentation, we rewrite the dynamics of (S_t^n, E_t^n, I_t^n) defined in (4.1)–(4.3) in the vector form

$$\mathrm{d}\boldsymbol{X}_t = b(t, \boldsymbol{X}_t, \boldsymbol{\ell}(t, \boldsymbol{X}_t), \boldsymbol{h}(t, \boldsymbol{X}_t))\,\mathrm{d}t + \Sigma(\boldsymbol{X}_t)\,\mathrm{d}\boldsymbol{W}_t, \tag{B.1}$$

where $\boldsymbol{X}_t \equiv [\boldsymbol{S}_t, \boldsymbol{E}_t, \boldsymbol{I}_t]^{\mathrm{T}} \equiv [S_t^1, \cdots, S_t^N, E_t^1, \cdots, E_t^N, I_t^1, \cdots, I_t^N]^{\mathrm{T}} \in \mathbb{R}^{3N}$, the Markovian controls $(\boldsymbol{\ell}, \boldsymbol{h})$ are given by $\boldsymbol{\ell}(t, \boldsymbol{x}) = [\ell^1, \ldots, \ell^N]^{\mathrm{T}}(t, \boldsymbol{x})$ and $\boldsymbol{h}(t, \boldsymbol{x}) = [h^1, \ldots, h^N]^{\mathrm{T}}(t, \boldsymbol{x})$. In the sequel, for a vector $\boldsymbol{x} \equiv (\boldsymbol{s}, \boldsymbol{e}, \boldsymbol{i}) \in \mathbb{R}^{3N}$, we shall index them in two ways,

$$(s_1, \cdots, s_N, e_1, \cdots, e_N, i_1, \cdots, i_N) \text{ or } (x_1, \ldots, x_{3N}). \tag{B.2}$$

and use them interchangeably. The former one emphasizes the dependence on each category, while the later notation is more condensed. Similarly, for partial derivatives, we will have two set of notations

$$(\partial_{s_1}, \cdots, \partial_{s_N}, \partial_{e_1}, \cdots, \partial_{e_N}, \partial_{i_1}, \cdots, \partial_{i_N}) \text{ or } (\partial_{x_1}, \cdots, \partial_{x_{3N}}). \tag{B.3}$$

We give precise definitions of (B.1) in this appendix. $b(t, \boldsymbol{x}, \boldsymbol{\ell}, \boldsymbol{h}) = [b_1, \ldots, b_{3N}]^{\mathrm{T}}(t, \boldsymbol{x}, \boldsymbol{\ell}, \boldsymbol{h})$ is a deterministic vector-valued function:

$$b_j(t, \boldsymbol{x}, \boldsymbol{\ell}, \boldsymbol{h}) = \begin{cases} -\sum_{k=1}^{N} \beta^{jk} s_j i_k (1 - \theta \ell^j(t, \boldsymbol{x}))(1 - \theta \ell^k(t, \boldsymbol{x})) - v(h^j(t, \boldsymbol{x})) s_j, & j \in \mathcal{N}, \\ \sum_{k=1}^{N} \beta^{j'k} s_{j'} i_k (1 - \theta \ell^{j'}(t, \boldsymbol{x}))(1 - \theta \ell^k(t, \boldsymbol{x})) - \gamma e_{j'}, & j \in \mathcal{N} + N, \\ \gamma e_{j'} - \lambda(h^{j'}(t, \boldsymbol{x})) i_{j'}, & j \in \mathcal{N} + 2N, \text{ and } j' = j \bmod N. \end{cases} \tag{B.4}$$

$\Sigma(\boldsymbol{x}) = (\Sigma_{j,k}(\boldsymbol{x}))$ is a matrix-valued deterministic function in $\mathbb{R}^{3N \times 2N}$ with non-zero entries given below:

$$\Sigma_{j,j}(\boldsymbol{x}) = -\sigma_{s_j} s_j, \qquad \Sigma_{j+N,j}(\boldsymbol{x}) = \sigma_{s_j} s_j, \tag{B.5}$$

$$\Sigma_{j+N,j+N}(\boldsymbol{x}) = -\sigma_{e_j} e_j, \qquad \Sigma_{j+2N,j+N}(\boldsymbol{x}) = \sigma_{e_j} e_j, \quad j \in \mathcal{N}. \tag{B.6}$$

and $\{\boldsymbol{W}_t\}_{0 \le t \le T}$ is a $2N$-dimensional standard Brownian motion:

$$\boldsymbol{W}_t = [W_t^{s_1}, \cdots, W_t^{s_N}, W_t^{e_1}, \cdots, W_t^{e_N}]^{\mathrm{T}}.$$

According to (4.5), each region's running cost f^n defined in (4.9) is

$$f^n(t, \boldsymbol{x}, \boldsymbol{\ell}, \boldsymbol{h}) = e^{-rt} P^n[(s_n + e_n + i_n)\ell^n(t, \boldsymbol{x})w + a(\kappa i_n \chi + p i_n c)] + e^{-rt}\eta(h^n(t, \boldsymbol{x}))^2. \tag{B.7}$$

B.1.2 The decoupled HJB equations in the form of (4.15)

Recall that we aim to solve (4.13) using the BSDE approach (nonlinear Feynman Kac relation). To this end, in this appendix, we will rewrite it in the form of (4.15) and identify μ^n and g^n. The first step is to identify the minimizer in the Hamiltonian (4.12). Keeping in mind that our testing case is COVID-19 where vaccines are not fully developed yet, the term $v(h_t^n) = 0$ is essentially zero in the past. Also, to focus on the lockdown/travel ban policy between different regions (as we did in the case study), we shall exclude the health policy $\boldsymbol{h}(t, \boldsymbol{x})$ from planners' decision problem, *i.e.*, $v(\cdot) = 0$, $\lambda(\cdot) = \lambda$, and $\eta = 0$ in the following derivations, and remove the dependence of $\boldsymbol{h}$ in all relevant functions. We remark that including the health policy $\boldsymbol{h}(t, \boldsymbol{x})$ is a straightforward generalization.

Recall that the Hamiltonian in (4.13) reads:

$$H^n(t, \boldsymbol{x}, (\ell^n, \boldsymbol{\ell}^{-n,m}), \nabla_{\boldsymbol{x}} V^{n,m+1}) \tag{B.8}$$

$$= b(t, \boldsymbol{x}, (\ell^n, \boldsymbol{\ell}^{-n,m})) \cdot \nabla_{\boldsymbol{x}} V^{n,m+1} + f^n(t, \boldsymbol{x}, \ell^n) \tag{B.9}$$

$$= \sum_{j=1}^{3N} b_j(t, \boldsymbol{x}, (\ell^n, \boldsymbol{\ell}^{-n,m})) \frac{\partial V^{n,m+1}}{\partial x_j} + e^{-rt} P^n[(s_n + e_n + i_n)\ell^n(t, \boldsymbol{x})w + a(\kappa i_n \chi + p i_n c)], \tag{B.10}$$

and recall that $\boldsymbol{\ell}^{-n,m} = (\ell^{1,m}, \ldots, \ell^{n-1,m}, \ell^{n+1,m}, \ldots, \ell^{N,m})$ represents the m^{th} stage strategies of all players other than n, which are given functions in this derivation. The first

order condition requires for ℓ^n:

$$0 = \sum_{\substack{j=1 \\ j\neq n}}^{N}(1-\theta\ell^{j,m})\left[\beta^{jn}s_j i_n\left(\frac{\partial V^{n,m+1}}{\partial e_j}-\frac{\partial V^{n,m+1}}{\partial s_j}\right)+\beta^{nj}s_n i_j\left(\frac{\partial V^{n,m+1}}{\partial e_n}-\frac{\partial V^{n,m+1}}{\partial s_n}\right)\right] \tag{B.11}$$

$$+2(1-\theta\ell^n)\beta^{nn}s_n i_n\left(\frac{\partial V^{n,m+1}}{\partial e_n}-\frac{\partial V^{n,m+1}}{\partial s_n}\right)-\frac{1}{\theta}e^{-rt}P^n(s_n+e_n+i_n)w. \tag{B.12}$$

The critical point given by the above equation indeed gives a minimizer of the Hamiltonian, as long as it is in $[0,1]$. Because we can show $\left(\frac{\partial V^{n,m+1}}{\partial e_n}-\frac{\partial V^{n,m+1}}{\partial s_n}\right)>0$ by comparing $V^{n,m+1}(t,\boldsymbol{x}+\epsilon_{n+N})$ and $V^{n,m+1}(t,\boldsymbol{x}+\epsilon_n)$ using their definitions, where ϵ_j is a $3N$-vector with only one nonzero entry $\epsilon \ll 1$ at j^{th} position. Intuitively, with all others players' initial condition the same, if player n starts with a higher exposed proportion $e_n+\epsilon$, it will produce more cost, comparing with the same increase proportion still being susceptible $s_n+\epsilon$. To summarize, we deduce the optimal policy for player n at stage $m+1$ is given by:

$$\ell^{n,m+1}(t,\boldsymbol{x})=\left\{2\beta^{nn}s_n i_n\left(\frac{\partial V^{n,m+1}}{\partial e_n}-\frac{\partial V^{n,m+1}}{\partial s_n}\right)-\frac{1}{\theta}e^{-rt}P^n(s_n+e_n+i_n)w\right. \tag{B.13}$$

$$\left.+\sum_{\substack{j=1 \\ j\neq n}}(1-\theta\ell^{j,m})\left[\beta^{jn}s_j i_n\left(\frac{\partial V^{n,m+1}}{\partial e_j}-\frac{\partial V^{n,m+1}}{\partial s_j}\right)+\beta^{nj}s_n i_j\left(\frac{\partial V^{n,m+1}}{\partial e_n}-\frac{\partial V^{n,m+1}}{\partial s_n}\right)\right]\right\} \tag{B.14}$$

$$\times\left\{2\theta\beta^{nn}s_n i_n\left(\frac{\partial V^{n,m+1}}{\partial e_n}-\frac{\partial V^{n,m+1}}{\partial s_n}\right)\right\}^{-1}\wedge 1\vee 0, \tag{B.15}$$

where we use the conventional notations $a\wedge b=\min\{a,b\}$ and $a\vee b=\max\{a,b\}$. Plugging (B.13) into equation (4.13) and by straightforward computation, one obtains

for the $(m+1)^{th}$ stage, $\mu^{n,m+1}(t,\boldsymbol{x};\boldsymbol{\ell}^{-n,m}) = [\mu_1^{n,m+1},\dots,\mu_{3N}^{n,m+1}](t,\boldsymbol{x};\boldsymbol{\ell}^{-n,m})^{\mathrm{T}}$ is

$$\mu_j^{n,m+1} = -\beta^{jn}s_j i_n(1-\theta\ell^{j,m}(t,\boldsymbol{x})) - \sum_{\substack{k=1\\k\neq n}}^{N}\beta^{jk}s_j i_k(1-\theta\ell^{j,m}(t,\boldsymbol{x}))(1-\theta\ell^{k,m}(t,\boldsymbol{x})), \tag{B.16}$$

$$j \in \mathcal{N}\setminus n \tag{B.17}$$

$$\mu_n^{n,m+1} = -\beta^{nn}s_n i_n - \sum_{\substack{k=1\\k\neq n}}\beta^{nk}s_n i_k(1-\theta\ell^{k,m}(t,\boldsymbol{x})) \tag{B.18}$$

$$\mu_{N+j}^{n,m+1} = -\mu_j^{n,m+1} - \gamma e_j, \quad \mu_{2N+j}^{n,m+1} = \gamma e_j - \lambda i_j, \quad j\in\mathcal{N}. \tag{B.19}$$

To write $g^{n,m+1}$ as a function of $(t,\boldsymbol{x},z)$, we first compute

$$\Sigma(\boldsymbol{x})^{\mathrm{T}}\nabla_{\boldsymbol{x}}V^n(t,\boldsymbol{x}) \tag{B.20}$$

$$= \left[\sigma_{s_1}s_1(\frac{\partial V^n}{\partial e_1}-\frac{\partial V^n}{\partial s_1}),\cdots,\sigma_{s_N}s_N(\frac{\partial V^n}{\partial e_N}-\frac{\partial V^n}{\partial s_N}),\sigma_{e_1}e_1(\frac{\partial V^n}{\partial i_1}-\frac{\partial V^n}{\partial e_1}),\cdots,\sigma_{e_N}e_N(\frac{\partial V^n}{\partial i_N}-\frac{\partial V^n}{\partial e_N})\right]^{\mathrm{T}}. \tag{B.21}$$

and then $g^{n,m+1}$ is given by:

$$g^{n,m+1}(t,\boldsymbol{x},z;\boldsymbol{\ell}^{-n,m}) = \frac{\theta^2}{\sigma_{s_n}}\beta^{nn}z_n i_n[\ell^{n,m+1}(t,\boldsymbol{x})]^2 \tag{B.22}$$

$$+\left\{e^{-rt}P^n(s_n+e_n+i_n)w - 2\frac{\theta}{\sigma_{s_n}}\beta^{nn}z_n i_n - \sum_{\substack{j=1\\j\neq n}}^{N}\theta(1-\theta\ell^{j,m}(t,\boldsymbol{x}))(\frac{\beta^{nj}}{\sigma_{s_n}}z_n i_j + \frac{\beta^{jn}}{\sigma_{s_j}}z_j i_n)\right\}\ell^{n,m+1}(t,\boldsymbol{x}) \tag{B.23}$$

$$+ e^{-rt}P^n a(\kappa i_n\chi + p i_n c). \tag{B.24}$$

$$\tag{B.25}$$

B.1.3 The derivation of the transmission rate β^{nk} in (4.26)

We denote by β the underlying transmission rate of the virus, which is assumed to be region independent. This transmission rate is the average number of people infected by an infectious person per day (assuming that the susceptible population is 100%). Thus, if a proportion S of the population is susceptible, then βS represents the average number of people infected per infectious person per day. If there are a total of P people in a population with a fraction I being infectious, then $\beta S(PI)$ is the number of newly infected per day.

Thus, in the context of a single-region SEIR model, the number of newly infected (or the influx to the exposed population) that occurs within $(t, t + \mathrm{d}t)$ is given by $\beta S(t)(I(t)P)\,\mathrm{d}t$. Dividing by P, the influx to the scaled exposed population in $(t, t + \mathrm{d}t)$ is $\beta S(t)I(t)\,\mathrm{d}t$.

Now let us consider the multi-region case and temporarily ignore the effect of lock-down. The term from (4.1)–(4.3) that gives the influx to E^n due to infection from I^k is $\beta^{nk} S^n(t) I^k(t)\,\mathrm{d}t$. To determine β^{nk}, we build this exact influx from core assumptions.

First, we quantify the number of people from region n that are infected by those from region k. Specifically, the influx in the interval $(t, t + \mathrm{d}t)$ to the unscaled exposed population n due to transmission from population k is given by

$$\sum_{\ell} (\beta f^{n\ell} S^n(t))(f^{k\ell} I^k(t) P^k)\,\mathrm{d}t, \tag{B.26}$$

where f^{ij} is the (assumed constant) fraction of people from i currently in region j at any moment in time.

Equation (B.26) is obtained by summing the number of infections in population n due to population k across each region. The summand represents these infections occurring in region ℓ. This can be seen as the term $f^{n\ell} S^n(t)$ is the proportion of population n that

are susceptible and within region ℓ. Of this population, there will be $\beta f^{n\ell} S^n(t)$ infections per infectious individual per day. Since the number of infectious from k that are in ℓ is $f^{k\ell} I^k(t) P^k$, we have that $(\beta f^{n\ell} S^n(t))(f^{k\ell} I^k(t) P^k)\,\mathrm{d}t$ is the number of new infections in population n due to population k occurring in the region ℓ within the time interval $(t, t + \mathrm{d}t)$.

Let us assume for now that $n \neq k$. Since we assume that $1 > f^{nn} \gg f^{nk}$, the terms in (B.26) besides the cases where $\ell = n$ or $\ell = k$ are negligible. Removing the negligible terms and dividing by the population P^n, we see that the influx to the scaled exposed population E^n due to transmission from population k over the interval $(t, t + \mathrm{d}t)$ is

$$\frac{P^k}{P^n} \beta (f^{nn} f^{kn} + f^{nk} f^{kk}) S^n(t) I^k(t)\,\mathrm{d}t, \tag{B.27}$$

which is exactly the influx represented by the model of $\beta^{nk} S^n(t) I^k(t)\,\mathrm{d}t$. This verifies the form of β^{nk} for $n \neq k$ in (4.26). Similarly if we take $n = k$ in (B.26) and ignore all terms other than $\ell = n (= k)$, then we derive the formula for β^{nn} in (4.26).

B.2 Detailed discussions on the enhanced DFP algorithm

B.2.1 Implementation details

In the NY-NJ-PA case study, we choose feedforward architectures for both V-networks and α-networks. Both have three hidden layers with a width of 40 neurons. The activation function in each hidden layer is $\tanh(x)$. We do not apply activation function to the output layer of V-networks, and choose sigmoid function $\rho_s(x) = \frac{1}{1+e^{-x}}$ for the α-networks. Other hyperparameters are summarized in the table below.

hyperparameter	lr	M	$N_{\text{SGD\_per\_stage}}$	N_{batch}	N_T	τ
value	5e-4	250	100	256	40	$1e^{-3}/180$

Table B.1: Hyperparameters in the case study: lr denotes the learning rate in stochastic gradient descent method, M is the total stages of fictitious play, $N_{\text{SGD\_per\_stage}}$ is the number of stochastic gradient descent done in each minimization problem (4.21), N_{batch} is batch size in each stochastic gradient descent, N_T is the discretization steps on $[0, T]$, and τ is the weight of the control part in the loss function (4.21).

B.2.2 Discussion on the choice of M and $N_{\text{SGD\_per\_stage}}$

We provide further experiments here on various choices of M and $N_{\text{SGD\_per\_stage}}$. In Figure B.1, we plot both validation loss and log loss against M for three states, which are produced by evaluating the NNs using unseen data after each fictitious play stage. In each panel, loss curves associated with different number of SGDs per stage are presented in different colors (blue = 50, orange = 75, green = 100, red = 125, purple = 150).

The numerical results show that the validation losses for all states decrease as the number of DFP stages M increases. Moreover, it shows that different $N_{\text{SGD\_per\_stage}}$ generates loss curves with similar patterns. Smaller $N_{\text{SGD\_per\_stage}}$ is more stable on the validation loss of PA. This result is consistent with [31] and [42], which convey that it is unnecessary to solve the problem extremely accurate in each stage and that a moderate number of $N_{\text{SGD\_per\_stage}}$ is sufficient. As a result, we choose $N_{\text{SGD\_per\_stage}} = 100$ in our case study.

B.2.3 Stability over different experiments

Here we present experiments to investigate the Nash equilibrium of the model with different combinations of parameters. For each combination of parameters, we use the same hyper-parameters and repeat the experiments several times. We run the algorithm for a certain computational budget, and then filter out the results with a fluctuating loss near the stopping and check the converged equilibrium. In the first combination of

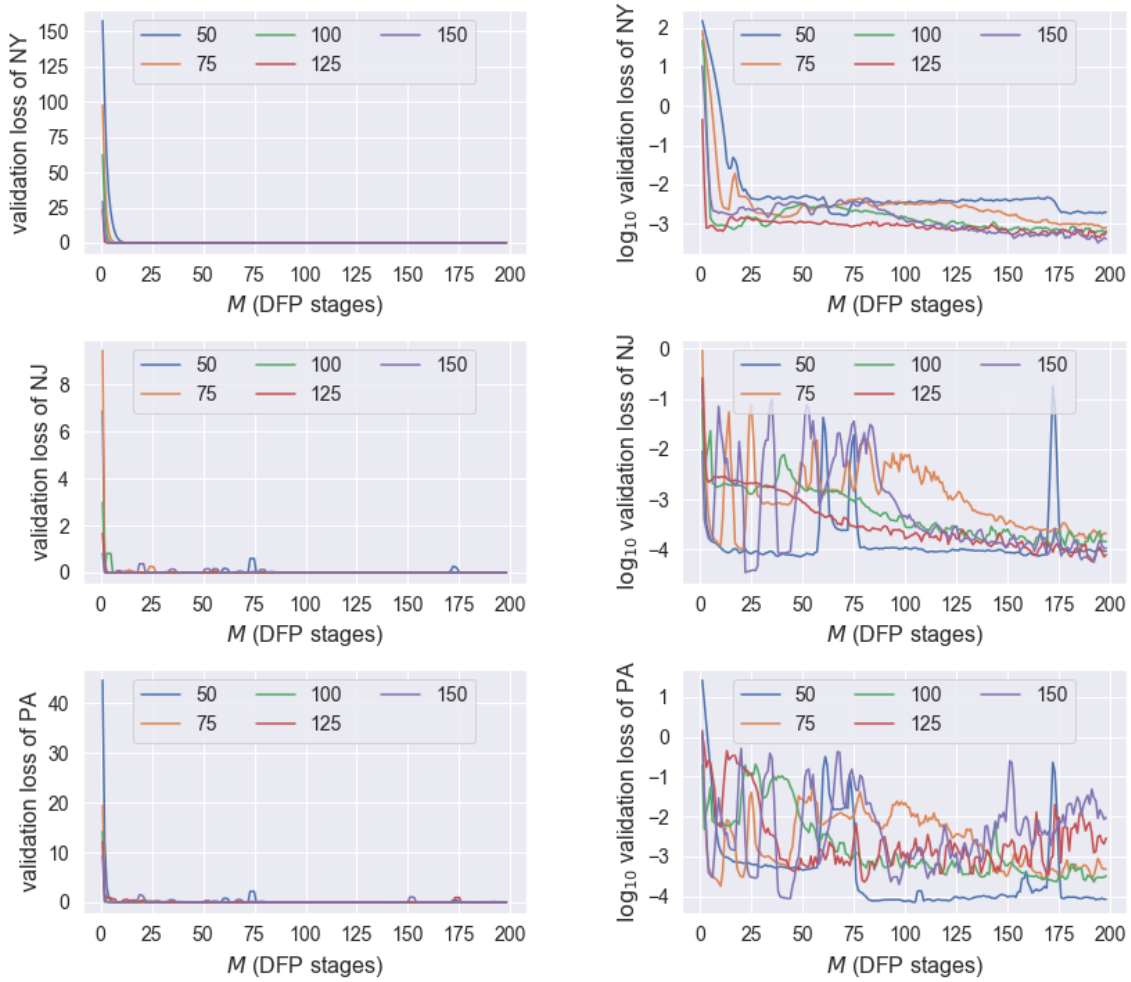


Figure B.1: Loss curves of each state. Left: validation losses versus rounds M of the enhanced deep fictitious play; Right: $\log_{10}$ validation loss versus rounds M of the enhanced deep fictitious play. The loss curves with respect to $N_{\text{SGD\_per\_stage}} = 50, 75, 100, 125, 150$ are depicted in blue, orange, green, purple and red. A smoothed moving average with window size 3 is applied in the final plots.

parameters, we take $\theta = 0.99$ and $a = 100$, corresponding to the case that a governor weighs the deaths much more than the economic loss and tries to avoid it, and the residents have a strong willingness to follow the governor's policies. Intuitively, the pandemic is possible to get well-controlled. Our numerical experiments confirm this intuition: all converging trails lead to the same Nash equilibrium. A representative plot of $X(t) = (S(t), E(t), I(t))$ is shown in Figure B.2.

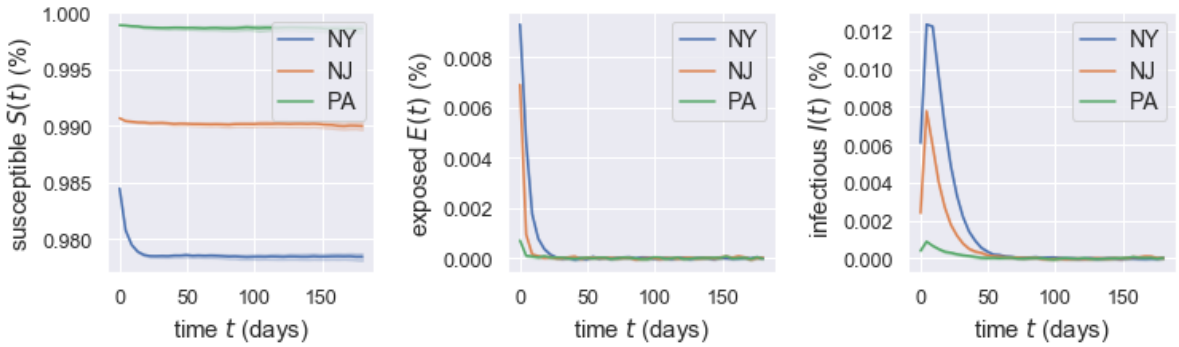


Figure B.2: With the parameter combination $\theta = 0.99$, $a = 100$, the algorithm identifies one Nash equilibrium for the NY-NJ-PA case study.

In the second batch of experiments, we take $\theta = 0.9$ and $a = 50$, corresponding to the case that a governor pays less attention to the number of death and the residents are less willing to follow the policies compared to the first batch of experiments. The change leads to the possibility of multiple Nash equilibrium and the pandemic being out of control. In this case, with different NNs' initialization, the algorithm identifies two Nash equilibria: 75% of the experiments converge to the Nash equilibrium that the pandemic gets controlled and 25% of the experiments converge to the other Nash equilibrium where the pandemic gets out of control.

In conclusion, it is possible to have multiple Nash equilibria depending on the parameter chosen in our stochastic multi-region SEIR model. There is usually a single Nash equilibrium for parameters chosen at extreme values, while for the parameters selected in the middle range, there could exist multiple Nash equilibria. When multiple equilibria exist, we conjecture that the possibility to reach a particular one depends on where we start the fictitious play (the initialization of the NNs' parameters).

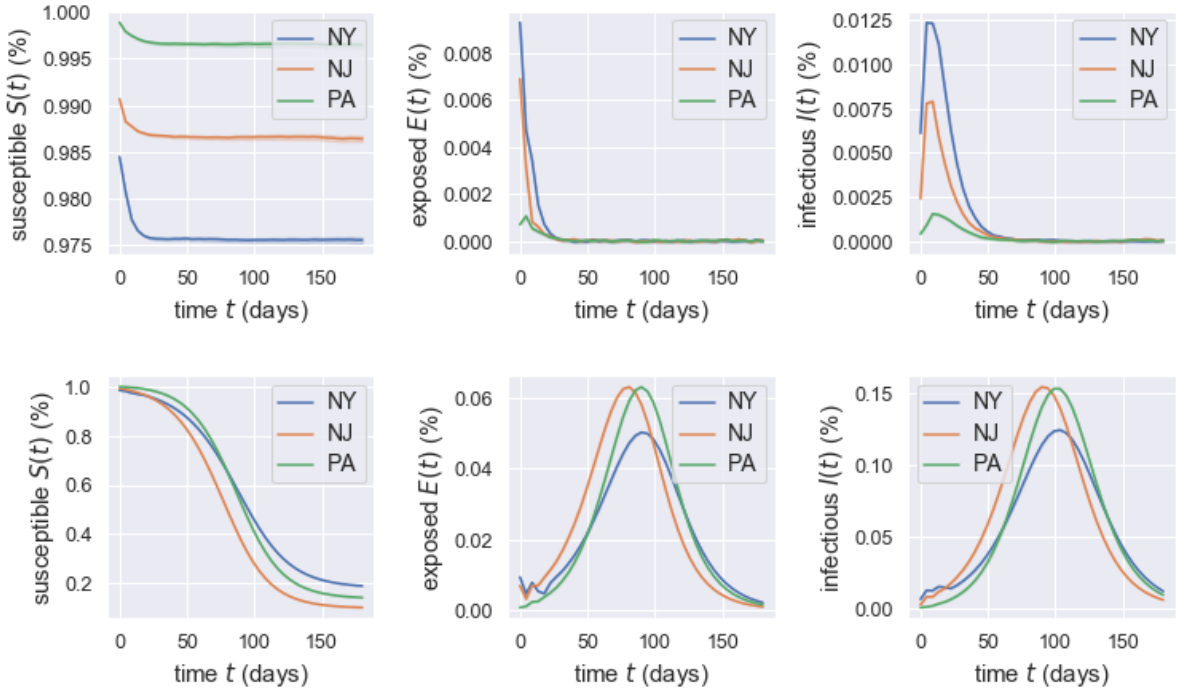


Figure B.3: With the parameter combination $\theta = 0.9$, $a = 50$, the algorithm identifies two possible Nash equilibria: an under control one (topic panels, with 75% of the experiments) and on out-of-control one (bottom panels, with 25% of the experiments).

Bibliography

[1] W. Schiesser, *Computational mathematics in engineering and applied science: ODEs, DAEs, and PDEs*, vol. 1. CRC press, 1993.

[2] G. Fredrickson, *The equilibrium theory of inhomogeneous polymers*. Oxford University Press, 2006.

[3] M. W. Matsen, *Self-Consistent Field Theory and Its Applications*, ch. 2, pp. 87–178. John Wiley & Sons, Ltd, 2007. https://onlinelibrary.wiley.com/doi/pdf/10.1002/9783527617050.ch2.

[4] F. Schmid, *Self-consistent-field theories for complex fluids*, *Journal of Physics: Condensed Matter* **10** (sep, 1998) 8105–8138.

[5] H. D. Ceniceros and G. H. Fredrickson, *Numerical solution of polymer self-consistent field theory*, *Multiscale Modeling & Simulation* **2** (2004), no. 3 452–474.

[6] P. Stasiak and M. W. Matsen, *Efficiency of pseudo-spectral algorithms with Anderson mixing for the SCFT of periodic block-copolymer phases*, *The European Physical Journal E* **34** (2011), no. 10 1–9.

[7] I. Nakamura, *Phase diagrams of polymer-containing liquid mixtures with a theory-embedded neural network*, *New Journal of Physics* **22** (2020), no. 1 015001.

[8] Q. Wei, Y. Jiang, and J. Z. Chen, *Machine-learning solver for modified diffusion equations*, *Physical Review E* **98** (2018), no. 5 053304.

[9] G. Cybenko, *Approximation by superpositions of a sigmoidal function*, *Mathematics of control, signals and systems* **2** (1989), no. 4 303–314.

[10] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, *Multilayer feedforward networks with a nonpolynomial activation function can approximate any function*, *Neural networks* **6** (1993), no. 6 861–867.

[11] K. Hornik, M. Stinchcombe, and H. White, *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, *Neural networks* **3** (1990), no. 5 551–560.

[12] W. M. Czarnecki, S. Osindero, M. Jaderberg, G. Swirszcz, and R. Pascanu, *Sobolev training for neural networks*, in *Advances in Neural Information Processing Systems*, pp. 4278–4287, 2017.

[13] R. Zhang, *Making convolutional networks shift-invariant again*, *CoRR* **abs/1904.11486** (2019) [arXiv:1904.1148].

[14] J. C. Snyder, M. Rupp, K. Hansen, K. R. Müller, and K. Burke, *Finding density functionals with machine learning*, *Physical review letters* **108** (2012), no. 25 253002.

[15] J. C. Snyder, M. Rupp, K. Hansen, L. Blooston, K. R. Müller, and K. Burke, *Orbital-free bond breaking via machine learning*, *The Journal of chemical physics* **139** (2013), no. 22 224104.

[16] J. C. Snyder, M. Rupp, K. R. Müller, and K. Burke, *Nonlinear gradient denoising: Finding accurate extrema from inaccurate functional derivatives*, *International Journal of Quantum Chemistry* **115** (2015), no. 16 1102–1114.

[17] K. Rasmussen and G. Kalosakas, *Improved numerical algorithm for exploring block copolymer mesophases*, *Journal of Polymer Science Part B: Polymer Physics* **40** (2002), no. 16 1777–1783.

[18] G. Tzeremes, K. Rasmussen, T. Lookman, and A. Saxena, *Efficient computation of the structural phase behavior of block copolymers*, *Physical Review E* **65** (2002), no. 4 041806.

[19] M. Chen, H. Jiang, W. Liao, and T. Zhao, *Nonparametric regression on low-dimensional manifolds using deep relu networks*, *arXiv preprint arXiv:1908.01842* (2019).

[20] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, *Gradient-based learning applied to document recognition*, *Proceedings of the IEEE* **86** (1998), no. 11 2278–2324.

[21] M. Valueva, N. Nagornov, P. Lyakhov, G. Valuev, and N. Chervyakov, *Application of the residue number system to reduce hardware costs of the convolutional neural network implementation*, *Mathematics and Computers in Simulation* **177** (2020) 232–243.

[22] 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* (Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger, eds.), vol. 27, Curran Associates, Inc., 2014.

[23] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, *Convolutional neural networks: an overview and application in radiology*, *Insights into imaging* **9** (2018), no. 4 611–629.

[24] M. Mirza and S. Osindero, *Conditional generative adversarial nets*, *arXiv preprint arXiv:1411.1784* (2014).

[25] A. Radford, L. Metz, and S. Chintala, *Unsupervised representation learning with deep convolutional generative adversarial networks*, *arXiv preprint arXiv:1511.06434* (2015).

[26] C. T. Bauch, A. P. Galvani, and D. J. D. Earn, *Group interest versus self-interest in smallpox vaccination policy*, *Proceedings of the National Academy of Sciences* **100** (2003), no. 18 10564–10567.

[27] C. T. Bauch and D. J. D. Earn, *Vaccination and the theory of games*, *Proceedings of the National Academy of Sciences* **101** (2004), no. 36 13391–13394.

[28] S. L. Chang, M. Piraveenan, P. Pattison, and M. Prokopenko, *Game theoretic modelling of infectious disease dynamics and intervention methods: a review*, *Journal of Biological Dynamics* **14** (2020), no. 1 1–33, [arXiv:1901.0414].

[29] R. Isaacs, *Differential Games: A Mathematical Theory with Applications to Warfare and Pursuit, Control and Optimization*. London: John Wiley and Sons, 1965.

[30] R. Hu, *Deep fictitious play for stochastic differential games*, *Communications in Mathematical Sciences* (2020).

[31] J. Han and R. Hu, *Deep fictitious play for finding Markovian Nash equilibrium in multi-agent games*, in *Proceedings of The First Mathematical and Scientific Machine Learning Conference (MSML)*, vol. 107, pp. 221–245, 2020.

[32] J. Han, R. Hu, and J. Long, *Convergence of deep fictitious play for stochastic differential games*, *arXiv preprint arXiv:2008.05519* (2020).

[33] J. Han, R. Hu, and J. Long, *Barron metric for the convergence of empirical distribution*, *in preparation* (2020).

[34] G. W. Brown, *Some notes on computation of games solutions*, tech. rep., Rand Corp Santa Monica CA, 1949.

[35] G. W. Brown, *Iterative solution of games by fictitious play*, *Activity Analysis of Production and Allocation* **13** (1951), no. 1 374–376.

[36] W.-M. Liu, H. W. Hethcote, and S. A. Levin, *Dynamical behavior of epidemiological models with nonlinear incidence rates*, *Journal of Mathematical Biology* **25** (1987), no. 4 359–380.

[37] W. E, J. Han, and A. Jentzen, *Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations*, *Communications in Mathematics and Statistics* **5** (2017), no. 4 349–380.

[38] J. Han, A. Jentzen, and W. E, *Solving high-dimensional partial differential equations using deep learning*, *Proceedings of the National Academy of Sciences* **115** (2018), no. 34 8505–8510.

[39] E. Pardoux and S. Peng, *Backward stochastic differential equations and quasilinear parabolic partial differential equations*, in *Stochastic Partial Differential Equations and Their Applications*, pp. 200–217. Springer, 1992.

[40] N. El Karoui, S. Peng, and M. C. Quenez, *Backward stochastic differential equations in finance*, *Mathematical Finance* **7** (1997), no. 1 1–71.

[41] E. Pardoux and S. Tang, *Forward-backward stochastic differential equations and quasilinear parabolic PDEs*, *Probability Theory and Related Fields* **114** (1999), no. 2 123–150.

[42] D. Seale and J. Burnett, *Solving large games with simulated fictitious play*, *International Game Theory Review* **8** (2006), no. 03 437–467.

[43] A. S. Fauci, H. C. Lane, and R. R. Redfield, *COVID-19 —navigating the uncharted*, *New England Journal of Medicine* **382** (2020), no. 13 1268–1269.

[44] N. Linton, T. Kobayashi, Y. Yang, K. Hayashi, A. Akhmetzhanov, S.-M. Jung, B. Yuan, R. Kinoshita, and H. Nishiura, *Incubation period and other epidemiological characteristics of 2019 novel coronavirus infections with right truncation: A statistical analysis of publicly available case data*, *Journal of Clinical Medicine* **538** (2020), no. 9.

[45] G. Meyerowitz-Katz and L. Merone, *A systematic review and meta-analysis of published research data on COVID-19 infection-fatality rates*, *medRxiv* (2020).

[46] S. A. Lauer, K. H. Grantz, Q. Bi, F. K. Jones, Q. Zheng, H. R. Meredith, A. S. Azman, N. G. Reich, and J. Lessler, *The incubation period of coronavirus disease 2019 (COVID-19) from publicly reported confirmed cases: Estimation and application*, *Annals of Internal Medicine* **172** (2020), no. 9 577–582.

[47] F. E. Alvarez, D. Argente, and F. Lippi, *A simple planning problem for COVID-19 lockdown*, Working Paper 26981, National Bureau of Economic Research, 2020.

[48] L. Peng, W. Yang, D. Zhang, C. Zhuge, and L. Hong, *Epidemic analysis of covid-19 in china by dynamical modeling*, 2020.

[49] R. E. Hall, C. I. Jones, and P. J. Klenow, *Trading off consumption and COVID-19 deaths*, Working Paper 27340, National Bureau of Economic Research, 2020.

[50] F. Health, *Costs for a Hospital Stay for COVID-19*, 2020.

[51] N. D. R, P.-Y. Huang, and F.-C. Yeh, *Structural analysis and optimization of convolutional neural networks with a small sample size*, *Scientific Reports, Nature* **10** (2020), no. 1 834.