Deep reinforcement learning and representation learning for chaotic dynamical systems

by

Kevin Zeng

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Chemical and Biological Engineering)

at the

UNIVERSITY OF WISCONSIN - MADISON

2023

Date of final oral examination: April 10, 2023

This dissertation is approved by the following members of the Final Oral Committee:

Michael D. Graham, Professor, Chemical and Biological Engineering

Dan Klingenberg, Professor, Chemical and Biological Engineering

Reid C. Van Lehn, Associate Professor, Chemical and Biological Engineering

Jean-Luc Thiffeault, Professor, Mathematics

"When the present determines the future, but the approximate present does not approximately determine the future."

– Edward Norton Lorenz, *On Chaos*

Dedication

The works throughout this thesis have brief and limited authorships–while this list of names explicitly credits those who developed these works, it does not credit those whose love, support, and company lifted me into such fortunate opportunity. This journey could not have been completed without their support and I would not be the person I am today.

I am eternally grateful to my family–my mother, father, sister, and grandmother who, through every moment of uncertainty and success, have given me so much unconditional love and support. You have celebrated my every stride and lifted me through every stumble along this journey. I am grateful to have such a warm and gentle foundation to stand on.

To all the old and new friends who have brought me so much joy and taught me so much, I am thankful–you have all introduced so much color into my life.

To Eric Yu and Alec Linot, I am honored to have had the chance to worked alongside such hardworking, driven, and talented friends. Thank you, Eric, for always being there to support me; you have been so selfless and have always extended a helping hand. To Alec, it has been a pleasure to work with you and bounce ideas around–I owe it to you for helping me leave my comfort zone and to take more pride in my work.

To RJ Hommel and Carlos Perez de Jesus, I am grateful to have made such great and funny friends. You have taken away the daily monotony and filled my memories with bad jokes and laughter. To RJ, I greatly admire your proactiveness in bringing attention to important issues and voicing your concerns for your fellow colleagues while also juggling a PhD–you have shown me how much impact a single person can have, and I am honored

to have been able to work on the Newsletter with you. To Carlos, I am thankful for you introducing me to so many new things I would have never tried without your encouragement– running, cocktails, music, and even comedy.

To Michelle Choi, Ricardo Constante-Amores, and Bryce Connors, I am grateful to have become friends with such genuine and scientifically passionate people. To Michelle, I admire your honesty and confidence–you have shown me to worry a little less about what people think. To Bryce, I admire your thoughtfulness and serene nature–you bring an energy that lifts the atmosphere wherever you go. To Ricardo, I admire your humbleness and drive–I am thankful for all of your encouragement and advocacy.

To Alvin Jonathan and Joe Wang, I am grateful to have become friends with such wonderful people. Alvin, you have showed me how important it is to balance life with hard work and to accept the outcome when we try our best. Joe, your determination and natural curiosity has inspired me to pursue my passions and interests unapologetically. And while our paths have separated from those long nights in Keller Hall, I hope that they will cross again in the future.

To Ly Nguyen, I am forever grateful to have had all of your support, encouragement, and companionship. You always find confidence in me even when I doubt myself and have given me more strength than I could have asked for. I admire your hard-working nature and self-initiation (despite your humbleness) and you have inspired me to improve myself on many occasions. I am truly thankful to have someone so warm-spirited, good-hearted, and caring along this journey.

I am fortunate to have received the guidance and confidence from so many mentors and advisors over the past ten years. To Julia Zhao and Steve Wu, thank you for giving a curious high-school student the opportunity to experience first-hand research and solidifying my interest in science and engineering. To Samira Azarin and Frankie Pelaez, thank you for all the guidance you have provided both inside and outside the lab. Without the confidence and opportunities you provided I may not have taken up the challenge of graduate school

at all. To Karthik Nayani, I am grateful to have a mentor teach me early on in graduate school to set boundaries between work and life. To Daniel Floryan, I am thankful to have had such a great mentor show me science is beyond just obtaining results, it is also how to present and communicate it to others. To my advisor Mike Graham, who I am incredibly grateful for extending a hand of opportunity to me when my future seemed so unclear, I am thankful for all your patience, guidance, confidence, and lessons you have shared with me, and I am grateful for your support and compassion during times of uncertainty. I have learned so much from you and I love the world of research you introduced to me. Finally, I am thankful for my Committee, who have followed me along this journey, challenged my ideas, and introduced new perspectives.

Abstract

Many ubiquitous phenomena in nature and engineering, such as turbulent flows, global weather patterns, and reaction-diffusion systems, can be described by dissipative infinite dimensional partial differential equations. Despite their prevalence, these systems often remain the source of engineering challenges when it comes to modeling and control. Specifically, generalizable and automated frameworks for reduced-order modeling and control remain an obstacle due to a number of systemic challenges such as complex spatiotemporal chaotic dynamics, high-dimensionality, and costly data generation. While many deep learning frameworks have experienced dramatic success in their fields of origin, their direct application towards our target systems can often be unsatisfactory or even intractable without innovation. Motivated by this disconnect, the main objective in this thesis is therefore to develop data-driven frameworks that combine concepts from dynamical systems theory, such as symmetries and manifolds, with deep learning, such as deep reinforcement learning (RL) and representation learning, to efficiently and automatically find control strategies and low-dimensional representations for complex dynamical systems.

In Chapter 1, we motivate our systems of interest from a dynamical systems perspective and provide background on core concepts. From here, we branch out into two themes of discussion–control and data representations. The primary theme, which is focused on the control of chaotic flows, overviews existing control methods, their successes, and their deficiencies. In particular, these deficiencies motivate us to turn to data-driven control via modern deep reinforcement learning, which we review in-depth. We then shift our attention

to the second theme of learning representations, which is motivated by challenges encountered in data-driven reduced-order modeling of our complex high-dimensional systems. Here, we provide a review of dominant dimensionality reduction methods with an emphasis on deep autoencoders.

In Chapter 2, we explore standard deep RL's (practical) inability to learn a truly optimal policy for dynamical systems with symmetries. We demonstrate this issue using the Kuramoto-Sivashinsky equation (KSE) equipped with 4 synthetic jets and a control objective of minimizing energy dissipation and power input. We showcase that the representation of the learning problem is critical and introduce a framework that reduces the symmetry complexity of the state representation. From a dynamical systems perspective, this framework effectively moves the RL problem into a symmetry-reduced subspace of state-action space. As a result, this framework not only guarantees controller equivariance but also substantially boosts data-efficiency and network capacity utilization.

We shift gears in Chapter 3 and demonstrate the feasibility of using manifold models inspired by dynamical systems theory to efficiently train RL agents. In this chapter we develop a model-based RL framework we call "Data-driven Manifold Dynamics for RL" or DManD-RL. The development of this framework is largely motivated by dynamical systems that are too expensive to apply RL to in the traditional sense (e.g. turbulent flows!). Notably, the RL agent learns in a low-dimensional manifold coordinate representation instead of the high-dimensional ambient space. We demonstrate the feasibility of this framework again on the Kuramoto-Sivashinsky equation.

We make the leap of faith in Chapter 4 and apply DManD-RL to a high-fidelity simulation of turbulent flow in the plane Couette geometry. Here we tackle all of the listed challenges: high-dimensional data, costly data generation, and highly complex chaotic dynamics. We demonstrate that our DManD-RL model is able to learn a state-of-the-art drag-reducing control strategy compared to existing heuristic-based strategies. We highlight our framework enabled a $\sim 400\times$ training speedup compared to a traditional application of RL.

In Chapter 5, we shift our focus purely onto representation learning, largely motivated by challenges that remain in reduced-order modeling and forecasting. In this chapter, we demonstrate a modified application of implicitly regularized autoencoders to develop a framework that automatically: 1) estimates the underlying dimensionality of the manifold the data lives on, 2) provides a working orthogonal manifold coordinate system, and 3) provides the mapping functions between the ambient representation and manifold representation. We demonstrate the viability of this framework on a number of example systems and showcase its natural extension to time-series forecasting. We conclude this chapter with an investigation of how our framework achieves these properties automatically.

In Chapter 6, we conclude this thesis with a summary of our insights and layout the foundation of potential future work. These future directions focus on 1) feasibly extending RL to significantly larger spatial domains via hierarchical distributed systems, and 2) extending learning representations to take advantage of the unique properties of dynamical systems data by considering geometric and graphical properties in the latent space.

Contents

List of Figures

1

General introduction

The works presented in this thesis are aimed at leveraging ideas from dynamical systems theory and machine learning to develop data-driven frameworks that efficiently find control strategies and reduced-order representations for complex dynamical systems. We are primarily interested in dissipative, chaotic, "infinite-dimensional" systems described by partial differential equations (PDEs). These systems describe ubiquitous phenomena in physics and engineering, such as reaction-diffusion systems, weather patterns, and turbulent fluid flows.

In this thesis, turbulent flows are our white-whale system for a number of industrially and academically motivated reasons. Industrially, the reason is simply economical–turbulent flows incur turbulent drag which incur energy losses. Wall-bounded turbulent flows dissipate approximately 25% of the energy expended in commerce and industry–on a global scale, this accounts for about 5% of the global carbon emission footprint [95]. On such a large scale, the economical and ecological savings that could be enabled by advancements in drag-reduction (i.e. flow control) is not trivial. Academically, turbulent flows and their dynamics remain far from fully understood. Advancements in the control and modeling of turbulent dynamics may allow us to glean insight into the hidden structure of this complex dynamical system–which in turn may inform us about other similar systems.

From a high level, many of these systems present a number of common, nontrivial challenges: complex spatiotemporal chaotic dynamics, high-dimensionality, and resource-costly

data generation. In the context of turbulent flows, which are known to be governed by the Navier-Stokes Equations (NSE), these properties have made traditional/heuristic-based control and high-fidelity modeling a significant challenge at best.

In recent years accessibility to powerful computing, data, and machine learning have blossomed, making data-driven approaches not just appealing but increasingly practical. While many deep learning frameworks have experienced extensive success in their fields of origin, their application towards our systems of interest can often be unsatisfactory or even intractable due to the above challenges. In the works presented in this thesis, we will tackle these challenges by specifically building upon deep reinforcement learning for data-driven control and regularized deep autoencoders for automated dimensionality reduction. Our end goal is to develop and demonstrate the effectiveness of these frameworks for dynamical systems as complex as turbulent flows.

The following introduction is set-up as a primer for key concepts in nonlinear dynamical systems theory and deep learning, which is focused on reinforcement learning and representation learning. The introduction is laid out as follows: In Sec. 1.1 we define the dynamical systems we are interested in and their properties. In Sec. 1.2 we review classical methods that have been developed to control these dynamical systems. In Sec. 1.3 we provide a brief review of neural networks, weight optimization, and a statement on what we can expect from our networks. In Sec. 1.4 we overview core concepts and derive equations of reinforcement learning that are necessary to understand modern deep reinforcement learning algorithms. In Sec. 1.5 we describe these deep reinforcement learning algorithms. We then deviate from our discussion of data-driven control to discuss representation learning in Sec. 1.6 in the context of dimensionality reduction and forecast modeling. We complete our discussion with a brief overview of data-driven forecasting methods in Sec. 1.7 and provide an outline of the remaining chapters in Sec.1.8.

1.1 Dynamical Systems

1.1.1 Formalizing our Dynamical Systems

In this thesis, we are concerned with the modeling and control of "infinite-dimensional" dynamical systems, which are described by a system state, $v(x,t)$, that evolves in space, x, and time, t. The system can be formalized as a partial differential equation whose right-hand-side (RHS) is generically defined as $\mathcal{F}$ and an initial condition,

$$\begin{aligned} \frac{dv}{dt} &= \dot{v} = \mathcal{F}(v) \\ v(t=0) &= v_0 \end{aligned} \tag{1.1}$$

For practical purposes, such as computational modeling and analyses, these systems are often "approximated" in a finite, high-dimensional form that represents v as time-varying modal coefficients on some orthonormal basis. For example, we can describe $v(x,t)$ as the sum of the time varying modes:

$$v(x,t) = \sum_{i=1}^{\infty} u_i(t)\phi_i(x) \cong \sum_{i=1}^{d_u} u_i(t)\phi_i(x), \tag{1.2}$$

where $u_i(t)$ are the time varying modal coefficients and $\phi_i(x)$ are the orthonormal spatial basis. Naturally, given $\mathcal{F}(v)$ and the above relationship, we can perform Galerkin projection,

$$\frac{du_j}{dt} = \left(\mathcal{F}\left(\sum_{i=1}^{d_u} u_i(t)\phi_i(x) \right), \phi_j(x) \right), \tag{1.3}$$

to obtain what is known as the full-order model (FOM) or full system,

$$\frac{du}{dt} = \dot{u} = f(u), \tag{1.4}$$

where $u \in \mathbb{R}^{d_u}$ is in the *ambient* space of finite dimension d_u. For a system with control

inputs, a, we can describe the FOM as the following when control is affine,

$$\frac{du}{dt} = f(u) + b(u)a, \tag{1.5}$$

where b maps control inputs to the RHS contribution. For control that is non-affine, which is common in many real-world applications, we can describe the FOM as,

$$\frac{du}{dt} = f(u, a). \tag{1.6}$$

We narrow the scope of our focus onto systems described by f that are *deterministic* and *nonlinear*. The time evolution and future states of a deterministic system is dependent only on u (i.e. independent of the history of u). While many physical phenomena can be described by linear dynamics (e.g. small angle undamped pendulum, flow over a cylinder with a periodic oscillatory wake), the bulk majority of complex systems we are interested in, such as turbulent flows, are highly nonlinear.

These deterministic nonlinear dynamical systems can often give rise to *chaos*. Chaotic dynamical systems are systems that possess deterministic, aperiodic long-time dynamics coupled with an extreme sensitivity to initial conditions [185]. These systems exhibit extreme sensitivity to initial conditions, where states starting arbitrarily close together evolve apart exponentially onto radically different paths. This property, when coupled with the long-time aperiodic dynamics, gives chaos the appearance of disorder and randomness. However, underlying the apparent disarray, there is structure in the form of embedded patterns, self-similarity, and organization.

Two dynamical systems concepts that are important are *invariant solutions* and *manifolds*. *Invariant solutions* are special states or sets of states that if the system were to begin on them, they would remain in them. These include fixed points, periodic orbits, relative periodic orbits, etc [185]. Invariant solutions can have stable and/or unstable directions around them, influencing the dynamics when the system passes near them. This influence

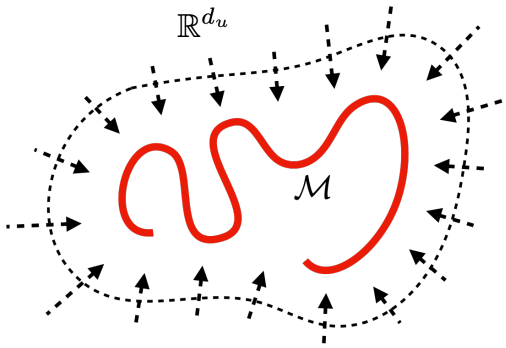

Figure 1.1: Cartoon of long-time dynamics approaching the $\mathcal{M}$ manifold embedded in the $\mathbb{R}^{d_u}$ ambient space.

leads to the dynamics self-organizing about these solutions. From this perspective, invariant solutions can be viewed as the underlying "skeleton" of the dynamics we observe.

Many physical phenomena in physics and engineering also have long times dynamics that reside on a lower-dimensional *manifold*, illustrated as a caricature in Fig. 1.1. For example, at low Reynolds numbers a flow past a cylinder will eventually develop Kármán vortex shedding, where the unsteady separated flow behind the cylinder oscillates regularly. Here the long time dynamics converge onto a rather trivial manifold–a periodic orbit. In this example, the manifold also happens to be a single invariant solution. However, for more complex systems, the manifold can be organized about multiple solutions, taking on complicated, nontrivial forms that are aperiodic. For the interested reader, a more rigorous discussion of manifolds is provided in Sec. 1.1.2. Our white-whale system, turbulent flow, is also a dissipative chaotic dynamical system. Like other dissipative dynamical systems, they exhibit dynamics that are organized about invariant solutions, called Exact Coherent States [66], that take on the forms of equilibriums, traveling waves, periodic orbits, etc. For these reasons, turbulent flows are strongly suspected to also have dynamics that live on a lower-dimensional manifold [190]. Through this dynamical systems lens, we are able to disconnect this outward appearance of randomness and disorder from our system, giving us clear hope that they can be modeled and controlled.

1.1.2 A More Formal Description of Manifolds

As defined by Lee [110], a topological manifold, $\mathcal{M}$, of dimension d_m is a topological space that holds the following three properties:

1. $\mathcal{M}$ is a Haussdorff space
2. $\mathcal{M}$ is second-countable
3. $\mathcal{M}$ is locally Euclidean of dimension d_m

The first property states that for every pair of distinct points, $p, q \in \mathcal{M}$, there are disjoint open sets (i.e. the intersection of these sets is empty) $U, V \subseteq \mathcal{M}$ such that $p \in U$ and $q \in V$. The second property states that there exists a countable basis for the topology of $\mathcal{M}$ and implies that the space is separable. The final property states that each point in $\mathcal{M}$ has a neighborhood that is homeomorphic to an open subset of $\mathbb{R}^{d_m}$, which means for each $p \in \mathcal{M}$ we can obtain 1) an open subset of $U \subseteq \mathcal{M}$ that contains p, 2) an open subset $\hat{U} \subseteq \mathbb{R}^{d_m}$, and 3) a homeomorphism $\phi : U \to \hat{U}$.

There are several important implications these properties leave us: 1) by definition the manifold has a specific integer dimension and 2) a homeomorphism between $\mathcal{M}$ and $\mathbb{R}^{d_m}$ is only required *locally*. For the purposes of dimensionality reduction, this means that if we seek to find a global map between $\mathcal{M}$ and $\mathbb{R}^{d_m}$, it is not guaranteed to exist. However, while a global map to $\mathbb{R}^{d_m}$ may not always exist, a global map will exist for $\mathbb{R}^{2d_m}$ where $2d_m$ is the upperbound [212]. If such a mapping were insufficient, an alternative route can be taken. Mappings from $\mathcal{M}$ to $\mathbb{R}^{d_m}$ can be achieved by assembling local patches of the manifold, called *charts*, where each chart locally maps their portion of the manifold to $\mathbb{R}^{d_m}$ [46]. A collection of charts can then be assembled into what is called an *atlas* to cover the entire manifold.

Connecting these ideas back to our discussion of dynamical systems, we can treat the set of states at long-times as belonging to a manifold. Because the manifold is finite-dimensional of dimension d_m, we can infer that the underlying dynamics that represent our "infinite-

dimensional" PDE systems are also finite and of dimension d_m. For the purposes of finding minimal representations, reduced-order modeling, and control, we have a lower-bound on the dimensionality we can work with.

1.2 Classical Control of Chaotic Systems and Flows

1.2.1 Chaos Control

As the systems we seek to control are chaotic dynamical systems, it is worth first discussing the pioneering works in chaos control. These strategies can be classified into closed-loop (feedback) or open-loop (non-feedback). We first briefly overview non-feedback methods. These methods are desirable for the practicality of their application–requiring no online monitoring, live computations, or knowledge of fixed points. These control methods aim to *suppress* chaos by driving the chaotic dynamics toward periodic dynamics. This is accomplished by applying weak perturbations in the form of constant, periodic, or quasiperiodic signals to some control variable [159]. However, because these methods are not optimized in the sense that they require continuous energy expenditure, estimations of resonant frequencies, and are not reactive to disturbances, we will focus our attention to the latter class of methods.

We now turn our attention to feedback methods. Feedback methods require some observation of the system and calculating a proper control perturbation to achieve a desired evolution of the system. We will first overview methodologies pioneered by the chaos community first, which characteristically require a target state or orbit known a priori. We will then broaden the scope of methodologies to frameworks developed by the fluid dynamics and machine learning communities.

The pioneering feedback chaos control method, devised by Ott, Grebogi, and Yorke (OGY method)[145], takes advantage of two properties of chaotic dynamical systems: 1) a chaotic system has embedded within it a large number of unstable periodic orbits meaning trajec-

tories will eventually visit the neighborhood of targeted or desirable states due to ergodicity and 2) the high sensitivity of trajectories to the current state indicates that the behavior of the system can be largely influenced with just small perturbations. The resulting methodology consists of the following steps. First, one identifies a set of low-period periodic orbits (or fixed points). From this set, a solution with desirable properties is chosen by evaluating the stability in space (via linearized poincaré maps). Using a small control perturbation, the target orbit is stabilized by perturbing the trajectory onto the stable direction of the solution when the system approaches sufficiently close to the neighborhood of the target orbit. In the event the controlled trajectory is lost, control is removed until the system naturally returns to the target neighborhood. Naturally, there is a trade off between how large of a perturbation is available (i.e. the neighborhood of control) and the average "wait" time for the system to naturally approach the neighborhood.

The OGY method requires the identification of solutions with desirable properties, as well as the identification of the unstable and stable directions along each point of the solution, which can be practically difficult and prohibitively expensive for arbitrary high-dimensional systems. This becomes even more difficult when perturbations are physically and practically constrained such as in turbulent flows. Furthermore, because the linear controller only acts when the trajectory is within the target neighborhood, systems with dynamics that are slow or intermittent may require waiting for unacceptable lengths of time. For a more thorough discussion on the OGY method, the interested reader is referred to Boccaletti et al. [8], Ott et al. [145].

Another notable feedback control method is the delayed feedback method introduced by Pyragas [151], whereby a feedback perturbation, $U(t)$, acts on one of the state variables. The underlying assumption is that there exists a scalar variable in x (here as an abuse of notation we use x as the state variable to be consistent with the original work) that is available for

measurement and perturbation. The form of the feedback control perturbation is as follows,

$$U(t) = K(x_1(t - T) - x_1(t)) \tag{1.7}$$

where K is a parameter weighting the perturbation, x_1 is the observable/perturbable variable, and T is the period of the target orbit. Generally, the choice of T is fixed once a target orbit is determined, with K being determined by sweeping over various values until desirable performance is achieved. Note that this controller is essentially a time-delayed feedback controller that attempts to drive the system to follow some predetermined unstable periodic orbit. Various improvements and renditions to this method have been introduced since, which are covered in a review by Boccaletti et al. [8].

These classical methods demonstrated that chaotic dynamics, despite all their complexity and sensitivity, could be controlled. Practically, these control solutions are far from ideal as they provide no globally optimized control solution, require a priori knowledge of invariant solutions, cannot address arbitrary control tasks, and are not very generalizable.

1.2.2 Flow Control

As turbulent flows are some of the most complex high-dimensional dynamical systems, they have stood as long-time engineering and industrial economic challenges. The main engineering challenge we are interested in is *drag* induced by turbulence (which incurs significant energetic losses in industrial transport) and how to develop strategies to mitigate it. In this introduction and throughout this thesis, we use the phrase *flow control* synonymously with strategies that promote *drag reduction*.

Here we dedicate a section to overview notable control strategies developed in the flow control community along with their advantages and their deficiencies. We first narrow our scope of turbulent flows to those bounded by no-slip walls, such as plane Couette and plane Poiseuille flow, simulated in high-fidelity as direct numerical simulations (DNS). These

canonical flows are an important point of study as they isolate the characteristic flow bounded by surfaces (such as air over an airplane wing or a liquid in a transport pipe) in a relatively tractable domain. Within these flow geometries, we begin our flow control overview by first narrowing the class of flow control to *reactive* flow control methods, which are methods that require energy expenditure for control and react in response to the system state (i.e. closed-loop, feedback). These methods are generally superior in performance over non-feedback methods, which do not consider the state of the system. Non-feedback methods can appear as *passive*, which do not expend energy for control, or *active*, which do. Several notable passive non-feedback examples include riblets and other surface modifications [3, 21, 62], and active non-feedback examples such as wall-oscillations [153, 154]. Notable reactive flow control methods include opposition control [23, 78] and model-predictive control [6].

Opposition control, introduced by Choi et al. [23], is a reactive flow control method which manipulates the wall-normal velocity at the wall using a simple opposition control law to reduce drag. More precisely, opposition control sets the wall-normal velocity, v, at the wall to be equal to the opposite of v (up to a constant of ϕ) at some $x-z$ detection plane located at y_s in the interior of the flow. In other words, the control law is defined as $v(x, y = \pm 1, z) = -\phi v(x, y = \pm y_s, z)$. Performance of course is dependent on y_s and ϕ, which are optimized empirically.

The main drawbacks of this method are its need for high-resolution actuation and its dependence on heuristics. The first drawback is a practical one: the ability to freely control the wall-normal velocity at the boundary at such a high resolution requires sensor and actuator array densities that are impractical. The second drawback is the method is based on physical heuristics and intuitions. The idea is if one can disrupt the near wall flow dynamics, the structures that lead to drag are also disrupted. If one were interested in adjusting the physical form of control, it is unclear how the control law should change. Thus, heuristics can only go so far. Opposition control can only disrupt some near-wall flow structures–without any real optimization, this method cannot address the problem at a global scale.

In terms of optimization-based flow control, several works have highlighted the value and efficacy of applying optimal control theory frameworks to choose control responses. A notable application of model-predictive control is implemented by Bewley [6]. In this work, the authors controlled a turbulent DNS simulation by optimizing a future sequence of control actions by running forward simulations and solving an adjoint problem backwards. In this work, the authors manipulate the wall-normal velocity at the wall. While achieving notable drag reduction, the caveat of this method is the compute and memory cost required on-the-fly. Because each action requires a forward solve of the DNS (and solving an adjoint system backwards), this method cannot be practically implemented. While turbulent structures appear and disappear in a fraction of a second, it would take on the order of minutes to hours to optimize a single action. Furthermore, the velocity fields obtained from the NSE must be stored for all time–these storage requirements can be prohibitive for three-dimensional long-time horizon optimization problems. For a more comprehensive review of variations of these flow control methods, we refer the reader to Scott Collis et al. [175].

The deficiencies outlined here motivate us to look for a control strategy framework that is globally optimized for reducing drag but can also be "front-loaded" ahead of time so that determining a control output during deployment is quick. Furthermore, it is ideal that the framework is agnostic to the form of the actuators and sensors and can be automated without relying on heuristics. In the following sections, we will overview the choice method for many of the works described in this thesis, deep reinforcement learning, which will allow us to obtain control strategies while avoiding the issues mentioned above.

1.3 Deep Learning Basics and a Practical Perspective

1.3.1 The Basics

The majority of the frameworks developed in this thesis utilize artificial Neural Networks (NNs), which are excellent at approximating nonlinear functions from data. Similar to

biological neurons, artificial NNs are composed of many interconnected units. These interconnected units are commonly organized into *layers*, which together form a neural network. The term "deep" simply refers to architectures with more than two layers.

The most basic architecture is a feed-forward network or multi-layer perceptron (MLP). As input data pass through each layer, they are transformed or *activated* via nonlinear functions. The intermediate layers of the network are often called *hidden layers* and the intermediate representation of the data, z, is often called a *latent representation*. The representation at layer i can be computed as the following,

$$z^{(i)} = f^{(i)}(W^{\top(i)} z^{(i-1)} + b^{(i)}), \tag{1.8}$$

where $W^{\top(i)}$ is the weight matrix at layer i, $b^{(i)}$ is the bias at layer i, and $f^{(i)}$ is the nonlinear activation function applied point-wise at layer i. The free parameters sought for optimization are the set of weights W and b which are often referred to collectively as parameters θ. Weights are optimized to reduce some user-defined *loss function*, $\mathcal{L}$, which is an equation or relationship the network aims to minimize. Weights are updated iteratively in fitting batches, where forward passes are used to compute the loss, and *back propagation* (essentially applying chain-rule in reverse) is used to compute the gradient of the weights with respect to the loss. Using some gradient descent method, the weights are then adjusted and the cycle repeats until the desired performance is reached [64]. Here we have detailed the most basic practical NN formulation, the MLP. There are many different formulations NNs can take-on, which is generally dictated by the task or data. A more detailed review of these can be found in Goodfellow et al. [64].

1.3.2 A Healthy, Practical Perspective on Neural Networks

In this section, we share an important *opinion* on what a healthy perspective one should have when working with network-based methods in terms of expectations and limitations.

NNs, like the multitude of other modeling frameworks, are tools used to approximate functions. They have found applications within nearly every research field due to their flexibility, shockingly good performance, and news-worthy accomplishments. However, in terms of our fundamental knowledge on them, their constant successes and new developments far outrun our concrete understandings. *This can dangerously lead to our expectations outrunning our understandings of their limitations.* It is important to remind ourselves that they are practically just *approximators*.

While there have been several universal approximation theorems developed that highlight the potential power of NNs, it is important to note that many of these are proofs on *existence* [33, 87, 88]–they do not guarantee that any network will perform well, especially in practical applications with finite practical datasets and finite available compute. NN performance is dictated by an astronomical number of parameters we can control (e.g. data/learning representation, layers depth/width, layer types, activation functions, optimizers, data-preprocessing, prior distribution assumptions, weight initializations, architecture, batching, normalizations, sampling, etc.) as well as many we cannot control (e.g. how much data we have, the underlying structure/distribution of the data, the quality of our data, compute limits). While each of these can significantly impact performance, the field has not developed a concrete theoretical understanding of the interplay between all of these properties and the limitations they impose (and it is unclear the field will ever be able to). The inherent challenges of noise, imperfect optimization, and stochasticity loosen these expectations further. Just because a NN theoretically could (under specific conditions) approximate your desired function, does not exclude the hundreds of practical reasons why it might not. This notion caries over to frameworks developed based on heuristics. For example, SimCLR uses a projection head layer to transform its latent representation because it performed better based on the authors' observations [18]. There is no concrete reason this design choice should be expected to be optimal for *all* datasets.

A healthy, practical perspective of NNs is that they are simply function approximators

that are powerful yet still poorly understood. To the date of this thesis, researchers are still trying to understand the internal mechanisms in deep learning on a theoretical level that lead to performance, generalizability, and feature formation. At the same time, NNs suffer from a wide array of practical deficiencies and they should not be viewed as infallible perfect approximators when subjected to arbitrary data, heuristic design choices, and ill-defined objective functions. It is therefore important we stay wary of the many underlying limitations of NNs and to exercise caution with our expectations of them when we are applying or designing frameworks.

1.4 Reinforcement Learning

In this section we briefly outline the core concepts and fundamentals of reinforcement learning [186]. These fundamentals are quite important to understand in order to appreciate and implement modern reinforcement learning algorithms, which are the frameworks utilized throughout this thesis. We will first begin with some basic terminology, a description of the general problem, and derive the core functions that are ubiquitous in deep RL. We refer the interested reader to Sutton and Barto [186] for additional background and its interesting connections to neuroscience and psychology.

1.4.1 Reinforcement Learning Basics

In the simplest terms, reinforcement learning (RL) is concerned with learning the mapping function between system states and actions that maximizes the cumulative sum of a reward signal through repeated interactions. Fundamentally, RL is analogous to the biological learning process–we learn through observation of the cause and effects of our actions on our surroundings as well as the impact they have on our goals. RL is thus the computational embodiment of this learning process. RL divides the world into two domains: the *agent* and the *environment*. The agent is the learner tasked with decision making, while the environ-

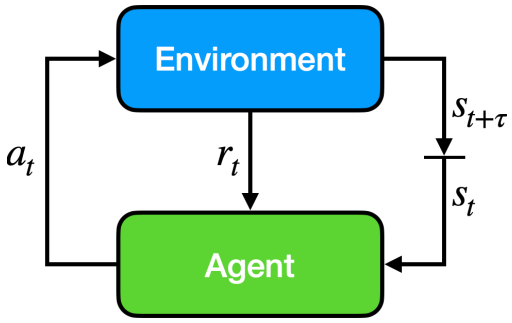

Figure 1.2: The basic reinforcement learning cycle.

ment is everything exterior of the agent. Between the agent and the environment are several important subcomponents. These are the *state*, *action*, *policy*, *reward signal* and *value function*. As the name suggests, the environment *state*, $s \in \mathcal{S}$, is an observation made by the agent on state of the environment. Importantly, s is not necessarily equivalent to the full state of the dynamical system, u, and can be a partial observation of the state. The *action*, $a \in \mathcal{A}$, is the control response implemented by the agent to manipulate the environment. Here and throughout, we denote the concept of time or sequence with the subscript t, e.g. s_t represents the observation of the state made at time t. The choice of action is determined by the (deterministic) *policy*, μ, which is the mapping between s_t and a_t,

$$a_t = \mu(s_t). \tag{1.9}$$

The policy is the functional form of the control "strategy" or stimulus-response rules and can be stochastic (as we will see later). Note here that the agent is the embodiment of the policy and that these terms are often used interchangeably in literature. In RL, policies are optimized to maximize the *reward signal*, $r \in \mathcal{R}$, over the future horizon. r is a scalar valued function that relays how well the agent is behaving immediately and if evaluated over the interval at t, we denote it as r_t. The reward is defined by the user and dictates the control objective and is generally a function of the environment state and action, $r = r(s, a)$. Finally, the *value function*, V_μ, quantifies how much reward the agent can expect to accumulate in

the future starting from a given state i.e. how good is this state if we follow the control policy μ from here on out.

Classically, a policy is optimized through repeated cyclic interactions, shown in Fig. 1.2. Given an observation, s_t, the agent will execute an action, a_t, based on its current policy μ. When a_t is applied to the environment, it will influence how the system evolves, resulting in the following state s_{t+1}. As we are interested in systems that are continuous in time, we generalize the nomenclature of the "next state" to $s_{t+\tau}$ where τ is the time to between steps. The generalized notation for "next" in the absence of time indexing is $\cdot'$ (e.g. the next state is s'). How favorably a_t impacted the environment from s_t to $s_{t+\tau}$ is then quantified by r_t. This constitutes a learning cycle tuple, $[s_t, a_t, r_t, s_{t+\tau}]$, while the repeated sequence of interactions, $[s_0, a_0, r_0, s_\tau, a_\tau, ...]$, constitutes a *trajectory*, $\mathcal{T}$. These pieces of information are used to update and improve μ. In the following, we will overview the underlying theory and core equations in RL that utilize this information to update μ.

Stochasticity and Reinforcement Learning

A few important distinctions to make at this point are related to stochasticity. Often times when we consider classical controllers we think of deterministic functions–a given input yields a consistent output response. In RL theory, the policy is often generalized as a stochastic function, $\pi(s|a)$, which describes the probability of $a_t = a$ given $s_t = s$,

$$\sum_a \pi(a|s) = 1. \tag{1.10}$$

To consolidate our nomenclatures of deterministic policy μ and our new generalized stochastic policy π, we relate these via the following assuming Gaussian probability distributions,

$$\begin{aligned} a_t &\sim \pi(\cdot|s_t) \\ a_t &= \mu(s_t) + \sigma(s_t) \odot \xi \end{aligned} \tag{1.11}$$

where $\sigma(s)$ is the standard deviation and z is a vector of noise sampled from a Gaussian ($\xi \sim \mathcal{N}(0, I)$). In this generalized form, the implemented a_t is sampled from $a \in \mathcal{A}$, based on the probability distributions set by π given s. For a deterministic policy, the probability of a_t given $s_t = s$ is effectively one and zero for all other a, i.e. $\sigma(s_t) = 0$. Note this is not to be confused with a greedy policy, which will be discussed later.

Another point of stochasticity is in the RL framework the environment is idealized to be a Markov Decision Process (MDP). Under this assumption, the environment's dynamics can be completely captured by the probability distribution p,

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1, \quad \forall s \in \mathcal{S}, a \in \mathcal{A}. \tag{1.12}$$

This means that the probabilities of s' (which we recall is the *generalized* time-index free notation of $s_{t+\tau}$) and r solely depends on s and a. In other words, the environment is Markov, where only the current state captures all necessary information to describe the future. In this thesis, we are interested in chaotic dynamical systems which are not only Markov but also deterministic. A deterministic interpretation of the above probability distribution is that given a specific s_t and a_t, there is a $s_{t+\tau}$, r_t pair with probability one, with all else having probability zero.

Returns, Value Functions, and Bellman Equations

The objective of the agent is to maximize the total cumulative reward, or more formally, we seek to obtain a π that maximizes the *expected* cumulative sum of rewards. We can define this discounted cumulative reward as the *return*, R_t,

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k\tau}, \tag{1.13}$$

where $0 \leq \gamma \leq 1$ is the discount factor. This discount factor weights rewards toward the immediate future more heavily compared to distant future rewards. The closer γ is to 1,

the more farsighted the agent becomes. Furthermore, for $\gamma < 1$, R_t has a finite value, assuming the sequence is bounded. An important theme in RL (and dynamic programming) is recursion. Successive instances of the return can be related to each other,

$$R_t = r_t + \gamma R_{t+\tau}. \tag{1.14}$$

This type of relationship is foundational to much of the theory in RL. With our definition of the return, we can now explore value functions, which underpin all RL algorithms. The *state-value function*, $V_\pi(s)$, of a state s following policy π is the following,

$$V_\pi(s) = \mathbb{E}_\pi \left[R_t | s_t = s \right], \tag{1.15}$$

where $\mathbb{E}_\pi[\cdot]$ is the expected value under the policy π. The intuitive interpretation of this function is if you were to begin at state s, how much reward would you expect to collect over the future by following the rules set by π? States with larger values are more desirable than those with lesser values. The natural extension of the state-value function is the *action-value function*, $Q_\pi(s, a)$, which is the expected return starting from s *and* implementing action a, and thereafter following π,

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[R_t | s_t = s, a_t = a \right]. \tag{1.16}$$

The intuitive interpretation of Q_π is if you were to begin at state s, and you were forced to take action a (regardless of how "good" it is), how much reward would you then expect to collect over the future following π? An s paired with a "good" a will yield a larger action-value, while an a that is detrimental to the goal will yield a lower action-value.

Naturally, the state-value function and the action-value function are related to one another,

$$V_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \cdot Q_\pi(s, a). \tag{1.17}$$

Furthermore, V_π and Q_π are also sometimes related by the *advantage function*,

$$A_\pi(s,a) = Q_\pi(s,a) - V_\pi(s). \tag{1.18}$$

The advantage function has quite a literal and intuitive meaning–how much of a quantitative advantage does taking a particular action have over the distribution imposed by π at a given state? If the action is extremely detrimental, the value of Q_π will be less than that of V_π, yielding a negative advantage value and vice-versa. The advantage function is utilized in several modern RL frameworks. Similar to the return function, value functions can also be expressed recursively. For example, by combining Eq. 1.10, Eq. 1.12, Eq. 1.13, Eq. 1.14, and Eq. 1.15 we can represent V_π recursively:

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi\left[R_t | s_t = s\right] \\ &= \mathbb{E}_\pi\left[r_t + \gamma R_{t+\tau} | s_t = s\right] \\ &= \sum_a \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\left[r + \gamma \mathbb{E}_\pi\left[R_{t+\tau} | s_{t+\tau} = s'\right]\right] \\ &= \sum_a \pi(a|s) \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\left[r + \gamma V_\pi(s')\right] \end{aligned} \tag{1.19}$$

This is known as the *Bellman Equation* for the value function, which describes the relationship between the value of a state and its successive states. The Bellman equation here describes that the value of a state is equal to the immediate reward and the discounted expected value of the next state. The Bellman equation is the foundation for how RL algorithms compute and approximate V_π. We can extend this naturally to Q_π to arrive at the Bellman equation for the action-value function:

$$Q_\pi(s,a) = \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\left[r + \gamma \sum_{a'} \pi(a'|s') Q_\pi(s', a')\right] \tag{1.20}$$

Optimal Policies, Optimality Equations

Now that we have established the connections between the policy, return, and value functions through the Bellman equations, we can now develop relationships between these functions for the *optimal policy*, π^*, which yields expected returns greater than or equal to all other policies for all states.

$$\pi^* = \arg\max_{\pi} V_\pi(s), \quad \forall s \in \mathcal{S} \tag{1.21}$$

Obtaining, estimating, or learning π^* is the goal of reinforcement learning and often this is achieved by first acquiring the *optimal state value function*, $V_*(s)$, and the *optimal action value function*, $Q_*(s, a)$, which are our value functions under π^*:

$$V_*(s) = \max_{\pi} V_\pi(s), \quad \forall s \in \mathcal{S} \tag{1.22}$$

$$Q_*(s, a) = \max_{\pi} Q_\pi(s, a), \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \tag{1.23}$$

The relationship between these two are similar to Eq. 1.17,

$$V_*(s) = \max_{a} Q_*(s, a). \tag{1.24}$$

From here, we can derive the *Bellman optimality equation* for the state value function,

$$\begin{aligned} V_*(s) &= \max_a Q_*(s, a) \\ &= \max_a \mathbb{E}_{\pi^*} \left[R_t | s_t = s, a_t = a \right] \\ &= \max_a \mathbb{E}_{\pi^*} \left[r_t + \gamma R_{t+\tau} | s_t = s, a_t = a \right] \\ &= \max_a \mathbb{E}_{\pi^*} \left[r_t + \gamma V_*(s_{t+\tau}) | s_t = s, a_t = a \right] \\ &= \max_a \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) \left[r + \gamma V_*(s') \right] \end{aligned} \tag{1.25}$$

which tells us that following the optimal policy, the value of a state is equal to the expected return for applying the best action to that state. The Bellman optimality equation for the action value function can be derived similarly,

$$\begin{aligned} Q_*(s,a) &= \mathbb{E}_{\pi^*}\left[r_t + \gamma \max_{a'} Q_*(s_{t+\tau}, a') | s_t = s, a_t = a\right] \\ &= \sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a)\left[r + \gamma \max_{a'} Q_*(s', a')\right] \end{aligned} \tag{1.26}$$

For finite MDPs, the Bellman optimality equations for V_* and Q_* each have a unique solution [186], where each recursive set of relationships form a system of equations (i.e. one equation for each state). These two sets can each be solved (in principle) for V_* and Q_*. The reason one would want to obtain $V_*(s)$ is it easy to determine the optimal action at any given state by performing a one-step-ahead search that maximizes Bellman optimality equation. This is possible because the one-step evaluation of V_* after an action already considers the optimal long-term return. Utilizing $Q_*(s, a)$ is even simpler, as the action, for a given state, that maximizes Q_* must be the optimal action, a^*.

$$a^* = \arg\max_a Q_*(s, a) = \mu^*(s) \tag{1.27}$$

This is extremely desirable because we can determine the best action immediately for a given state without relying on one-step-ahead searches nor do we need to know about successive states, their values, and the dynamics underlying them.

Practical considerations regarding the RL problem

Although we can define recursive equations describing optimality, analytically *solving* the underlying system of equations is entirely a different problem. For simple systems with small, finite state sets and action sets, it is possible to compute the solutions. However, for complex or continuous systems where the set of states is large or infinite, analytically solving for the optimal policy is intractable as we simply lack the computational power and memory capable of solving and storing the solutions for every possible state that exists. Tabular

methods have attempted to address this using tables with each entry corresponding to a state or state-action pair [186] (for continuous systems states and actions are discretized) but practically there simply are no tables big enough to record every possibility underlying the dynamics of the environment. Thus, the overwhelming majority of RL methods attempt to *approximate* the functions of π^*, V_*, and/or Q_* with network-based models.

1.5 Deep Reinforcement Learning

We now turn our attention to recent advances in RL made by the deep learning community. This section will briefly overview the algorithms, accomplishments, and challenges of deep RL. We will then review their application to controlling chaotic dynamical systems and fluid flows. For the reader unfamiliar with neural networks (NNs), a brief overview is provided in Sec. 1.3 and *highly* recommended.

The term "deep" RL refers to the reinforcement frameworks that utilize deep (albeit often a liberal usage of the term) NNs to approximate policies and/or value functions. These methods are divided into two general categories: *on-policy* and *off-policy*. On-policy methods characteristically require all data at the time of fitting to be generated with the *current* policy. Once the network parameters are updated, effectively updating the policy, all data must be discarded and new data must be generated. Broadly speaking, the "data" here refers to the recorded trajectories i.e. the repeated cyclic sequence of interactions between the agent and environment. Off-policy methods, in contrast, can utilize data that has been generated from *any* policy, including those from previous iterations of the policy or even data from completely unrelated policies.

In the following discussion we overview two "model-free" deep RL algorithms, deep Q-learning [136] and (vanilla) policy gradients [187], as they establish precursor framework components many of the more modern deep algorithms utilize. We will then overview the modern algorithms used in this thesis, Deep Deterministic Policy Gradients (DDPG) [119]

and Soft Actor Critic (SAC) [76], and briefly acknowledge others.

Before diving in, it is important to discuss a universal dilemma all practical RL frameworks deal with: *exploration vs exploitation*. RL algorithms seek to learn action values that are conditional on optimal behavior, but in order to find optimal actions they need to explore, which requires behaving non-optimally. There are several nonexclusive ways this issue is addressed. Some methods balance exploration and exploitation as a trade-off during learning–the extent of the policy's exploration decreases as it approaches near-optimality. Many on-policy methods implicitly achieve this via stochasticity as they slowly optimize their distributions. Others utilize two policies, one which is attempting to capture the policy (*target policy*) and another that is more exploratory that generates data (*behavior policy*). This strategy is a hallmark of off-policy methods. Finally, some algorithms explicitly introduce randomized actions to encourage state-action exploration. These are most common in deterministic off-policy methods.

1.5.1 Deep Q-Learning

We will begin our dive with Q-Learning in the form of deep Q-networks (DQN), which is the simplest deep off-policy method. Developed by Mnih [136], this method set a precedent for the potential of NN-based RL. Utilizing only a single NN, this framework was able to autonomously learn and play at a "superhuman"-level classical video games such as Video Pinball, Breakout, and Star Gunner [136]. The main idea in DQNs is to construct a NN to approximate $Q_\pi \approx Q_\pi(s, a; \theta)$ with NN parameters θ, hence the name. The input to this NN is the state observation and output is the estimated Q value for each possible action. The action that is selected for a given state is then simply the one that corresponds with the largest predicted Q value, i.e. a greedy policy. The network is fit with the following loss,

$$\mathcal{L} = \mathbb{E}\left[\left(\left(r_i + \gamma \max_{a'} Q_\pi(s', a'; \theta)\right) - Q_\pi(s, a, ; \theta)\right)^2\right]. \tag{1.28}$$

Note that this loss is the Bellman equation which we derived in Eq. 1.26. As this policy is greedy/deterministic, the algorithm needs a way to explore. The authors address this by having a small probability ϵ of choosing a random action at each step during training.

Importantly, the loss can be evaluated for any interaction cycle, and as a result the training data can simply consist of tuples of $[s_t, a_t, r_t, s_{t+\tau}]$ experiences. DQNs also utilize a "replay buffer" or "experience replay", which stores previously generated tuples in a rolling buffer. As new experiences are generated during training, they are added to the buffer. During fitting, the ensemble of tuples are sampled from this replay buffer. For improved training stability, the buffer is generally quite large to avoid over-fitting. However, it is not necessarily better or practical to store every experience so often times older experiences are slowly dequeued as training progresses.

Because the output of the NN are discrete nodes, the actions themselves must come from a finite discrete set. This is disadvantageous as it is impossible to produce actions from a continuous range. One could finely discretize the output layer, but this leads to massive NNs and poor learning. Nonetheless, this framework is able to map states from a *continuous* state space to finite discrete actions, addressing the tractability issues of value function approximation on the state-space side. The challenge of implementing continuous actions are addressed in more modern RL algorithms.

1.5.2 Policy Gradients

Policy gradients, specifically the vanilla policy gradient (PG) [187], is the simplest on-policy deep RL framework. In PGs, the goal is still the same–to obtain a policy that maximizes the long-time return $J(\pi_\theta) = \mathbb{E}_{\mathcal{T}\sim\pi_\theta}[R(\mathcal{T})]$ (recall here $\mathcal{T}$ is the trajectory sampled following policy π_θ). However, unlike deep Q-learning, PGs seek to directly optimize the policy π by gradient *ascent*,

$$\theta_{i+1} = \theta_i + \alpha \nabla_\theta J(\pi_\theta)|_{\theta_i}. \tag{1.29}$$

In other words, PGs directly optimize the policy by iteratively estimating the gradient of performance with respect to the parameters of the model. This type of approach is called *policy optimization*. Below is a brief derivation,

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}[R(\mathcal{T})] \\
&= \nabla_\theta \int_{\mathcal{T}} P(\mathcal{T}|\theta) R(\mathcal{T}) \\
&= \int_{\mathcal{T}} \nabla_\theta P(\mathcal{T}|\theta) R(\mathcal{T}) \\
&= \int_{\mathcal{T}} P(\mathcal{T}|\theta) \nabla_\theta \log P(\mathcal{T}|\theta) R(\mathcal{T}) \\
&= \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}[\nabla_\theta \log P(\mathcal{T}|\theta) R(\mathcal{T})] \\
&= \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\mathcal{T})\right] \\
&= \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}, s_{t'+\tau})\right)\right]
\end{aligned} \tag{1.30}$$

where $P(\mathcal{T}|\theta)$ is the probability of trajectory $\mathcal{T}$ occurring,

$$P(\mathcal{T}|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+\tau}|s_t, a_t) \pi_\theta(a_t|s_t), \tag{1.31}$$

and $\rho_0(s_0)$ is the distribution of initial states. This derivation utilizes several "math tricks" which include the "log-derivative" trick (the derivative of $\log(x)$ with respect to x is $1/x$),

$$P(\mathcal{T}|\theta) \nabla_\theta \log P(\mathcal{T}|\theta), \tag{1.32}$$

and that the product of a sequence of probabilities can be written as the *summation* of the log of their probabilities,

$$\log P(\mathcal{T}|\theta) = \log \rho_0(s_0) + \sum_{t=0}^{T} (\log P(s_{t+\tau}|s_t, a_t) + \log \pi_\theta(a_t|s_t)). \tag{1.33}$$

An important note is that the gradient of the environment-dependent functions are zero (i.e. $\rho_0(s_0)$, $P(s_{t+\tau}|s_t, a_t)$, $R(\mathcal{T})$) which greatly simplifies the derivation. This derivation formulates the gradient as an expected value, which means it can be estimated from data via ensemble averages. This is the foundation for the REINFORCE algorithm by Williams [213], which has an update rule of:

$$\nabla_\theta J(\pi_\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}, s_{i,t'+\tau}) \right) \right] \tag{1.34}$$

Note that unlike Q-learning, which could be trained on ensembles of unrelated tuples of $[s_t, a_t, r_t, s_{t+\tau}]$, here in PGs we must estimate the gradient over multiple *trajectories* and we cannot utilize replay buffers as we cannot optimize using off-policy data.

An aside on baselines: Many practical applications of policy gradients utilize whats known as a *baseline* [174], $b(s)$,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{t'}, a_{t'}, s_{t'+\tau}) - b(s_t) \right) \right]. \tag{1.35}$$

This modification is possible via the "Expected Grad-Log-Prob" Lemma, which states that the expected value of a grad-log-prob over a random variable is zero. In the context of our problem, it allows modification to the policy gradient without altering the expected value so long as the function is not dependent on a. Baselines are implemented because policy gradients easily suffer from high variance–trajectories must always be generated on policy and can vary greatly. Increasing batch sizes can help, but they also significantly reduce sample efficiency. Baselines have been empirically shown to reduce this variance in addition to faster and more stable function learning [174]. A commonly use baseline is the on-policy state value function, $V_\pi(s)$, which is parameterized by a second network with weights ϕ. This

leads to the advantage-based formulation,

$$\begin{aligned}\nabla_\theta J(\pi_\theta) &= \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(\sum_{t'=t}^{T}\gamma^{t'-t}r(s_{t'},a_{t'},s_{t'+\tau}) - V_\pi(s_t;\phi)\right)\right] \\ &= \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)\left(Q_\pi(s_t,a_t) - V_\pi(s_t;\phi)\right)\right] \\ &= \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi_\theta}\left[\sum_{t=0}^{T}\nabla_\theta \log \pi_\theta(a_t|s_t)A_\pi(s_t,a_t;\phi)\right],\end{aligned} \tag{1.36}$$

which in addition to improving variance is also more conceptually intuitive. Finally, as we have introduced a baseline value network, $V_\pi(s;\phi)$, this network is updated with a standard MSE loss,

$$\mathcal{L} = \mathop{\mathbb{E}}_{s_t,R_t\sim\pi_\theta}\left[\left(V_\pi(s_t) - R_t\right)^2\right]. \tag{1.37}$$

1.5.3 Modern deep RL Algorithms

Here we provide a review of modern deep RL algorithms with an emphasis on the Deep Deterministic Policy Gradient (DDPG) [119] and Soft-Actor Critic (SAC) [76] algorithms, as they appear in the works later discussed in this thesis. Alternative modern methods will be briefly discussed afterwards.

Deep Deterministic Policy Gradients (DDPG)

Deep Deterministic Policy Gradients (DDPG), developed by Lillicrap et al. [119], combines Q-learning with policy optimization, forming a framework that can input states from a continuous state space and output actions from a continuous range. To summarize, this algorithm simultaneously aims to learn an approximation to $Q_*(s,a;\theta_Q)$ and a deterministic optimal policy $a^* = \mu^*(s;\theta_\mu)$. Similar to DQNs, the Q function is learned using off-policy data and updated via the Bellman equation. However, unlike DQNs where the Q network inputs s and outputs the Q value for every possible action (i.e. $Q_\mu(s,a_i)$ for all a_i) the Q network in DDPG inputs both s and a and outputs only that particular $Q_\mu(s,a)$ value. In

doing so, during the evaluation of Bellman loss, Eq. 1.28, we can make the approximation $\max_a Q_\mu(s,a) \approx Q_\mu(s,\mu(s))$. This Q function is then simultaneously used to learn the policy. In this context, DDPG can be thought of as deep Q-learning for continuous action spaces.

For stability, DDPG also utilizes a replay buffer in addition to *target networks*, where *target* refers to the comparison term $r + \gamma \max_{a'} Q_\mu(s', a'; \theta_Q)$ in Eq. 1.28. However, because this loss seeks to make the $Q_\mu(s, a; \theta_Q)$ network equal to the target, the learning problem can become very unstable because both sides of the loss depend on the same parameters θ_Q. Target networks address this subtlety by being a time-delayed copy of all the networks with weights $\theta_{.,targ}$ which is slowly updated via polyak averaging,

$$\theta_{targ} \leftarrow \alpha_p \theta_{targ} + (1 + \alpha_p)\theta_{targ}, \tag{1.38}$$

where $0 < \alpha_p < 1$ is generally close to 1. Thus, the mean-squared Bellman error with target networks is,

$$\mathcal{L}(\theta_Q) = \mathbb{E}\left[\left((r_i + \gamma Q_\mu(s', \mu(a'; \theta_{\mu,targ}); \theta_{Q,targ})) - Q_\mu(s, a, ; \theta_Q)\right)^2\right]. \tag{1.39}$$

To optimize the policy network, $\mu(s; \theta_\mu)$, we simply seek the parameters θ_μ that maximize the expected Q value,

$$\max_{\theta_\mu} \mathbb{E}\left[Q_\mu(s, \mu(s; \theta_\mu); \theta_Q)\right], \tag{1.40}$$

which is a policy gradient ascent problem. Thus, μ is updated with the ascent of the following gradient,

$$\nabla_{\theta_\mu} J \approx \frac{1}{N} \sum_{i=1}^{N} \nabla_a Q_\mu(s, a; \theta_Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta_\mu} \mu(s; \theta_\mu)|_{s_i}. \tag{1.41}$$

Similar to DQNs, DDPG is a deterministic framework which requires it to address the issue of training exploration. In the original implementation of DDPG, the authors choose to add a temporally correlated noisy signal, the Orstein-Uhlenbeck noise [119], to encourage a more correlated exploration.

Soft Actor-Critic

The Soft Actor-Critic (SAC) algorithm, developed by Haarnoja et al. [76], fuses the DDPG-style (i.e. actor-critic) framework with stochastic policy optimization principles. Similar to DDPG, SAC learns a policy that maps state observations to actions from a continuous action space. However, unlike DDPG which optimizes a deterministic policy $a = \mu(s;\theta)$, SAC optimizes a stochastic policy $a \sim \pi(s;\theta)$. Both DDPG and SAC learn critic networks to estimate Q_π. SAC takes this actor-critic structure one step forward by learning two critic networks concurrently, and uses the more conservative Q_π value estimate during evaluation. This has been found to aid policies from over-correcting when the critic network overestimates values during the learning [51, 76]. Like policy gradients, the original implementation of SAC also utilizes an approximation of the state value function V_π as an additional point of consistency (an updated version of SAC without this has been developed). Finally, like DDPG, SAC also makes use of target networks (for the value network) to address the brittleness of value function learning.

The most unique feature of SAC is the incorporation of entropy regularization, which forces the stochastic agent to strike a balance between maximizing expected return and entropy i.e. randomness in the policy. For a random variable, x, with probability density $P(x)$, the entropy, $\mathcal{H}$, is described,

$$\mathcal{H}(P) = \underset{x\sim P}{\mathbb{E}} \left[-\log P(x)\right]. \tag{1.42}$$

Note that $\mathcal{H}$ decreases as the probability density tightens or as the distribution becomes "more" deterministic. By incentivizing entropy in the policy, the agent is encouraged to explore more widely which can often lead to accelerated learning. Furthermore, if there are multiple avenues or strategies that are equally optimal, the algorithm can commit to exploiting both, as committing to both strategies is more entropic than a policy that learns to only exploit a single strategy. In other words, the agent can learn multiple modes of

approximately optimal behaviors. The incorporation of entropy as a regularization term in the return modifies the RL objective from Eq. 1.21 to the following,

$$\pi^* = \arg\max_{\pi} \mathop{\mathbb{E}}_{\mathcal{T}\sim\pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(r_t + \alpha \mathcal{H}\left(\pi(\cdot|s_t)\right)\right)\right], \tag{1.43}$$

where $0 < \alpha$ is the regularization coefficient. An important *practical* comment regarding this coefficient is while the entropic term is commonly scaled in theory (e.g. above and below), practical implementations (e.g. code bases) choose to absorb this scaling into the reward instead, making the scale of the reward larger or smaller relative to the entropic term. From here on, the coefficient is omitted. Regardless, the modification propagates to our V_π and Q_π functions, which we defined in Eq. 1.15 and Eq. 1.16, respectively. Importantly, the state value and action value equations, when considering this modification, become the following,

$$\begin{aligned} V_\pi(s) &= \mathop{\mathbb{E}}_{a\sim\pi} \left[Q_\pi(s,a) + \mathcal{H}(\pi(\cdot|s))\right] \\ &= \mathop{\mathbb{E}}_{a\sim\pi} \left[Q_\pi(s,a) - \log\pi(a|s)\right] \\ &\approx \left(Q_\pi(s,\tilde{a}) - \log\pi(\tilde{a}|s)\right) \end{aligned} \tag{1.44}$$

$$\begin{aligned} Q_\pi(s,a) &= r(s,a,s') + \gamma \mathop{\mathbb{E}}_{s_{t+\tau}} \left[V_\pi(s_{t+\tau})\right] \\ &\approx r(s,a,s') + \gamma V_\pi(s_{t+\tau}) \end{aligned} \tag{1.45}$$

To learn NN function approximations of V_π, Q_π, and π, these approximate equations will be used as loss equations for $V_\pi(s;\theta_V)$ and $Q_\pi(s,a;\theta_Q)$, respectively:

$$\mathcal{L}(\theta_V) = \mathbb{E}\left[\frac{1}{2} \left(V_\pi(s;\theta_V) - \left(\min_{j=1,2} Q_\pi(s,a;\theta_{Q,j}) - \log\pi(s,a;\theta_\pi)\right)\right)^2\right], \tag{1.46}$$

$$\mathcal{L}(\theta_{Q,j}) = \mathbb{E}\left[\frac{1}{2} \left(Q_\pi(s,a;\theta_Q,j) - \left(r(s,a,s') + \gamma V_\pi(s';\theta_{V,targ})\right)\right)^2\right]. \tag{1.47}$$

Note that V_π is updated with the conservative estimate made between the two critic networks,

and the critic networks are updated with the target value network. As the policy is stochastic, a subtlety we must account for is how to parameterize the policy in a way we can compute the gradient for, i.e. in the form of Eq. 1.11,

$$a = f(z, s; \theta_\pi), \tag{1.48}$$

where z is a noise vector sampled independent of θ_π. With this formulation, we can update the policy by ascending the gradient of our objective,

$$\nabla_{\theta_\pi} J \approx \frac{1}{N} \sum_{i=1}^{N} \left[\min_{j=1,2} Q_\pi(s, f(z, s_i; \theta_\pi); \theta_{Q,j}) - \log \pi(f(z, s_i; \theta_\pi) | s_i) \right]. \tag{1.49}$$

where actions are sampled from the current iteration of the policy.

A Comment on State-of-the-Art On-Policy Algorithms

So far we have only scraped the surface of deep RL in our overview. We primarily focus on deep off-policy type algorithms as they are the algorithms used in the works of this thesis, but there are several powerful policy optimization frameworks that are worth mentioning. These methods include Asynchronous Actor-Critic Agent (A3C) [137], Proximal Policy Optimization (PPO) [173], Trusted-Region Policy Optimization (TRPO) [172].

1.5.4 Challenges of Deep RL

Deep RL, while extremely powerful, has its own set of challenges and limitations. A thorough discussion of the challenges RL experiences in the real world is detailed in Dulac-Arnold et al. [40]. But to summarize, the main challenges RL experiences in real world usage are: 1) limited or sparse samples, 2) large or unknown delays in sensing or actuating, 3) high-dimensional state-action spaces, 4) hard constraints, 5) partially observable systems, 6) multi-objective rewards 7) extremely low latency requirements 8) utilizing only data from an external policy, and 9) providing explainable policies.

Each of these challenges is its own facet of RL research and warrants its own spotlight. However, for the applications we are interested in (e.g. controlling a turbulent flow), the immediate challenges we are concerned with are challenges 1, 4, and 8 with the first as our main concern. When the systems sought to be controlled are computationally or experimentally costly (e.g. high-fidelity turbulent flow simulations), generating enough experiences from the environment to learn a sufficient policy can quickly become an intractable endeavor.

1.5.5 Model-Based Reinforcement Learning

In the previous sections, we overviewed modern "model-free" deep RL frameworks that learned purely from interactions with the environment. In the following section, we will overview an alternative, model-based RL algorithms, which utilize a model of the environment in some form factor. The purpose of the model is to simulate the environment to produce synthetic experiences. These models can be *distribution models*, which output all the probabilities of all possible state transitions, or *sample models*, which outputs a single sampled state transition. As our systems of interest are deterministic and continuous, our models will be the latter for practicality,

$$s', r = \text{model}(s, a). \tag{1.50}$$

An important concept in RL literature is *planning*, which refers to the use of a model as input to improve or produce a policy for the target environment. There are many ways models can be utilized to assist in learning the policy. Here we will outline a few notable model-based archetypes including *dyna* [186] and *world models* [75]. There are other notable frameworks such as Model-based Value Expansion [45], and Imagination-Augmented Agents (I2A) [157] but we will focus on discussing the aforementioned.

The most basic dyna-style algorithm involves fitting a model online. At each step, the real environment interaction is used to fit the policy *and* the model. Before the next step,

the model is used to additionally fit the policy with simulated experiences. These simulated experiences can be chosen uniformly from state-action sweeps, smarter prioritized sweeps, or as entire sampled simulated trajectories.

Planning can also be utilized at decision-time, which philosophically heavily overlaps with model-predictive control (MPC). For example, a model can be learned concurrently with MPC, where the MPC optimized actions serve as the policy [26]. For a given state observation, MPC optimizes over the current model for the best action and implements it. The resulting transition is used to update the model, thereby updating the solutions optimized by MPC and thus the policy.

On the other end of the spectrum, a model can be first generated and then used to solely update the policy. World models [75] utilize low-dimensional models to fully *replace* the environment. The models are trained on trajectories of the original environment experiencing a random policy. The objective is to capture the low-dimensional underlying dynamics of the system and their response to control perturbations accurately enough for an algorithm to learn and exploit. This avenue is quite attractive for RL applications for high-dimensional, computationally (or experimentally) expensive environments as it front-loads the data generation step to a one-time fixed-cost. In the limit of slow or costly data-generation, this can be the most time-efficient way of estimating a policy.

Importantly, many of the working concepts of world models thematically overlap with concepts of our dynamical systems of interest–primarily the assumption that beneath the high-dimensional ambient space lie a low-dimensional yet equally information rich representation. Intuitively, we can apply our dynamical systems knowledge of manifolds to make data-driven (world) models of our system in the lower-dimensional *manifold* representation rather than the costly high-dimensional ambient representation–a schematic of this is presented in Fig. 1.3.

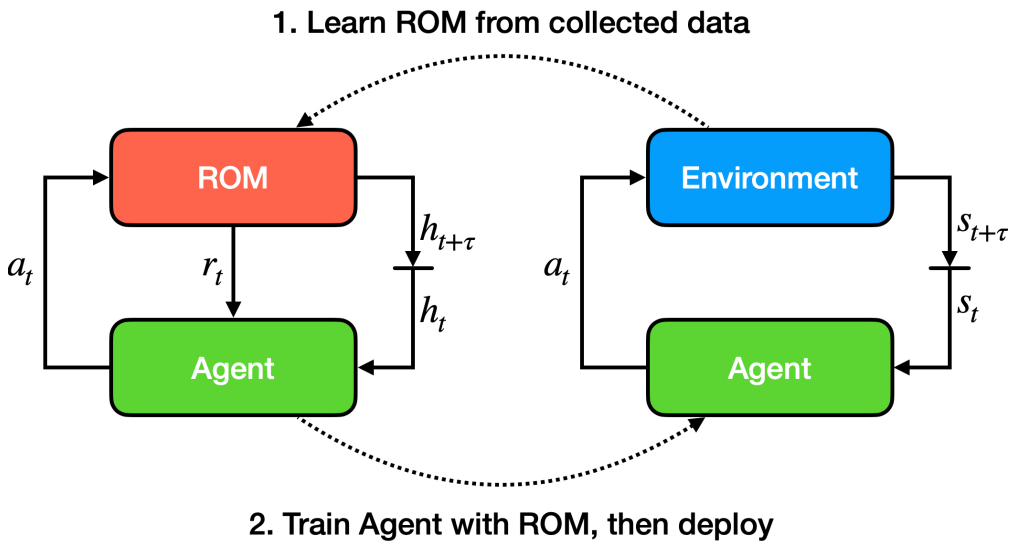

Figure 1.3: Model-based reinforcement learning cycle which uses a reduced-order model (ROM) learned from data as a surrogate environment to accelerate training.

1.6 Representation Learning for Data Dimensionality Reduction

In the following section we will depart from our overview of reinforcement learning and discuss another "RL" known as *representation learning* (or sometimes learning representations). Representation learning is a branch of artificial intelligence and machine learning concerned with unsupervised learning of a *representation function* from raw data that is useful for downstream learning tasks. Ideally, the representation function captures and isolates characteristics that are fundamental to the dataset, such as symmetries, geometric distances, energies, etc., to provide features or inputs that are richer in information and provide greater discriminative power (i.e. a more separable representation). The intuition is that these transformations of the data make downstream tasks more conducive. Representation learning is extremely powerful; it unlocks a pathway to reducing unneeded complexity in downstream models, reducing the amount of data and compute needed to train said models, or even giving insight into the underlying structure of the data.

Representational learning, like reinforcement learning, is its own immense field of study

with an exponentially growing community and broad interests. In this review, we will only be able to focus on a fraction of the frameworks to contextualize some concepts and tools that are utilized in this thesis. To avoid confusion in this thesis and to maintain consistency in nomenclature described thus far, we will generically define ambient data snapshots as u (whereas it is more commonly defined in the learning community as x). These snapshots are collected or observed from trajectories of our target system. We will primarily focus on representation learning methods that achieve dimensionality reduction. In otherwords, we are interested in utilizing frameworks that can compress information into fewer degrees of freedom while avoiding information loss for our mapping functions and world models.

1.6.1 PCA, Nonlinear Embeddings, Contrastive Learning

We first acknowledge the ubiquitous linear method Principle Component Analysis (PCA), which is also known as Proper Orthogonal Decomposition and Karhunen-Loève. PCA seeks a linear transformation that projects the data onto an orthogonal coordinate system organized by variance. Intuitively, one can imagine this as fitting an n-dimensional ellipsoid onto the data, where each axis corresponds to a principal component–the larger the axis, the larger the variance of the data in that direction. The computation behind PCA is simple–only requiring one to compute the singular value decomposition of the mean centered data matrix, $X = [u_1 - \bar{u}, ..., u_l - \bar{u}]$, which has l snapshots and mean $\bar{u}$. The left singular vectors, U, provide the orthogonal basis while the singular values, S, yield the corresponding variance for each respective direction of the basis. Dimensionality reduction via PCA is accomplished by projecting u onto the truncated leading singular vector basis, $\hat{U}$, which are often chosen by trends in the singular value spectra. The reconstruction of the ambient representation of the snapshot, $\tilde{u}$, can be recovered by projection onto $\hat{U}^\top$. PCA alone however is often insufficient at achieving dramatic dimensionality reduction for our systems of interest.

Several nonlinear dimensionality reduction methods have become ubiquitous in the data-visualization community. These embedding methods include isometric feature mapping

(ISOMAP) [191], Locally Linear Embedding (LLE) [166], t-distributed stochastic neighbor embedding (tSNE) [193], Laplacian Eigenmaps [5], and Uniform Manifold Approximation and Projection (UMAP) [134] to name a few. At a high level, many of these methods are similar in that they aim to find a lower-dimensional representation that captures some geometric distance or distribution relationship found in the ambient data. Without modifications, most of these methods are only able to project ambient data into the lower-dimensional representation–they do not provide a way to recover the ambient representation given the compressed representation. Furthermore, many of these methods are unable to project new out-of-sample data and scale poorly with high-dimensional data. These qualities can be disadvantageous for dynamics modeling when new unseen initial conditions are introduced.

We now turn our attention to nonlinear network-based representation learning methods such as contrastive learning and autoencoders. We will first discuss contrastive learning here and dedicate the following section to autoencoders as they are what we utilize in this thesis. Contrastive learning aims to find a latent representation that is similar for inputs that are the "same"–you can imagine two augmented versions of the same image of a cat. A powerful method that utilizes this idea is SimCLR [18], which utilizes two networks. The first network maps or encodes each of the paired inputs to a latent representation. A second dense network then maps that latent representation to a vector representation, z, via a projection head. The vector representations are then compared with a contrastive loss. The idea of the contrastive loss is that a *positive* sample pair (e.g. two augmented versions of the sample image) should be closely aligned or similar in the *vector* representation while a *negative* sample pair (e.g. another unrelated image from the training pool) should not. Given a set of inputs that includes a positive pair of augmented examples $\tilde{u}_i$ and $\tilde{u}_j$, the contrastive

loss function aims to identify $\tilde{u}_j$ in the set for a given $\tilde{u}_i$:

$$\begin{aligned} \mathcal{L} =& \frac{1}{2N}\sum_{k=1}^{N}[l(2k-1,2k)+l(2k,2k-1)] \\ l(i,j) =& -\log\frac{\exp(s_{i,j}/\tau)}{\sum_{k=1}^{2N}\mathbb{1}_{[k\neq 1]}\exp(s_{i,k}/\tau)}, \quad s_{i,j}=z_i^\top z_j/(\|z_i\|\|z_j\|) \end{aligned} \tag{1.51}$$

Here, N is the batch size, k is the snapshot index, $s_{i,j}$ is the pairwise similarity, and τ is a constant. The reason why a projection head is used to transform the latent representation to the vector representation for the final contrastive comparison rather than just using the latent representation is based on empirical performance [18]. Successful contrastive learning generally requires heavy data augmentation, large batch sizes, and strong negative data sets. The first and last requirement is generally not a difficulty for image-based problems, as most images are easily augmented and labeled. However, for dynamical systems data, what type of data augmentation and labels the data should have is not as obvious. Furthermore, if one were interested in recovering the ambient representation from the latent vector, this would need to be addressed in a separate learning task. Several other notable contrastive frameworks in addition to SimCLR include Barlow Twins [216] and and Bootstrap Your Own Latent (BYOL, which doesn't need negative samples!) [71].

1.6.2 Autoencoders and More Autoencoders

Autoencoders (AE) are unsupervised representation learning neural networks composed of two neural networks: an encoder network, $z = \mathcal{E}(u;\theta_\mathcal{E})$ and a decoder network, $\tilde{u} = \mathcal{D}(z;\theta_\mathcal{D})$. Here $z \in \mathbb{R}^{d_z}$ is the latent representation and $u \in \mathbb{R}^{d_u}$ is the target data. The task of these networks is to satisfy that their composition yields the identity function (i.e. the autoassociation task), $\tilde{u} = \mathcal{D}(\mathcal{E}(u;\theta_\mathcal{E});\theta_\mathcal{D})$, in addition to any constraints imposed. These constraints largely depend on the AE framework and are generally motivated in constraining or regularizing the intermediate latent representation z to have specific useful properties such as dimensionality, distributional densities, isotropy, etc. In this regard, AEs are powerful and

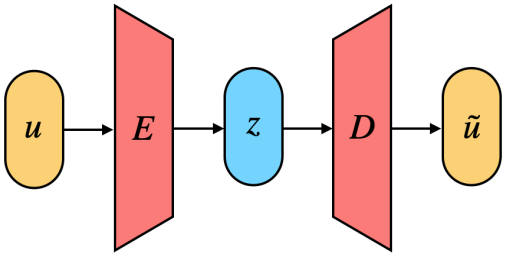

Figure 1.4: A standard autoencoder framework.

customizable unsupervised tools that can be used to obtain or "discover" more useful representations of data. A standard autoencoder is shown in Fig. 1.4 and a more thorough discussion on these regularizations is provided later in this section.

As alluded to earlier, a particularly powerful application of AEs is data dimensionality reduction. The most straight forward way this can be accomplished is by simply constructing an undercomplete AE, where $d_z < d_u$. As the architecture aims to learn the identity, the network is forced to learn a lower-dimensional representation that is informationally lossless to the original data u while still being able to reconstruct the original input data. This is an attractive method as the practitioner not only obtains an approximately lossless lower-dimensional representation, the trained networks $\mathcal{E}(u)$ and $\mathcal{D}(z)$ also provide mapping functions to and from the original representation and the new latent representation. This is especially useful for two reasons: the first is these architectures are easily incorporated into larger tasks and frameworks. Second, these allow mapping of *new* data to and from the latent space, which is not immediately possible for many other methods.

Like all other network-based approaches, many variants have been developed by (tired) graduate students and eager PIs with seemingly limitless machine learning funding. A far from exhaustive list of variants include Variational AEs (VAEs) [103], Wasserstein AEs (WAEs) [192], Contractive AEs (CAEs) [165], Sparse AE (SAEs) [142], and implicit rank minimizing AEs (IRMAEs) [97]. Autoencoders, like RL algorithms, can be formulated as probabilistic or deterministic models.

We will first review probabilistic frameworks–these include VAEs and WAEs. Compared

to vanilla undercomplete AEs, VAEs have two unique characteristics. The first is the model is based on ideas of Bayesian modeling and aims to learn a probabilistic latent space parameterized by priors and noise distributions. This latent space is practically implemented using the same "reparameterization trick" as the RL algorithm SAC i.e. $z \sim \mu + \sigma \odot \xi$, where $\sigma = h(u)$ and $\mu = g(u)$. The second unique characteristic is in addition to the autoassociation task, there is an additional goal of maximizing the marginal log-likelihood of the training data (or minimizing the Kullback-Leibler divergence, $D_{\mathrm{KL}}(P\|Q)$). For priors approximated with Gaussians (as is often the practical case) the loss becomes,

$$\mathcal{L} = c\mathcal{L}_{MSE}(u) + D_{\mathrm{KL}}(\mathcal{N}(g(u), h(u))\|\mathcal{N}(0, 1)) \tag{1.52}$$

Many renditions of the VAE have been developed over the years: β-VAEs [80], Information maximizing VAE (InfoVAE) [222], Hierarchically Factorized VAEs (HFVAE) [43], and Variational-Graph AE (VGAE) [104] to name a few. Like VAEs, WAEs also aim to parameterize its latent space with distributions and have an additional regularization loss term. However, unlike VAEs, WAEs aim to minimize the Wasserstein distance between the model distribution and target distribution (for those familiar, the 1-Wassertein distance is also known as the "Earth Moving Distance" [167]). Another point of view is the WAE aims to minimize the optimal transport distance between two probabilities for a cost function c.

We now discuss deterministic AEs, which include CAEs, SAEs, and IRMAEs to name a few. CAEs utilize a regularization term imposed on the Jacobian of the encoder network with respect to the inputs, $J_{\mathcal{E}}(u)$. The intuition is by minimizing the squared Jacobian norm, the latent representations of the input will be more similar to each other and less sensitive to perturbations.

$$\mathcal{L} = \mathcal{L}_{MSE}(u) + \lambda\|J_{\mathcal{E}}(u)\|_F^2 \tag{1.53}$$

An interesting connection is for a linear encoder, the squared Frobenius norm of the CAE penalty is equivalent to L2 weight-decay regularization. In a similar vein, SAEs utilize a

regularization term that penalizes weights with the L1 norm, which drives "excess" weights to zero. The intuition is reducing weight redundancies in the network will lead to sparser activations and thus more unique latent features. SAEs can also be formulated as a KL-divergence based regularization loss–driving each layer's j-th neuron to have an activation distribution $\rho_j^{(l)}$ equal to a target distribution mean, ρ.

$$\begin{aligned}\mathcal{L} =& \mathcal{L}_{MSE}(u) + \lambda \sum_i |w_i|^2 \quad \text{or} \\ \mathcal{L} =& \mathcal{L}_{MSE}(u) + \beta \sum_l \sum_j D_{\mathrm{KL}}(\rho \| \hat{\rho}_j^{(l)})\end{aligned} \tag{1.54}$$

Here l is the layer index, β and λ are tuning parameters. Finally, IRMAE, which unlike the above methods that utilize an explicit regularization term, utilizes *implicit regularization* [2]. Implicit regularization is a phenomenon observed in the optimization of sequential linear layers via stochastic gradient descent having a tendency to learn low-rank representations. IRMAE utilizes this in the architecture to automatically drive the latent space towards a low-rank space. This idea is the foundation of the the work presented in Chapter 5.

1.7 Data-Driven Forecasting Frameworks

Several works in this thesis related to reinforcement learning and representation learning use or are motivated by data-driven forecasting models. Here we (very) briefly give mention to some of the available data-driven modeling frameworks and motivations.

The least complex data-driven frameworks are linear models–these include dynamic mode decomposition and its variants (DMDc, etc) [150, 170]. While requiring only solving a linear problem to find a linear dynamics operator, these methods can only capture decaying or regular dynamics, which is insufficient for the systems we are interested in. Koopman-based methods [146], which also revolve around finding a linear operator, fundamentally assume an infinitedimensional operator, which for the purposes of model reduction and implementation,

are unfavorable and impractical. Other regression methods, such as Sparse Identification of Nonlinear Dynamical Systems (SINDy) [11] and its variants, make use of a library of candidate functions and attempt to sparsely fit coefficient values to each of them from RHS data.

Network-based forecasting models are diverse in complexity and formulation. Simple feed-forward networks have been shown to be capable of forecasting complex chaotic dynamics [122]. A subclass class of networks, known as recurrent networks, are designed for handling sequence-based tasks such as time-series forecasting and natural language processing. This type of network utilizes some form of internal historical memory of previous inputs to assist in computing the output and have been shown to be promising performance wise. These include reservoir computers, such as Echo State Networks [132] and Liquid-State Machines [132], which have an extremely high-dimensional non-trainable nonlinear internal state that is dependent on previous inputs. Other recurrent frameworks include gated networks and long-short-term-memory (LSTM) networks [82], which have an explicit internalized memory representation that is repeatedly passed through the network.

Notably all the above recurrent methods rely on a history of inputs, making them non-Markov by construction. Our dynamical systems of interest are Markovian, and while it is by no means a *sin* to model them via non-Markov methods, the need for warm-up sequences is non-ideal and a history dependent internal state greatly obfuscates the dynamics and interpretability.

In this thesis we have chosen to use Neural ordinary differential equations (Neural ODEs) [17] for forecasting tasks. Neural ODEs are neural networks that aim to model the RHS of the sequential data they are given, rather than learn the discrete-time map from one snapshot to another.

$$\frac{du}{dt} = F(u(t); \theta) \tag{1.55}$$

This is ideal in that a Neural ODE can be paired with a choice numerical integration scheme and forecast for arbitrary time horizons. Nearly every method described above *cannot* do

this as they take the form of a discrete time-stepper,

$$u(t+\tau) = F(u(t);\theta) \tag{1.56}$$

Finally, as all forecasting tasks described in this thesis pertain to physical systems, modeling the dynamics as a set of ODEs is the most natural formulation. For dynamical systems modeling, our implementation of Neural ODEs will follow closely to Linot and Graham [123], in which the dynamics are evolved on a discovered manifold.

1.8 Outline of this work

The main objective in this thesis is to develop data-driven frameworks that combine ideas from dynamical systems theory, learning representations, and reinforcement learning to efficiently find control strategies and representations for complex dynamical systems such as turbulent flows. These systems exhibit the challenges of spatiotemporal chaotic dynamics, high-dimensionality, and compute-costly data generation.

The remainder of this thesis is divided into five chapters, each addressing aspects of these challenges. In Chapter 2, we demonstrate with the Kuramoto-Sivashinsky equation (KSE) that the representation in the RL problem is important and introduce a framework that reduces the symmetry complexity of the state representation, substantially boosting data efficiency and guaranteeing controller equivariance. In Chapter 3, we tackle the challenge of limited data generation, which is commonly the bottle-neck for practical applications of RL. Here we develop a model-based RL framework we call Data-driven Manifold Dynamics for RL (DManD-RL) and demonstrate its ability again with the KSE. In Chapter 4 we apply DManD-RL to a turbulent plane Couette flow and achieve state-of-the-art control performance compared to existing methods while avoiding the data bottle-neck problem.

Motivated by the open challenges remaining in DManD modeling and reduced-order modeling in general, in Chapter 5, we shift our focus purely onto representation learning and develop a new data-driven framework that automatically 1) estimates the underlying dimensionality of the manifold the data lives on, 2) provides a working orthogonal manifold coordinate system, and 3) provides the mapping functions between the ambient representation and manifold representation. Finally, in Chapter 6 we summarize our key findings in this thesis and pose interesting and promising future directions of research.

2

Symmetry reduction for deep reinforcement learning [1]

Deep reinforcement learning (RL) is a data-driven, model-free method capable of discovering complex control strategies for macroscopic objectives in high-dimensional systems, making its application towards flow control promising. Many systems of flow control interest possess symmetries that, when neglected, can significantly inhibit the learning and performance of a naive deep RL approach. Using a test-bed consisting of the Kuramoto-Sivashinsky Equation (KSE), equally spaced actuators, and a goal of minimizing dissipation and power cost, we demonstrate that by moving the deep RL problem to a symmetry-reduced space, we can alleviate limitations inherent in the naive application of deep RL. We demonstrate that symmetry-reduced deep RL yields improved data efficiency as well as improved control policy efficacy compared to policies found by naive deep RL. Interestingly, the policy learned by the symmetry aware control agent drives the system toward an equilibrium state of the forced KSE that is connected by continuation to an equilibrium of the unforced KSE, despite having been given no explicit information regarding its existence. I.e., to achieve its goal, the RL algorithm discovers and stabilizes an equilibrium state of the system. Finally, we

[1] The text of this chapter is adapted from the publication by K. Zeng and M. D. Graham *Physical Review E.*, 104, 2021

demonstrate that the symmetry-reduced control policy is robust to observation and actuation signal noise, as well as to system parameters it has not observed before.

2.1 Introduction

The recent explosive growth in machine learning research has led to a large set of data-driven algorithms that map inputs to outputs by learning patterns and building inferences from the data without the need to hardcode explicit instructions. A subset of these methods are called semi-supervised learning algorithms, which learn under partial supervision through feedback from the environment. This subset is dominated by deep reinforcement learning (RL) algorithms, which, with the aid of neural networks, are particularly well-suited for tackling complex control problems with elusive optimal policies. In the past few years, deep RL has garnered the spotlight by solving complex, high-dimensional control problems and defeating the best human players in the world in games such as Go [177] and DOTA II [144], which were once thought to be too high-dimensional to feasibly solve.

Using a model system, the Kuramoto-Sivashinsky equation (KSE), that has chaotic dynamics as well as continuous and discrete symmetries analogous to those found in wall turbulence, the present work takes a step toward application of deep RL to control of spatiotemporally complex fluid flow problems, with the ultimate aim being to reduce energy losses in turbulent flows.

Deep RL offers a potential avenue for discovering active flow control policies for several reasons. Designing a complex active flow controller via analytical means is, in general, intractable. Given an array of sensor readings and actuators, no obvious strategy exists to analytically develop a concerted control scheme between the two sets [41]. Furthermore, the nonlinear complexity and high dimensionality of turbulent flows render real-time predictive simulations of potential possible actuations impractical. The bulk of existing active flow control policies are relatively simple, relying on oscillatory or constant actuation [171]. These

open-loop control policies are suboptimal in that they do not leverage the full potential of their action space. Model-based approaches can also face difficulties. As noted, in many settings solving the governing equations is too slow for any prediction-based method to be practical. Reduced-order models aimed to expedite the modeling process face difficulties in accurately modeling the non-stationary dynamics caused by the introduction of control, which can lead to unwanted behavior when far from the target state [6]. In fact, a well described model of the system may not always be readily available.

Finally, deep RL offers the ability to discover control strategies for macroscopic goals, such as minimizing drag over the entire system, as opposed to traditional control methods that focus on microscopic goals, such as suppressing certain vortex motions. Indeed an outstanding challenge in flow control is the identification of ideal control targets achievable in specific flow problems [6].

Although current deep RL methods do not provide explicit performance guarantees, they may discover non-trivial novel control strategies that when paired with dynamical insight can serve as guides for the development of more robust novel controls. In this regard, Deep RL can also be viewed as a control strategy discovery tool in addition to a data-driven controller.

Although we are ultimately interested in controlling the drag in wall-bounded turbulent flows, the application of deep RL toward fluid dynamics and spatiotemporal chaotic systems in general still remains in its nascent stages, with a handful of advancements sprouting from various niche domains. Deep RL has been utilized to control simple chaotic dynamical systems such as the Lorenz system [73, 195]. Recently, [12] demonstrated the deep RL control of the Kuramoto-Sivashinsky equation by directing the flow from one fixed point of the system to another with a series of artificial jets. Other applications of RL involve learning the collective motion of fish [53, 196], maximizing the range of robotic gliders [163], and optimizing the motion of microswimmers [29]. With regard to fluid flow control, two recent works have explored the application of RL in two-dimensional simulations of fluid flowing over bluff bodies [73], [156]. Using data from velocity sensors, these algorithms learned

control policies to reduce skin-friction drag over the bluff body by actuating jets located on the cylinders. The flows in these studies however, were performed at laminar Reynolds numbers, where the dynamics are simple and low-dimensional. Recently, [44] demonstrated the viability of deep RL to learn flow control strategies experimentally. In this work, the algorithm used data from a series of towing experiments to learn an efficient control strategy for spinning a pair cylinders downstream of a larger cylinder to reduce the drag on the entire assembly. Although promising, many of these approaches considered flow problems that exhibit low-dimensional dynamics, lack the rich system symmetries found in wall-bounded turbulent flows such as translational and reflection symmetries, and do not aim to explicitly control the time-averaged energy dissipation rate. Furthermore, these works do not focus on understanding dynamically the learned controlled strategies.

Many systems of interest for flow control have symmetries. Deep learning approaches that respect these symmetries automatically rather than learn to approximate them from data are likely to have superior performance, and within the deep-learning community is a growing body of work demonstrating the importance of incorporating symmetries of the learning domain into the deep neural-network (NN) models. For example, it has been observed by [113] that the state-of-the-art AlexNet NN image classifier [106] spontaneously learns redundant internal representations that are equivariant to flips, scalings, and rotations when trained on ImageNet data. Other works, described later in this section, have found that directly incorporating system symmetries can yield improved learning and performance results. As many flow geometries of interest possess a range of system symmetries, it is natural to incorporate these symmetries into the deep RL model, which to our knowledge, has not been demonstrated in deep RL flow control. Because the state of many flow systems can appear in a number of symmetric orientations, it is in our interest to ensure that these dynamically equivalent states are mapped to dynamically equivalent actions for dynamical and performance consistency. Fundamentally, this implies that we seek deep models that are functionally invariant/equivariant to the state-action symmetries of the system.

Typically, for a feedforward NN to obtain an invariant/equivariant functional form, it will need to implicitly learn weight-sharing constraints [160, 176]. It is generally accepted that these invariances can be learned given sufficient training data [160] and capacity [33]. However, a consequence of symmetry preserving weight constraints is the substantial decrease in the number of effective free parameters [160, 168]. This lowers the overall network capacity, which means that for an arbitrary feedforward NN, one will need larger networks and by extension more training data and computing time to obtain desired performance and functional properties. This exacerbates an existing challenge in deep RL algorithms in that they can be expensive in terms of training data needs. For perspective, some of the most impressive successes, such as OpenAI 5, the deep RL model that defeated the best professional teams in the world in the game DOTA II, required 10 months of 770 Petaflops/s per day of training [144].

To ensure that the invariances/equivariances of the domain are respected in the learning task, there are primarily three solution types. The first solution type, data augmentation, is the simplest. This approach augments the training data to include additional symmetric permutations of the original training data with the goal of pressuring the model to implicitly learn equivariant representations [34, 106]. However, this method does not guarantee that the model will generalize, nor does it address the issue in a principled method.

The second solution type is to hard-code the symmetries into the network architecture itself. Some now ubiquitous architectures, such as convolutional NNs and recurrent NNs, have demonstrated success in improving performance by accounting for translational symmetries. However, for systems with complex or collections of symmetry groups, this hard-coding method requires carefully tailoring proper weight-constraints, designing non-dense connections, or incorporating new novel NN architectures [101, 160, 168, 211].

The third solution type is accounting for system symmetries by applying symmetry transformations to the input prior to the network or to its encoding features [35, 91, 121]. [121] demonstrated for learning models of systems with symmetry/invariance properties, such as

turbulence and crystal elasticity, these models perform better when invariance properties are embedded into the training features compared to when training features were synthetically augmented with additional symmetric data.

In the present work, we opt for the third solution type, building the symmetries explicitly around the NN model, in favor of simplicity while still obtaining explicit symmetric properties. Although there is a growing number of works seeking to ensure invariance by hard-coding novel architectures, there is yet to be a general method of applying these into arbitrary concerted network designs (e.g. how does one handle networks that feed into each other or have multiple input-types such as Actor-Critic networks?). We will demonstrate that for the control task of minimizing system dissipation and power cost for the Kuramoto-Sivashinsky equation in a parameter regime exhibiting chaotic dynamics, symmetry-reduced deep RL yields improved data efficiency, control policy efficacy, and dynamically consistent state-action mappings compared to naive deep RL. We further observe that the symmetry-reduced control policy learns to discover and target a forced equilibrium, related to a known equilibrium of the system, that exhibits low dissipation and power input cost, despite having been given no explicit information of its existence.

The remainder of this chapter is divided into the following: In Section 3.2 we introduce the Kuramoto-Sivashinsky equation and the control task, as well as providing a brief review of deep RL and a discussion of the implications of the symmetries of the state-action space on the learning problem. We then conclude this section with an outline of our method of reducing the symmetry of the deep RL problem. In Section 3.3 we compare the performance of our symmetry-reduced deep RL to naive deep RL approaches, investigate the learned control strategy through a dynamical systems lens, and probe the robustness of the policy. Finally, we summarize our results in Section 3.4 and provide a discussion of the extension of this work towards more realistic problems.

2.2 Formulation

2.2.1 The Kuramoto-Sivashinsky Equation and Controls

The Kuramoto Sivashinsky Equation (KSE) is given by,

$$u_t = -uu_x - u_{xx} - u_{xxxx} + f(x,t). \tag{2.1}$$

Here f is a spatio-temporal forcing term that will be used for control actuation. We consider the KSE in a domain of length $L = 22$ with periodic boundary conditions as this system has been extensively studied and exhibits analogous symmetries to flow systems of interest. The uncontrolled KSE, $f = 0$, exhibits rich dynamics and spatio-temporal chaos, which has made it a common toy problem and proxy system for the Navier-Stokes Equations. The equation is time evolved with a time step of $\Delta t = 0.05$ using the same numerical method and code as [12] with a third-order semi-implicit Runge-Kutta scheme, which evolves the linear second and fourth order terms with an implicit scheme and the nonlinear convective and forcing terms with an explicit scheme. Spatial discretization is performed with Fourier collocation on a mesh of 64 points. We primarily consider the domain size $L = 22$.

Importantly, the KSE possesses translational and reflection symmetries, which are also present in higher dimensional fluid systems of fluid control interest. Due to the periodic boundary conditions, the KSE can be naturally expressed in terms of Fourier modes,

$$u(x,t) = \sum_k F_k \exp\left(\frac{i2\pi kx}{L}\right). \tag{2.2}$$

The real-valued Fourier state space vector of the system can be described as the following,

$$F = [b_0, c_0, b_1, c_1, \dots], \tag{2.3}$$

where $F_k = b_k + ic_k$. The dynamics of the unforced KSE with periodic boundary conditions

are equivariant under translations: i.e. if $u(x,t)$ is a solution, then so is

$$u(x+\delta x,t) \equiv T_{\delta x}u(x,t) \tag{2.4}$$

for any spatial shift δx [13]. In Fourier space, for an arbitrary state F, its translationally symmetric state differing by a phase angle of θ can be described by the following operator,

$$\tau\left(\theta, F_k\right) = \exp(-ik\theta)F_k. \tag{2.5}$$

Here the phase angle and spatial shift is related by $\delta x = L\theta/2\pi$. The KSE also has no preferred "drift" direction. That is, there is a reflection symmetry across $x = L/2$ such that if $u(x,t)$ is a solution, then so is

$$-u(L-x,t) \equiv \Sigma u(x,t). \tag{2.6}$$

Note the sign change in both position and amplitude. Indeed, because of the translation symmetry and periodicity, one can reflect across any value of x. We say that any two states that are related by the symmetry operations $T_{\delta x}$ and/or Σ are "dynamically equivalent". In Fourier space, reflection symmetric states are related by a complex conjugation followed by negation, resulting in a sign change in the real component b_k, yielding the operator,

$$\sigma(F) = [-b_0, c_0, -b_1, c_1, \dots] \tag{2.7}$$

[13]. For a flow system with no preferred drift direction or spatial localization, it is natural to choose identical and uniformly spaced actuators for control. Spatially localized control is implemented in the KSE with $N = 4$ equally spaced Gaussian jets located at

$X \in \{0, L/4, 2L/4, 3L/4\}$ as done in [12],

$$f(x,t) = \sum_{i=1}^{4} \frac{a(t)_i}{\sqrt{2\pi\sigma_s}} \exp\left(-\frac{(x - X_i)^2}{2\sigma_s^2}\right). \tag{2.8}$$

To serve as an analogue to energy-saving flow control problems, we are interested in the minimization of the integral quantities of dissipation and total power input (required to power the system and jets) of the KSE system, which are described by $D = \langle u_{xx}^2 \rangle$ and $P_f = \langle u_x^2 \rangle + \langle uf \rangle$, respectively. Here $\langle \cdot \rangle$ is the spatial average. The presence of actuators at fixed positions in the domain modifies the symmetries of the system; we describe these changes in Section 2.2.3.

2.2.2 Deep Reinforcement Learning

Reinforcement learning is a model-free, data-driven, method to learn the mapping function between an observed state, s_t, of the environment, and the action, a_t, that maximizes the cumulative reward, R_t, by experiencing the consequences of these state-action pairs. The basic RL process is cyclic: at time, t, the agent samples the state, s_t, of the environment and, in Markovian fashion, outputs an action, a_t, which belongs to a prescribed range of actions. This action is applied to the environment for a duration of $T = 0.25$ and the environment is evolved forward in time to state, s_{t+1}. We note here the subscript s_{t+1} is equivalent to s_{t+T}, but we maintain the s_{t+1} nomenclature for consistency with RL literature. How desirably the environment evolved from s_t to s_{t+1} under the influence of action a_t is then quantified by the scalar reward, r_t, and the process repeats. The cumulative reward, R_t, is the sum of discounted individual reward returns of state-action pairs,

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots + \gamma^{n-1} r_n. \tag{2.9}$$

The discount factor, γ, is chosen to be $0 < \gamma < 1$, as events further into the future are more uncertain than those nearer the current instant. Here the instantaneous reward, r_t,

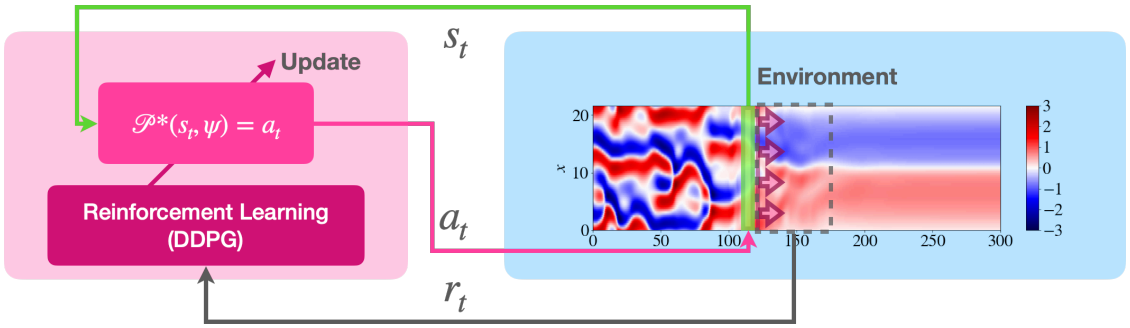

Figure 2.1: Graphical schematic of the KSE flow control problem using deep RL.

computed for each observed state-action pair, was chosen to achieve our aim of minimizing the energy dissipation rate,

$$r_t = -\overline{(D + P_f)}, \tag{2.10}$$

where $\bar{\cdot}$ is the time average over the duration of an actuation time interval of T.

In our work the environment is the KSE, the state observation is the state of the KSE, $s_t = u(t)$, and the action output is the control signal to the Gaussian jets, $a_t = a$. An arbitrary state-action mapping function is called the policy function, $\mathscr{P}(s_t) = a_t$, whereas the optimal policy that maximizes reward is denoted as $\mathscr{P}^*$. The $\mathscr{P}^*$ learning problem is illustrated in Fig. 2.1, which shows the mapping between s_t and a_t to maximize R_t through principles of RL which will be described later this section.

In this work we use the Deep Deterministic Policy Gradient (DDPG) algorithm [119], which takes on an Actor-Critic structure and serves as the baseline deep RL algorithm from which we will introduce symmetry-reducing modifications later on. The DDPG algorithm aims to approximate two key functions with NNs: the aforementioned optimal policy, $\mathscr{P}^*$, and the optimal state-action value function, $Q^*(s, a)$. In order to understand how to learn these two functions, it is necessary to understand the Q function, which quantifies the expected cumulative reward when action a_t is performed on state s_t given the current policy $\mathscr{P}$,

$$Q(s, a) = \mathbb{E}[R_t | s_t = s, a_t = a, \mathscr{P}]. \tag{2.11}$$

We seek the policy $\mathscr{P}^*$ that yields the largest state-action value $Q^*(s,a)$. I.e.:

$$Q^*(s,a) = \max_{\mathscr{P}} \mathbb{E}[R_t | s_t = s, a_t = a, \mathscr{P}]. \tag{2.12}$$

Importantly, $Q^*(s,a)$ obeys the Bellman Equation [136]:

$$Q^*(s_t,a_t) = r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1}). \tag{2.13}$$

Obtaining Q^* by explicitly evaluating all possible state-action pairs in a continuous state-action space is intractable. DDPG resolves this difficulty by utilizing NNs to approximate the Q^* function and the optimal policy $\mathscr{P}^*(s)$, which are also known as the "Critic" and "Actor" networks, respectively [119]. The Critic and Actor networks are parameterized by weights ϕ and ψ, respectively,

$$Q^*(s,a) \approx Q(s,a,\phi), \tag{2.14}$$

$$\mathscr{P}^*(s) \approx \mathscr{P}(s,\psi). \tag{2.15}$$

The Actor network is generically referred to as the "agent" in this method. Shown in Fig. 2.2 is a schematic of the Actor-Critic learning cycle. During training, the Actor network attempts to map the state observation to the optimal action. The output action along with the state observation are then passed to the Critic network, which attempts to estimate the Q-value of the state-action pair. Note that once training is complete, the Critic network may be discarded and closed-loop control is performed between the Actor network and the environment only.

During training the weights of the Critic network are updated with the following loss function,

$$L_i(\phi_i) = [(r(s_t,a_t) + \gamma Q(s_{t+1}, a_{t+1}, \phi_i)) - Q(s_t,a_t,\phi_i)]^2, \tag{2.16}$$

which will be minimized when the Bellman Equation, Eq. 2.13, is satisfied, signaling that the

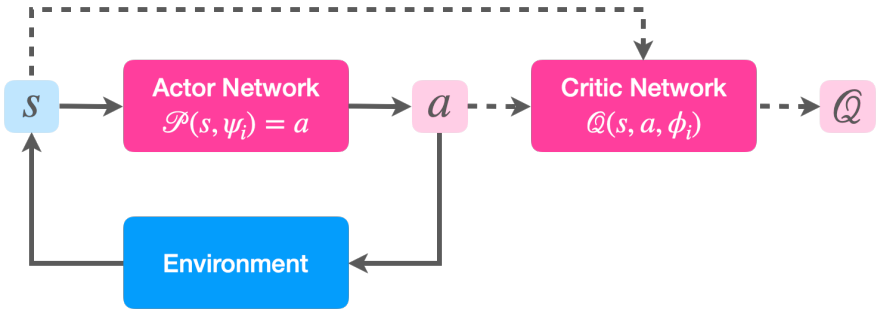

Figure 2.2: Actor-Critic learning scheme: During training the Actor and Critic Network train simultaneously (dashed, solid lines). Once training is complete, the Actor network interacts with the environment independently as the feedback controller (solid lines).

optimal policy has been approximated. This loss is used for back-propagation through the Critic network and the resulting gradient is utilized in updating the Actor network [119]. The optimization algorithms implemented in training assume that samples are distributed independently and identically, which is generally untrue for data generated from our exploratory trajectories. To mitigate this, the algorithm is trained on minibatches of experience tuples, $e_t = (s_t, a_t, r_t, s_{t+1})$, selected randomly from a memory cache of past experience tuples. This memory cache technique is called *experience replay* and is implemented to combat the instabilities in Q-learning caused by highly correlated training sets [136]. The DDPG algorithm used is shown in Algorithm 1.

We utilize Actor-Critic networks each with dense two hidden layers of size 256 and 128 with ReLU activation functions. The output layer of the Actor and Critic networks are composed of tanh and linear activation functions, respectively. Increasing the hidden layer size did not appear to strongly influence overall performance. We employ a rolling experience replay buffer of generated training data, (s_t, a_t, r_t, s_{t+1}), of size 500,000 experiences and update with batch sizes of 128. We trained each agent for 4000 episodes, with each episode initialized randomly on the unforced KSE attractor and lasting 100 time units (400 actions). For exploration of state-action space during training we employ Ornstein-Uhlenbeck noise, $\mathcal{N}$, [119] to encourage action exploration.

Algorithm 1: Deep Deterministic Policy Gradient with Experience Replay

Initialize Critic network $Q(s, a, \phi)$, Actor network $\mathscr{P}(s, \psi)$ with random weights ϕ, ψ; Initialize target networks Q', $\mathscr{P}'$ with weights $\phi' \leftarrow \phi$, $\psi' \leftarrow \psi$;
Initialize $\beta = 1.0$;
for *episode = 1, M* **do**
    Initialize s_1 ;
    Initialize OU process, $\mathscr{N}$, for action exploration;
    for t *= 1, T* **do**
        Select action $a_t = \mathscr{P}(s_t, \psi) + \beta \mathscr{N}_t$;
        apply action a_t to environment, observe r_t, s_{t+1} ;
        store memory tuple (s_t, a_t, r_t, s_{t+1}) in replay memory cache update $s_t = s_{t+1}$;
        sample random minibatch of e_j=(s_j, a_j, r_j, s_{j+1}) from mem. cache;
        set $y_j = r_j + \gamma Q'(s_{j+1}, a_{j+1}, \phi')$;
        Update critic network: $L = \frac{1}{N}\sum_j (y_j - Q(s_j, a_j, \phi))^2$, Eq. 2.16 ;
        Update actor network with learning rate α_ψ:
        $\psi \leftarrow \psi + \alpha_\psi \frac{1}{N} \sum_j \nabla_a Q(s_i, a_i, \phi) \nabla_\psi \mathscr{P}(s_i, \psi)$ [119] ;
        update target networks with learning rate α_T: $\phi' \leftarrow \alpha_T \phi + (1 - \alpha_T)\phi'$,
        $\psi' \leftarrow \alpha_T \psi + (1 - \alpha_T)\psi'$;
        $\beta \leftarrow D_\beta \ \beta$;
    end
end

2.2.3 State-Action Space Symmetries and the Deep RL Problem

It is important to now consider the symmetry of the overall controlled system. As mentioned earlier, the KSE possesses a continuous translational symmetry and a discrete reflection symmetry. The presence of N equally spaced actuators breaks the continuous translation symmetry, leaving only equivariance with respect to shifts of integer multiples of L/N. The symmetry of the overall controlled system is the intersection of the symmetry group operations, which in this problem are simply the discrete translational shift τ_N (defined precisely below) and reflection operations. Thus any state of the system is $2N$-fold degenerate, or equivalently any state can appear in $2N$ dynamically equivalent forms. The impact this degeneracy has on learning the optimal policy is twofold. First, the $\mathscr{P}$ function should map dynamically equivalent states to dynamically equivalent actions, which requires the Actor network to learn to be an equivariant function with respect to the total controlled system's $2N$ symmetries, e.g. $\tau_N(\mathscr{P}(s, \psi)) = (\mathscr{P}(\tau_N(s), \psi))$. Second, the Q function should be in-

variant to discrete translations and reflections of the state-action input, which requires the Critic network to learn to be an invariant function of dynamically equivalent state-action pairs, e.g. $Q(s, a, \phi) = Q(\sigma(s), \sigma(a), \phi)$.

Because NNs are not intrinsically equivariant nor invariant, naive NNs must learn and internally approximate the optimal policy $2N$ times, one for each of the $2N$ dynamically equivalent sectors of state-action space. The implications of this are twofold. First, this requirement to learn redundant policies within the same network can be viewed as an implicit weight constraint that not only exhausts network capacity, but also requires an ergodic exploration of all of state-action space to generate sufficient training data. Inaccuracies in properly estimating the Q function lead to poor approximations in the optimal policy, and ultimately poor training and control performance. Second, in tow with control performance, if the policy approximation is unable to map dynamically equivalent states to dynamically equivalent actions, the policy cannot be the optimal policy. An illustration of this in a cartoon-world example is shown in Fig. 2.3, which possess a τ_4 (discrete translational shift) symmetry. Note that given dynamically equivalent states, the optimal policy, shown in Fig. 2.3a, should produce dynamically equivalent trajectories as a consequence of producing dynamically equivalent actions. A sub-optimal approximation of the optimal policy that does not respect symmetry, shown in Fig. 2.3b, yields dynamically nonequivalent actions, and thus dynamically nonequivalent trajectories and final states despite being given dynamically equivalent initial conditions.

Our method to circumvent these limitations is to move the deep learning problem to a symmetry-reduced subspace. We accomplish this by reducing the translational symmetry of the state observation via a modification of the method of slices [13], followed by a reflection symmetry reduction operation in order to obtain a discrete translational and reflection-reduced state. This symmetry-reduced state is then passed to the agent, which then outputs the corresponding optimal symmetry-reduced action. The previously removed symmetries are then reintroduced to the output actions prior to implementation to ensure that they

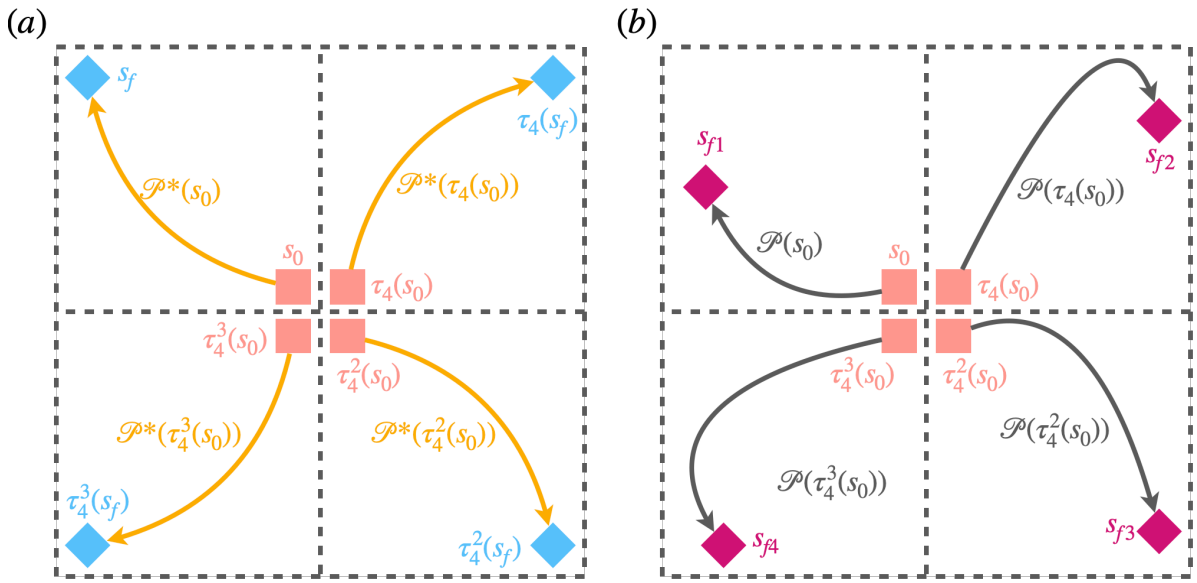

Figure 2.3: (a) For the initial condition s_0, the optimal controlled trajectory following the optimal policy $\mathscr{P}^*$ leads to state s_f. For simplicity, $\tau_4(s) = \tau_4(\pi/2, s)$. Given a translation of the s_0 by τ_4, following the optimal policy $\mathscr{P}^*(s)$ should yield a dynamically equivalent trajectory translated by $\pi/2$. (b) A policy that does not have symmetry enforced, $\mathscr{P}(s)$, yielding dynamically nonequivalent trajectories.

respect the true orientation of the system.

We first reduce discrete translational symmetries with a modified method of slices. This operation moves all state observations to the same discrete reference phase while preserving the relative location of the N actuators to the original and discrete translation-reduced state. The phase angle of the state can be calculated via Eq.(2.17),

$$\theta_1 = \text{arctan2}\left(b_1, c_1\right), \tag{2.17}$$

where $\text{arctan2}(b, c)$, not to be confused with $\arctan^2(b)$, is the 2-argument arctangent function that returns the phase of a complex number and is bounded by $-\pi$ and π. To preserve the uniqueness of a state-action pair, i.e. the relative location of the state to the N spatially fixed actuators, the state phase angle, θ_1, is rounded up to the nearest discrete phase, θ_N,

$$\theta_N = \frac{2\pi}{N}\text{ceil}\left(\frac{\theta_1}{2\pi/N}\right). \tag{2.18}$$

The Fourier state, F, is moved into the discrete translation-reduced subspace via the discrete translational reduction operator, $\hat{F}_k = \tau_N(\theta_N, F_k)$, where

$$\tau_N(\theta_N, F_k) = \exp(ik\theta_N)F_k. \tag{2.19}$$

In the discrete translation-reduced subspace, $\hat{F}$ preserves the relative location with respect to the actuators. The resulting real-valued Fourier state space vector in the discrete translation-reduced subspace is then,

$$\hat{F} = \left[\hat{b}_0, \hat{c}_0, \hat{b}_1, \hat{c}_1, \hat{b}_2, \hat{c}_2, \hat{b}_3, \hat{c}_3, \dots\right]. \tag{2.20}$$

Within the discrete translation-reduced subspace, reflection symmetric states are related by the following reflection operator with respect to N,

$$\sigma_N(\hat{F}) = \exp\left(\frac{2\pi}{N}ik\right)\sigma(\hat{F}). \tag{2.21}$$

For $N = 4$, the discrete translation-reduced reflection operator is defined as the following repeating sequence,

$$\sigma_4(\hat{F}) = \left[-\hat{b}_0, \hat{c}_0, -\hat{c}_1, -\hat{b}_1, \hat{b}_2, -\hat{c}_2, \hat{c}_3, \hat{b}_3, \dots\right]. \tag{2.22}$$

We note that the sign of $\hat{c}_2$ is the first unique value that can distinguish between two reflection symmetric states within the discrete translation-reduced subspace and thus construct a reflection indicator function $\rho = \text{sign}(\hat{c}_2)$. The reflection operator, σ_4, is then applied if the indicator value $\rho < 0$ to collapse reflection symmetric states into a common half of the

discrete translation-reduced subspace:

$$\breve{\hat{F}} = \begin{cases} \sigma_4(\hat{F}), & \text{if } \rho < 0. \\ \hat{F}, & \text{otherwise.} \end{cases} \tag{2.23}$$

The resulting discrete translational and reflection-reduced Fourier state, $\breve{\hat{F}}$, is then Fourier transformed back to the real domain and passed to the deep RL agent as the state observation. A schematic of this symmetry reduction process is shown in Fig. 2.4. By performing the appropriate transformations to the state, the deep RL agent learns purely within a discrete symmetry-reduced subspace, and thus outputs discrete symmetry-reduced actions, $\breve{\hat{a}}$. As a result, the agent trains with symmetry-reduced experiences, $\breve{\hat{e}}_t = (\breve{\hat{s}}_t, \breve{\hat{a}}_t, r_t, \breve{\hat{s}}_{t+1})$. We highlight here that the reward in our particular learning problem is constructed from quantities that are invariant to the symmetry transformations of the system (e.g. the dissipation of a particular state is the same regardless of its discrete-translational orientation). However, if one were interested in specifically targeting a state, u_{target}, and wanted to use a distance metric as the reward, $r = -||u - u_{target}||$, then that calculation must also be performed in the transformed space as well, $\breve{\hat{r}} = -||\breve{\hat{u}} - \breve{\hat{u}}_{target}||$. Importantly, because the output actions are symmetry-reduced actions, we must ensure they respect the orientation of the true state of the system before we apply them to the environment. This requires that the reverse symmetry operations that were applied to the state be applied to the actions prior to actuation. The control signal generated by the agent is therefore reflected if $\rho < 0$, then rotated by $N\theta_N/2\pi$, before being applied to the system. This ensures that dynamically equivalent states receive dynamically equivalent actions. Note that in this work the Actor-Critic deep RL model is inserted in “training domain", but generally any deep RL model may be chosen. Additionally, because the RL agent is isolated within the reduced space in this reformulated learning problem, the networks can be updated without back-propagating through the transform functions and these functions can be thought of as simply pre-/post-

Algorithm 2: Deep Deterministic Policy Gradient with Symmetry Reduction

Initialize Critic network $Q(s, a, \phi)$, Actor network $\mathscr{P}(s, \psi)$ with random weights ϕ, ψ; Initialize target networks $Q', \mathscr{P}'$ with weights $\phi' \leftarrow \phi$, $\psi' \leftarrow \psi$;
Initialize $\beta = 1.0$;
for *episode = 1, M* **do**
  Initialize s_1 and compute $\breve{\hat{s}}_1$;
  Initialize OU process, $\mathscr{N}$, for action exploration;
  for *t = 1, T* **do**
    Select action $\breve{\hat{a}}_t = \mathscr{P}(\breve{\hat{s}}_t, \psi) + \beta \mathscr{N}_t$, compute a_t;
    Apply action a_t to environment, observe $\breve{\hat{r}}_t$, s_{t+1};
    Compute $\breve{\hat{s}}_{t+1}$;
    Store symmetry reduced memory tuple $(\breve{\hat{s}}_t, \breve{\hat{a}}_t, \breve{\hat{r}}_t, \breve{\hat{s}}_{t+1})$ in mem. cache;
    update $s_t = s_{t+1}$, $\breve{\hat{s}}_t = \breve{\hat{s}}_{t+1}$;
    sample random minibatch of e_j=$(\breve{\hat{s}}_j, \breve{\hat{a}}_j, \breve{\hat{r}}_j, \breve{\hat{s}}_{j+1})$ from mem. cache;
    set $y_j = \breve{\hat{r}}_j + \gamma Q'(\breve{\hat{s}}_{j+1}, \breve{\hat{a}}_{j+1}, \phi')$;
    Update critic network: $L = \frac{1}{N} \sum_j (y_j - Q(\breve{\hat{s}}_j, \breve{\hat{a}}_j, \phi))^2$, Eq. 2.16 ;
    Update actor network with learning rate α_ψ:
     $\psi \leftarrow \psi + \alpha_\psi \frac{1}{N} \sum_j \nabla_a Q(\breve{\hat{s}}_j, \breve{\hat{a}}_j, \phi) \nabla_\psi \mathscr{P}(\breve{\hat{s}}_j, \psi)$ [119] ;
    update target networks with learning rate α_T: $\phi' \leftarrow \alpha_T \phi + (1 - \alpha_T)\phi'$,
     $\psi' \leftarrow \alpha_T \psi + (1 - \alpha_T)\psi'$;
    $\beta \leftarrow D_\beta \, \beta$;
  end
end

processing mechanisms around the Actor-Critic network ensemble. The symmetry-reduced DDPG algorithm is shown in Algorithm 2.

2.3 Results

Section 2.3.1 presents a quantitative comparison between a naive (i.e. “symmetry-unaware") agent, a naive agent trained with augmented data, and a symmetry-reduced agent. The learned control strategy and its dynamical significance are characterized in Section 2.3.2 while Section 2.3.3 examines a classical LQR approach to control for this problem. Comparison with the RL results provides some insight into why the RL algorithm learns the policy that it did. Finally, in Section 2.3.4, the robustness of the agent to input-output noise and

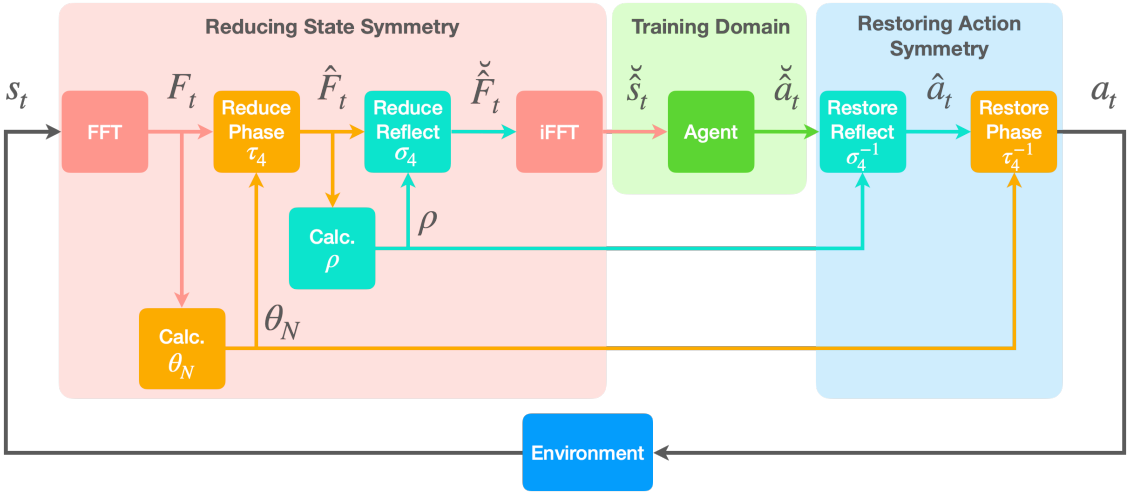

Figure 2.4: Flow diagram of discrete-symmetry reduced deep reinforcement learning. State observations have discrete translational symmetries reduced by τ_4, followed by a reflection symmetry reduction by σ_4. The symmetry reduced state is then passed to the agent, which outputs a symmetry reduced action. The previously removed reflection and translational symmetries are reintroduced to the output action before being implemented in the environment.

perturbations to system parameters is evaluated.

2.3.1 Performance Comparison

To illustrate the importance of discrete symmetry reduction, the naive and symmetry-reduced agents are tested with dynamically equivalent initial conditions, which are related by a translation of half the domain and a reflection operation. That is, we use initial conditions $u_0(x)$ and $T_{L/2}\Sigma u_0(x)$. The resulting trajectories controlled by the naive agent are shown in Fig. 2.5a and 2.5b, which exhibit distinctly different structures between the two. The dynamically nonequivalent trajectories are a product of the naive agent being unable to map dynamically equivalent states to dynamically equivalent actions. This dynamic inequality is due to the inherent difficulty for the NNs to consolidate and learn identical optimal sub-policies for each symmetric sector of state-action space (i.e. it is unable to become equivariant to symmetry-related state-action pairs), which is further exacerbated by the system's chaotic nature.

To aid the implicit learning of dynamically equivalent state-action mappings, we also

trained naive agents with additional synthetic training data, produced by applying symmetry operations to the originally generated data. Agents trained with this augmented data set will be called augmented naive agents. Shown in Fig. 2.5c and Fig. 2.5d are trajectories controlled by an augmented naive agent beginning with dynamically equivalent initial conditions. This augmented naive agent also produces dynamically nonequivalent trajectories, which indicate that it fails to learn dynamically equivalent state-action mappings.

In contrast, the symmetry-reduced agent, given dynamically equivalent initial conditions, will produce dynamically equivalent actions and therefore dynamically equivalent controlled trajectories, which are shown in Fig. 2.5e and Fig. 2.5f. We highlight here that the two controlled trajectories are identical and only differ by the symmetry-orientations of their initial conditions. As the initial conditions were reflected, the blue regions in Fig. 2.5e correspond to the red regions 2.5f and vice versa. As the initial conditions were also translated by $L/2$, the trajectories differ by a spatial shift of $L/2$. For example, the upward drifting red region of the transient in Fig. 2.5e corresponds to the downward drifting blue region of the transient of 2.5f. Note that these equivalent structures appear at the same temporal location but differ by only a translation and reflection. To further demonstrate this point, shown in Fig. 2.5g is the invariant quantity of dissipation, D, for the trajectories shown in Fig. 2.5e and 2.5f. As D does not depend on the orientation of the state, D is the same for both trajectories. As a result, the symmetry-reduced method inherently yields improved performance variance and robustness to initial conditions over the naive method, as the controlled trajectories of the naive agent depends on the orientation of the state while the symmetry reduced method does not. Additionally, while the dynamics of the naively controlled system remain chaotic, those with the symmetry-aware controller evolve to a low-dissipation steady state, a phenomenon that we analyze further in Section 2.3.2.

To demonstrate this improved performance, the naive and symmetry-reduced agent are tested to control 100 random initial conditions sampled from the attractor of the unforced KSE. The controlled trajectory duration is extended to 250 time units, 2.5 times the 100

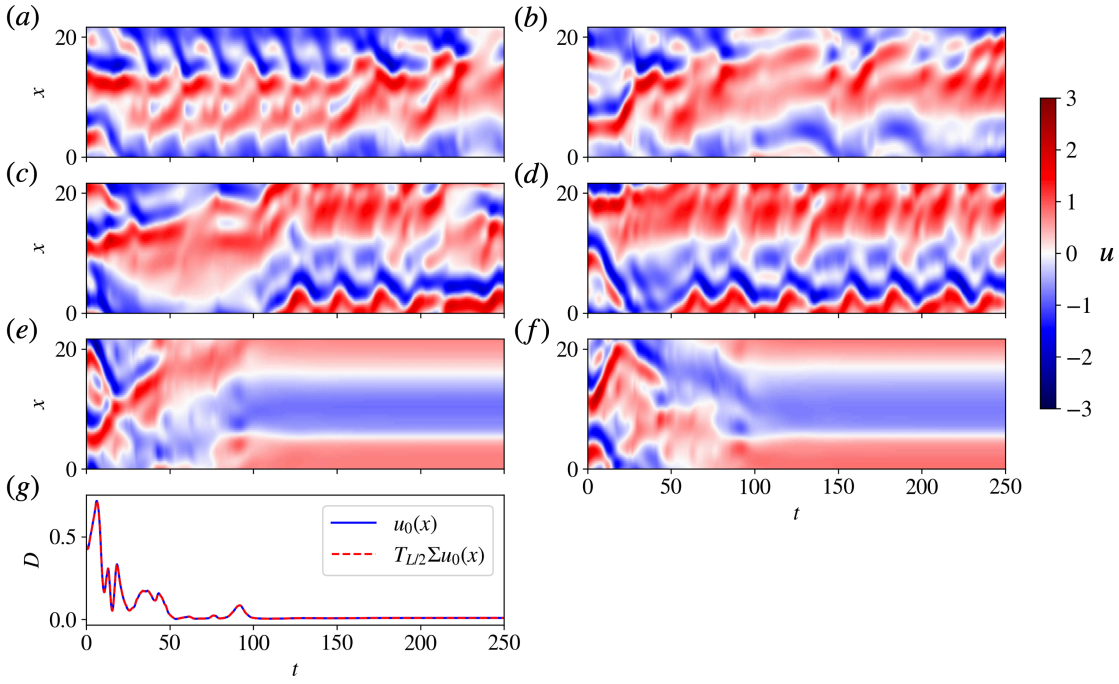

Figure 2.5: Pairs of controlled trajectories with dynamically equivalent initial conditions (translated by $L/2$ and reflected) controlled by the (a),(b) Naive agent, (c),(d) augmented naive agent, and (e),(f) symmetry- reduced agent. (g) The dissipation of the trajectories of (e) and (f).

time unit duration experienced during training, to examine the robustness of the policies beyond the training time horizon. The mean and standard deviation of $D + P_f$ of the 100 controlled trajectories are shown in Fig. 2.6 with respect to time. Notably, the addition of symmetry-reduction yields controlled trajectories with significantly lower mean $D + P_f$ and tighter variance than compared to the naive and augmented agents. Furthermore, the symmetry-reduced agent reaches its low $D + P_f$ target state in a much shorter time than the naive agents.

The advantage of symmetry-reducing the RL problem also appears in training. In Fig. 2.7, the mean reward return of 10 models is shown for each RL method with respect to training episode. The symmetry-reduced agents not only reaches greater reward returns than compared to the naive agents given the same amount of training, but they do so in

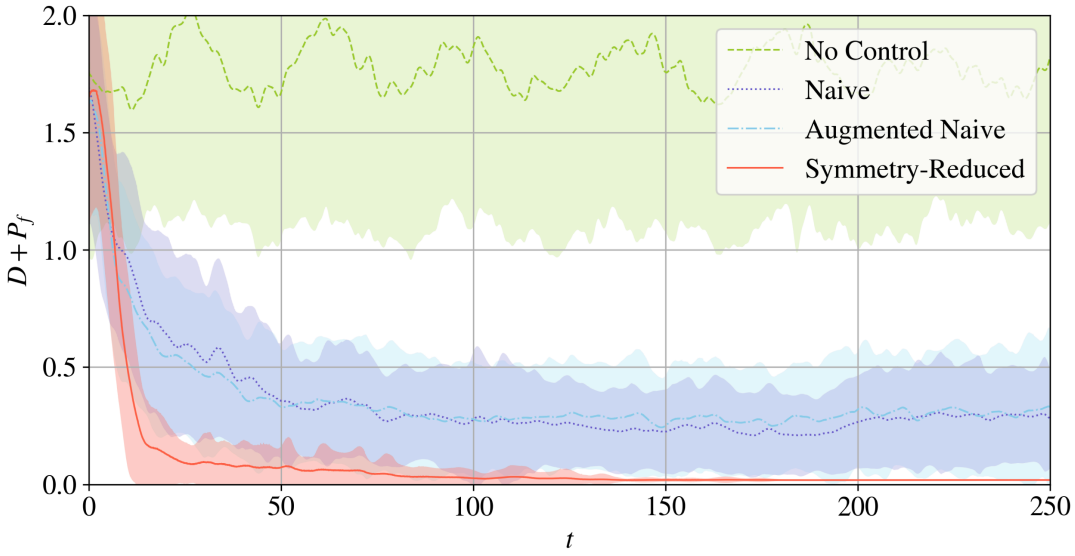

Figure 2.6: Ensemble mean $D + P_f$ of 100 trajectories controlled by: No control (green), Naive (purple), Augmented naive (blue), and translation+reflection-reduced (red). Each initial condition is randomly initialized on the KSE attractor. Standard deviation of each ensemble is shaded in its respective color.

significantly fewer training episodes, demonstrating the enhanced efficiency in training data usage. This improved training efficiency is a result of the symmetry-reduced agents only needing to learn one symmetric sector of state-action space, as opposed to the naive agents, which must rely completely on ergodicity to explore and learn all of state-action space. Furthermore, the variance in the training reward-return of the symmetry-reduced agents are much lower than that of the naive agents. This also highlights the improved policy robustness, as each episode differs in initial conditions and noisy actuation perturbations (exploration noise).

We also comment that the augmented naive agents with the additional synthetic symmetric training data exhibited improved early training performance compared to the naive agents. However, at the end of training, the augmented naive agents ultimately achieve similar reward returns as the naive agents without synthetic training data.

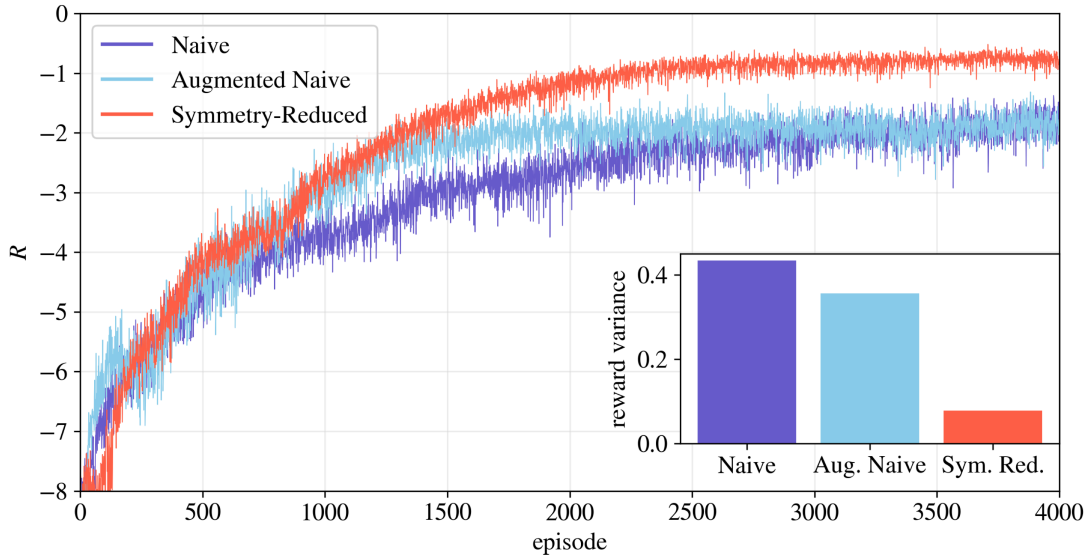

Figure 2.7: Ensemble training reward (10 models each) vs. episode for: Naive (purple), Augmented Naive with synthetic symmetric data (blue), and translation+reflection-reduced (red). The reward variance of the last 1000 episodes is shown in the inset.

2.3.2 Characterizing the Learned Control Solution

We noted above that the symmetry-aware agent drives the system from chaotic dynamics to a steady state. Fig. 2.8a illustrates such a controlled trajectory, along with the control action $f(t)$ in Fig. 2.8b, and the $D(t)$, $P_f(t)$ in Fig. 2.8c. In this figure, control action begins at $t = 100$. Qualitatively, the control action occurs in two phases. The first phase, approximately $t = 100$ to $t = 180$, is characterized by complex transient actuations that navigate the system to the neighborhood of a steady state. The second phase, approximately $t = 180$ and onward, is characterized by an essentially constant forcing profile with an extremely small time dependent residual corresponding with stabilizing the equilibrium state. We denote this "constant" forcing as $f = \alpha_{22}$, where α denotes the forcing profile and the subscript corresponds to the respective domain length L, and the steady state as $u_{\alpha_{22}}$. When control is removed the system returns to its original chaotic dynamics.

To understand the appearance of this steady state in the controlled system, we first note that at $L = 22$ in the absence of control, the KSE has a number of unstable steady states,

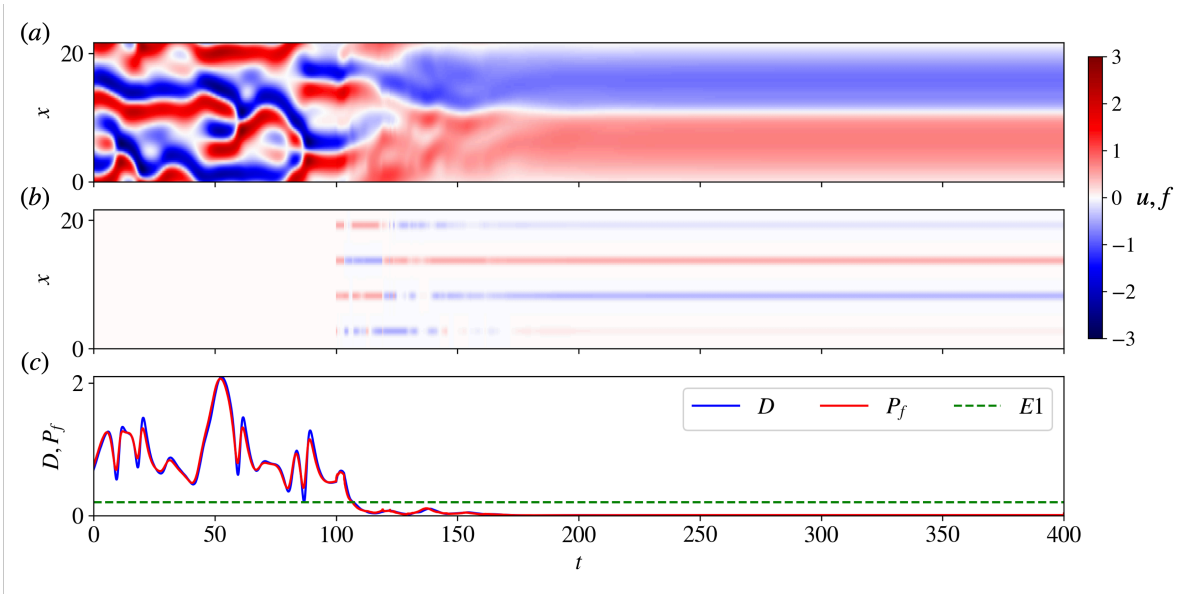

Figure 2.8: (a) Symmetry-reduced agent controlled trajectory vs. time. The controller is turned on at $t = 100$. (b) Forcing profile vs. time. (c) The dissipation and total power cost as a function of time. For reference the dissipation of $E1$ is included.

some of which lie in the vicinity of the chaotic attractor [32]. To investigate the dynamical connection between the state $u_{\alpha 22}$ found with the RL agent with forcing profile $f = \alpha_{22}$ and the uncontrolled KSE, continuation was performed between the final forced system to the unforced system. We accomplished our forcing continuation by iteratively Newton-solving for an equilibrium solution that satisfies Eq. (5.5) with initial solution guess of $u_{\alpha 22}$ and forcing profile $f_0 = \alpha_{22}$. The resulting solution, $E_{\alpha_{22}}$, was then used as an initial guess for a system with slightly reduced forcing amplitude. The process was repeated until $f = 0$, where the solution converged to an equilibrium solution of the unforced KSE. The solutions found for incremental scalings of $f = \alpha_{22}$ are shown in Fig. 2.9a, which converge to a known solution of the KSE denoted by [32] as $E1$. Interestingly, $E1$ is the lowest-dissipation solution known aside from the trivial zero solution, and $E_{\alpha_{22}}$ exhibits even lower dissipation and power input than $E1$. The discovered $E_{\alpha_{22}}$ solution corresponds to the $E1$ solution modified by the "jets" in a manner that smooths its peaks, leading to weaker gradients and thus lower dissipation.

Shown in Fig. 2.9b are the leading eigenvalues of the $E1$ and $E_{\alpha_{22}}$ solutions, demonstrat-

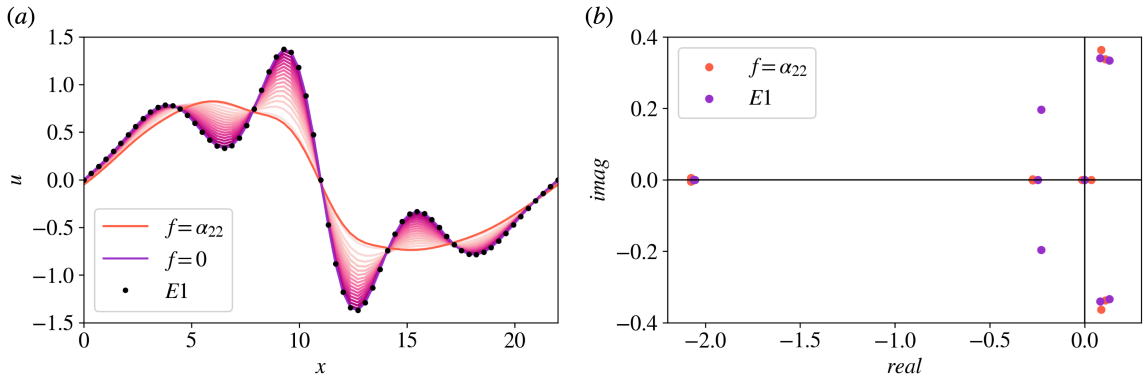

Figure 2.9: (a) Force-continuation solutions from $f = \alpha_{22}$ (red) to $f = 0$ (purple), yielding equilibrium solutions from $E_{\alpha_{22}}$ to $E1$ (dots). (b) The leading eigenvalues of the KSE linearized around $E_{\alpha_{22}}$ and $E1$.

ing that they are both linearly unstable. As the $E_{\alpha_{22}}$ solution is linearly unstable, we note that the agent maintains this state with oscillatory-like adjustments that are several orders of magnitude smaller than the mean actuation about $f = \alpha_{22}$.

Importantly, nowhere in the algorithm was $E_{\alpha_{22}}$ explicitly targeted – the algorithm discovers and stabilizes an underlying unstable steady state of the dynamical system, despite having been given no information about such solutions. Stabilizing an underlying steady state of the dynamical system is an efficient strategy as it requires less control effort than brute-forcing the system to a region of state space where it would not naturally reside. We speculate that this "strategy" of finding and stabilizing an unstable recurrent solution (steady state, periodic orbit) might arise in RL control of a wide variety of systems displaying complex dynamics. We contrast this with other recent data-driven control-target identifying methods, such as [10] which identifies and stabilizes periodic orbits by approximating their Poincaré mapping, whereas here we seek targets defined by macroscopic properties, and the learned solution turns out to be a recurrent solution.

2.3.3 Comparison to Linear Quadratic Regulator

To compare our learned control policy to a conventional control method, we compare to Linear Quadratic Regulator (LQR) [79] given the same control authority. We now consider the system state as x and the control signal as u where previously we referred to them as u and a, respectively, to maintain nomenclature consistency with LQR conventions. The LQR method seeks to find a gain matrix, $\mathbf{K}$, for a linear state-feedback controller, $u = -\mathbf{K}x$, that minimizes the quadratic cost function J,

$$J = \int_0^\infty (x^T \mathbf{Q} x + u^T \mathbf{R} u) dt. \tag{2.24}$$

for a system whose dynamics are approximated by a set of linear (or linearized) ODEs $\dot{x} = \mathbf{A}x + \mathbf{B}u$, where x is the state of the system and u the control input. We take $\mathbf{Q}$, the state cost, and $\mathbf{R}$, the input cost, to be the identity. Importantly, the target state of LQR i.e. the state about which the dynamical model is linearized must be chosen a priori. Although the KSE possesses nonlinear dynamics, LQR might in certain situations be able to control the dynamics toward its target, given sufficiently close initial conditions.

We first consider applying an LQR controller to the trivial zero solution, which in the interest of minimizing dissipation and power-input cost, is the natural target as it possesses zero dissipation and zero power input cost. The trivial zero solution is known to be linearly unstable. Shown in Fig. 2.10a is a trajectory of the trivial zero solution given an infinitesimal perturbation; it evolves to the chaotic attractor. In this case, we find that LQR cannot stabilize the zero solution. Given the linearized KSE dynamics and the available control authority, the LQR approach fails the Popov-Belevitch-Hautus (PBH) controllability test [79],

$$\text{rank}[\mathbf{A} - \lambda \mathbf{I} \quad \mathbf{B}] = n, \quad \forall \lambda \in \mathbb{C}, \tag{2.25}$$

where n is the number of rows of $\mathbf{A}$. This indicates that LQR is not capable of transferring

every state to the origin in finite time. Furthermore, it also fails the PBH stabilizability test [79],

$$\text{rank}[\mathbf{A} - \lambda\mathbf{I} \quad \mathbf{B}] = \text{n}, \quad \forall\lambda \in \mathbb{C} : \text{Re}[\lambda] \geq 0, \tag{2.26}$$

which indicates LQR is not capable of reaching the origin even given infinite time. These results suggest that the zero solution cannot be controlled by linear means with the current actuation scheme. We comment that this inability to control the zero solution is further linked to work performed by [69, 70], which demonstrated that periodically arranged actuator or sensor sites in systems with translational or reflection invariance can detrimentally impact controllability.

Shown in 2.10b is a trajectory initialized on the zero solution plus a small random perturbation, and controlled by the LQR controller designed based on the zero solution. We observe that this controller is unable to stabilize the trivial solution. We further note the immediate deviation from the zero solution, which is a product of a runaway controller, and emphasize that the resulting dynamics are a product of the LQR controller reaching the actuator saturation limit (which we set to be 10 times that available to the deep RL agent). We comment that although the initial short-time trajectory is asymmetric, the resulting symmetry of the trajectory is the product of the saturated controller actuating symmetrically.

We hypothesize that a RL policy attempting to stabilize the zero solution would likely meet the same issue as the LQR approach. This may explain why the agent chooses to target the solution $E_{\alpha_{22}}$. To further investigate, we linearize the KSE, forced with $f = \alpha_{22}$ around its steady state $E_{\alpha_{22}}$, and use LQR to find the gain matrix for this system:

$$a_{LQR} = -\mathbf{K}(u - E_{\alpha_{22}}). \tag{2.27}$$

Here a_{LQR} are the control signals produced by LQR, which are applied to the KSE via Eq. 2.8 as f_{LQR}. This linear feedback controller is then applied to the full nonlinear KSE

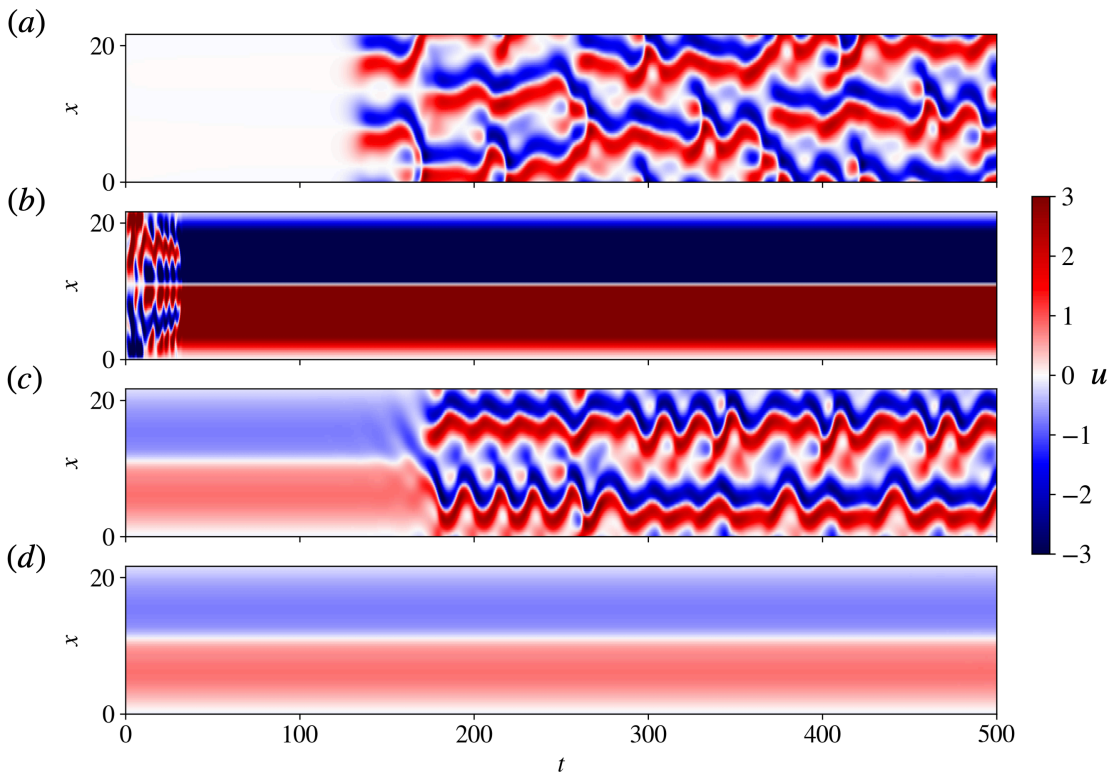

Figure 2.10: A trajectory initialized on the trivial solution with an infinitesimal perturbation with (a) no additional control, $f_{LQR} = 0$, and (b) with a LQR controller, f_{LQR}, based on the zero solution. A trajectory initialized on $E_{\alpha_{22}}$ with constant forcing $f = \alpha_{22}$ and an infinitesimal perturbation with (c) no additional control, $f_{LQR} = 0$, and (d) with a LQR controller, f_{LQR}, based on $E_{\alpha_{22}}$.

under constant forcing $f = \alpha_{22}$,

$$u_t = -uu_x - u_{xx} - u_{xxxx} + f_{\alpha_{22}} + f_{LQR}. \tag{2.28}$$

Shown in Fig. 2.10c is a trajectory initialized on $E_{\alpha_{22}}$ with constant forcing $f = \alpha_{22}$ and given an infinitesimal perturbation. In the absence of additional control, the linearly unstable solution $E_{\alpha_{22}}$ eventually evolves to the dynamics of the forced attractor. Shown in Fig. 2.10d is a trajectory initialized on $E_{\alpha_{22}}$ under constant forcing $f = \alpha_{22}$ with the LQR controller f_{LQR} applied. With the addition of the LQR controller, the perturbed $E_{\alpha_{22}}$ solution can be maintained. These observations indicate that one reason why the agent learns to discover

and stabilize $E_{\alpha_{22}}$ rather than the zero solution is because in the limit of approaching an equilibrium solution the agent will behave approximately linearly, and in that limit $E_{\alpha_{22}}$ can be controlled by linear means while the zero solution cannot. We reiterate that, while we have found that an LQR approach can stabilize $E_{\alpha_{22}}$, thus reducing energy consumption, that approach needed to know the existence and structure of the steady state a priori, while the RL approach did not.

2.3.4 Robustness

Chaotic systems are characterized by their sensitivity to noise and changes to system parameters. To assess the robustness of the agents to measurement and actuation noise, we tested the performance of our various RL policies, without additional training, on 100 trajectories of 250 time units with Gaussian measurement and actuation noise with zero mean and standard deviation 0.1. The ensemble mean and standard deviation of the $D + P_f$ trajectories of each agent type is shown in Fig. 2.11. Comparison with Fig. 2.6 reveals that all agent have comparable performance with or without noise. In particular the symmetry-reduced agent is capable of maintaining its high performance compared to the naive and augmented naive agents.

For the KSE, altering the domain size parameter, L, can lead to very different dynamics. These distinct dynamics are shown in Fig. 2.12a and Fig. 2.12b for domain sizes of $L = 21$ and $L = 23$, respectively. To assess the robustness of the learned control policy to changes in L, the symmetry-reduced agent trained in the domain $L = 22$ is applied, without additional training, to control the KSE dynamics at $L = 21$ and 23. In these experiments we maintain the same actuator jet parameters as was available in $L = 22$ training (spatial Gaussian distribution and magnitude range) while maintaining equidistant placement in the new domain sizes. Shown in Fig. 2.12c and Fig. 2.12d are trajectories with the controller turned on at $t = 100$, in domain sizes of $L = 21$ and $L = 23$, respectively. We observe that the agent drives both systems to equilibrium-like states similar to that found in the original

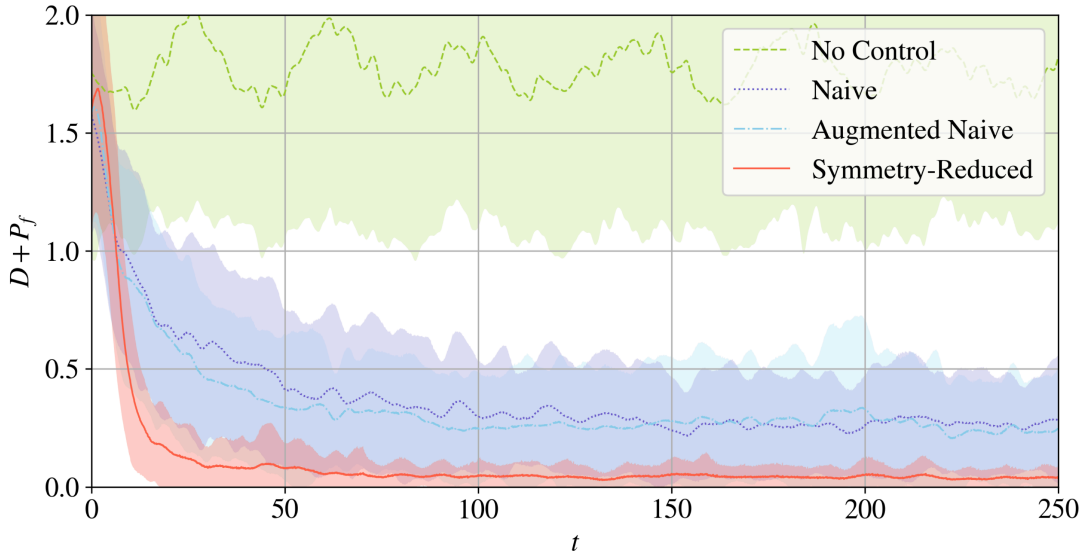

Figure 2.11: Ensemble $D + P_f$ vs. Time for 100 initial conditions controlled by: Ensemble mean $D+P_f$ of 100 trajectories controlled by: No control (green), Naive (purple), augmented naive (blue), and translation+reflection-reduced (red). All agents experience Gaussian measurement noise and actuation noise of mean zero and standard deviation 0.1.

domain $L = 22$. Furthermore, shown in Fig. 2.12e and Fig. 2.12f are D and P_f of these controlled trajectories; these controlled steady states again exhibit low dissipation and low power-input. In both $L = 21$ and $L = 23$ the targeted states are also unstable, as once the controller is switched off the systems return to their respective typical dynamics. These experiments highlight the robustness of the control policy to new unseen dynamics as well as deviations in relative control authority, as the artificial jets are smaller relative to the $L = 23$ domain than in the original $L = 22$ domain.

To investigate the dynamical connection between the final targeted states of the $L = 21, 23$ systems and the original $L = 22$ system, a two-stage continuation was performed. First, continuation in the magnitude of the forcing was performed to determine the connection between the forced systems yielding $E_{\alpha_{21}}$ and $E_{\alpha_{23}}$ and their unforced counterparts ($f = 0$) of their respective domain size. The solutions found as f is decreased to zero are shown in Fig. 2.13a and Fig. 2.13b for $L = 21$ and $L = 23$, respectively. These results reveal that $E_{\alpha_{21}}$ and $E_{\alpha_{23}}$ are the forced counterparts of existing equilibria in the unforced systems,

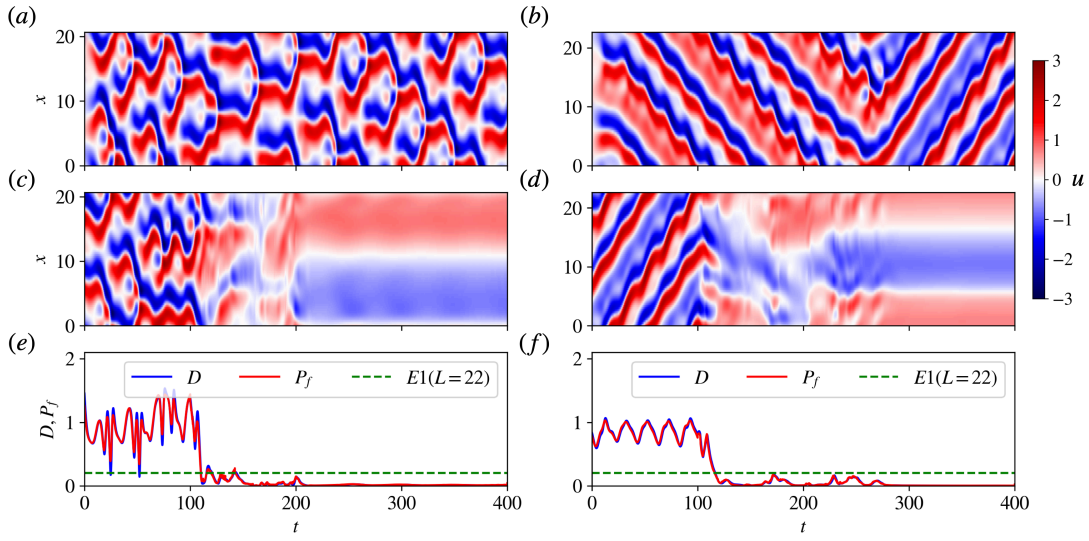

Figure 2.12: Typical dynamics for domain sizes of (a) $L = 21$, (b) $L = 23$. Controlled trajectory in which the $L = 22$ symmetry reduced agent is applied with no additional training at $t = 100$ in domain sizes of (c) $L = 21$, (d) $L = 23$. Dissipation and total power input as a function of time for the controlled trajectories in domain sizes of (e) $L = 21$, (f) $L = 23$.

which we denote as E_{L21} and E_{L23}, respectively.

We next perform a second continuation, this time in the domain size, to determine the connection between E_{L21}, E_{L23} and the original dynamics of $L = 22$. Solutions are shown in Fig. 2.13c and Fig. 2.13d respectively, as the domain size changes. These evolve to the $E1$ solution of $L = 22$. These connections indicate that the symmetry-reduced agent is also capable of finding and stabilizing the forced $E1$ solution in domain sizes it has not seen before. Interestingly, the equilibria found by the agent, $E_{\alpha_{21}}$ and $E_{\alpha_{23}}$, are not simply spatial dilations or compressions of $E_{\alpha_{22}}$, as $E_{\alpha_{21}}$ exhibits 4 velocity peaks while $E_{\alpha_{21}}$ exhibits only two. Furthermore, we comment that the long-time mean actuation profiles utilized by the agent in the two unseen domain sizes are also distinctly different than that utilized in $L = 22$, which indicates the agent is not just simply imposing the same long-time control signals it found in its original domain size of training.

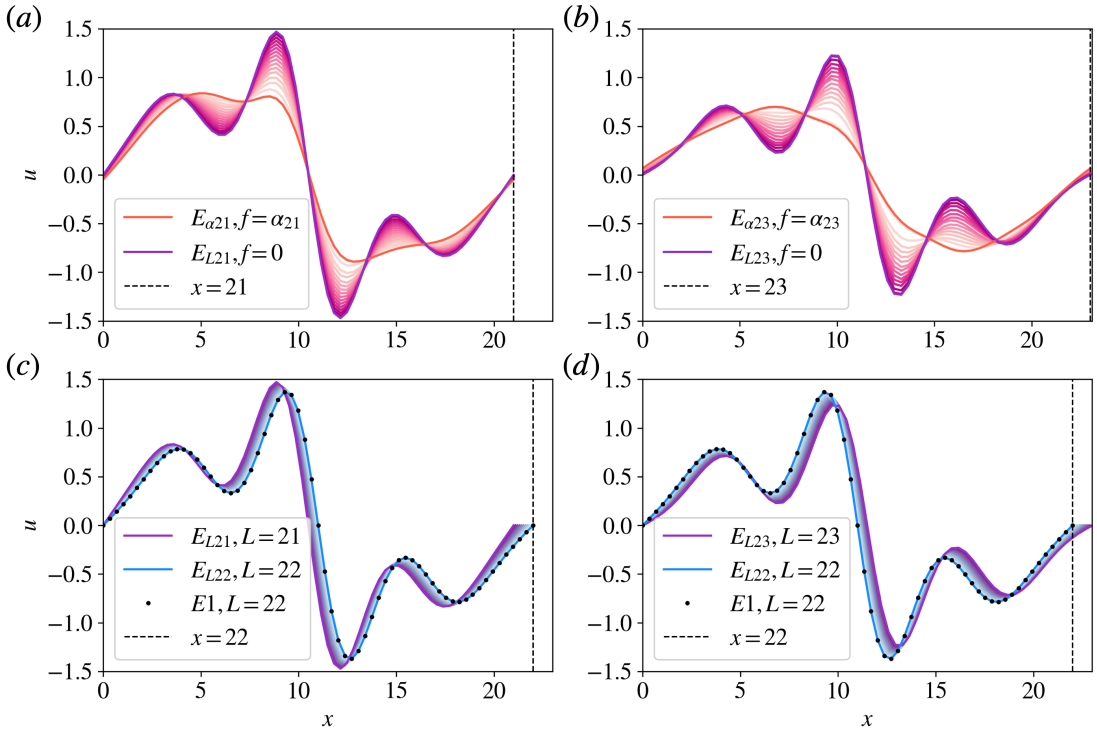

Figure 2.13: Continuation in forcing and domain size. (a) Forcing continuation from $f = \alpha_{21}$ (orange) to $f = 0$ (purple) to yield $E_{\alpha_{21}}$ to E_{L21}. (b) Forcing continuation from $f = \alpha_{23}$ (orange) to $f = 0$ (purple) to yield $E_{\alpha_{23}}$ to E_{L23}. (c) Domain size continuation from $L = 21$ (purple) to $L = 22$ (blue) to yield E_{L21} to $E1$ (dots). (d) Domain size continuation from $L = 23$ (purple) to $L = 22$ (blue) to yield E_{L23} to $E1$ (dots).

2.4 Summary

Although deep RL in recent years has demonstrated the capability of controlling systems with high-dimensional state-action spaces, its naive application towards spatiotemporal chaotic systems exhibiting symmetry, which encompasses many flow geometries of interest, can be limited by NN architecture and the cost of exploring the full state-action space. In this chapter we proposed a modification to the general deep RL learning problem that can better learn control strategies for chaotic flow problems exhibiting symmetries by moving the learning problem into a state-action symmetry-reduced subspace.

Our method alleviates technical demands of NN architectures in existing deep RL methods such as the need for the Actor and Critic networks in DDPG to learn weight constraints

that preserve equivariance and invariance, respectively. From a policy perspective, symmetry reduction alleviates the need to learn and consolidate the optimal policy for each symmetric-subspace within a single network, freeing capacity for approximating the optimal policy while maintaining a dynamically equivalent state-action mapping. As the learning problem is performed in the symmetry-reduced subspace, all training data is also generated within the symmetry-reduced subspace improving training data efficiency, as the agent no longer requires a complete exploration of the full state-action space as it would in the naive application of deep RL. Although in this work we utilized the DDPG algorithm, the commentary and conclusions drawn regarding symmetry reduction of the learning space can be extended to other deep RL methods.

We demonstrated these ideas by controlling the periodic KSE to minimize dissipation, a spatiotemporally chaotic model system for turbulence that exhibits translational and reflection symmetries. We show that by reducing the symmetry of the learning problem we can obtain faster and more consistent learning. Furthermore, we demonstrate that the control strategy found by the symmetry-reduced agent is robust to input/output noise as well system parameter perturbations. Finally, we observe that in order to achieve the objective of reduced overall power consumption, the symmetry-reduced agent discovers a low-dissipation equilibrium solution of a nontrivially forced KSE. This observation highlights a potentially important connection to effective control approaches for drag reduction in turbulent flows, as the dynamics of turbulence are organized, at least to some extent, by underlying invariant solutions known as Exact Coherent States [67].

We further emphasize that conventional controllers typically target microscopic objectives, such as a priori known states that exhibit desirable macroscopic properties such as system dissipation, pressure drop, etc., but these a priori targets may not always be accessible with the available control authority. In our experiments here, the symmetry-reduced deep RL demonstrates the potential to serve as a discovery tool for alternative solutions with desirable macroscopic properties. These discovered states might then be utilized as alterna-

tive control targets for conventional controllers when a priori known states are inaccessible given the available control freedom.

3

Reinforcement learning with model-based control[1]

Deep reinforcement learning (RL) is a data-driven method capable of discovering complex control strategies for high-dimensional systems, making it promising for flow control applications. In particular, the present work is motivated by the goal of reducing energy dissipation in turbulent flows, and the example considered is the spatiotemporally chaotic dynamics of the Kuramoto-Sivashinsky equation (KSE). A major challenge associated with RL is that substantial training data must be generated by repeatedly interacting with the target system, making it costly when the system is computationally or experimentally expensive. We mitigate this challenge in a data-driven manner by combining dimensionality reduction via an autoencoder with a neural ODE framework to obtain a low-dimensional dynamical model from just a limited data set. We substitute this data-driven reduced-order model (ROM) in place of the true system during RL training to efficiently estimate the optimal policy, which can then be deployed on the true system. For the KSE actuated with localized forcing ("jets") at four locations, we demonstrate that we are able to learn a ROM that accurately captures the actuated dynamics as well as the underlying natural dynamics just from snap-

[1]The text of this chapter is adapted from the publication by K. Zeng, A. J. Linot and M. D. Graham *Soc. Royal Proc. A.*, 478, 2022

shots of the KSE experiencing random actuations. Using this ROM and a control objective of minimizing dissipation and power cost, we extract a control policy from it using deep RL. We show that the ROM-based control strategy translates well to the true KSE and highlight that the RL agent discovers and stabilizes an underlying forced equilibrium solution of the KSE system. We show that this forced equilibrium captured in the ROM and discovered through RL is related to an existing known equilibrium solution of the natural KSE.

3.1 Introduction

In recent years, deep reinforcement learning (RL), a data-driven model-free control method, has achieved recognition for its ability to learn and discover complex control strategies for high-dimensional systems. Many of its milestone achievements involve defeating the best professional players in complex games such as GO [178], DOTA II [144], and Starcraft II [198], as well as the leading GO, Chess, and Shogi engines [179]. The successes of RL in these complex and nontrivial systems has made its application towards control of high-dimensional chaotic dynamical systems, such as turbulent flows, extremely promising. Deep RL has recently been demonstrated to be able to discover strategies in flow systems that exhibit nonchaotic dynamics such as reducing the drag in flow over bluff bodies [44, 117, 147, 156, 164] and decreasing convection amplitude in Rayleigh-Benard systems [4]. Using the Kuramoto-Sivashinsky equation (KSE) as an example, Bucci et al. [12] demonstrated that RL is able to direct a system with natural chaotic dynamics to a given (i.e. previously known) equilibrium solution of the system. In another study of the KSE in a chaotic parameter regime, the present authors [219] demonstrated that an RL policy whose aim is to minimize dissipation is able to discover and stabilize a nontrivial underlying equilibrium solution with low dissipation, even when no such information of its existence is given. The aim of the present work is to combine RL with data-driven reduced-order dynamical models for the purpose of control of spatiotemporal dynamics, as a step toward the goal of reducing energy

dissipation in turbulent flows.

Despite all of the successes of deep RL, a major challenge is that it suffers from poor sampling complexity. For example, recent work by Du et al. [39] showed that even if the optimal control policy could be perfectly represented by a linear function, the reinforcement learning agent still requires an exponential number of trajectories to find a near-optimal policy. Furthermore, the overall practical problem is further exacerbated by the temporal credit assignment problem, which is the difficulty in parsing out what actions in a long series of actions actually contributed to the end result–often bottlenecking RL algorithms attempting to train large models with millions of weights in application [75].

In practice, this issue is often addressed by simply brute-force generating an extraordinary amount of interaction data from the target system for training. For example, to achieve the previously mentioned accolades, over 11,000+ years of DOTA II gameplay [144], 4.9 million self-play games of GO [178], and 200 years of Starcraft II gameplay [198] was required. In the realm of fluid control, even learning a two-parameter control scheme for reducing drag in an LES bluff-body simulation required over 3 weeks of training and simulation time [44]. The default solution of simply generating more data can quickly make deep RL a prohibitively costly solution when the target flow system is computationally (e.g. via direct numerical simulations) or experimentally (e.g. via wind tunnel tests) expensive to realize.

Despite these challenges, there are growing efforts toward developing more data-efficient learning algorithms and work-around techniques. These efforts can be broadly categorized into three groups: modified RL algorithms with added complexity, true off-policy RL, and model-based RL. The first group, modified RL algorithms, is a catch-all group comprising of methods that build upon existing established algorithms with advanced architectures and formulations and/or incorporate domain knowledge or hierarchy to improve training stability and data efficiency at the cost of greater implementation and training complexity. For example, [219] reduced the data-sampling complexity of RL by exploiting the natural symmetries of the system and [140] found complex behavior tasks could be learned efficiently

via a hierarchical RL framework. A comprehensive survey of RL combined with attention mechanisms, transfer learning, and hierarchy can be found in [118]. The second group, true off-policy or Offline RL, is comprised of methods that reformulate or modify the typical semisupervised RL problem to extract control strategies from a pre-existing limited data set of transitions [1, 115]. However, these methods are still in the early stages of development and it is still unclear how well they perform and generalize in our complex systems of interest.

The last group, model-based RL, are methods that utilize a model of the target environment. These methods are advantageous in that they can leverage the existing vast classical modeling principles as well as the newly emerging deep learning techniques. As a result, model construction, implementation, and degree of data-drivenness vary widely from method to method. For example, Nagabandi et al. [141] introduced the Model-Based+Model-Free method, which utilizes a system model and an MPC controller to initialize the RL agent's weights. Towards more data-driven approaches, Wahlstrom et al. [204] introduced Deep Dynamical Models, which uses an autoencoder to learn a reduced-order representation of the observation data and a feedforward neural network to learn a discrete time stepping model in said reduced-order representation to control an inverted pendulum from pixel inputs. Similarly, Watter et al. [210] also utilized autoencoders and feedforward neural networks to learn a local linear dynamical representation of the pendulum problem. Feinberg et al. [45] utilized models to generate short "imagined" trajectories at each training step to better estimate the value functions. This so-called "Dyna"-style of model-based RL [186] has been recently applied in control of spatiotemporally chaotic systems by Liu et al. [129] who demonstrated with autoencoded recurrent network models containing hardcoded boundary conditions and soft conservation constraints that the number of real interactions needed with the target system to achieve control could be greatly reduced. Alternatively, Chua et al. [26] demonstrated another form of model-based RL wherein a data-driven dynamics model is iteratively learned online with model predictive control (MPC) serving as the controller. The authors demonstrated that this method was able to reach similar asymptotic performance to

state-of-the-art model-free RL algorithms on benchmark tasks while using much fewer data samples, although at the expense of requiring real-time model trajectory optimization at every step during implementation. Finally, Ha and Schmidhuber [75] introduced recurrent world models, which model the control system of interest with mixture density networks and recurrent neural networks. In this method, the authors use a data-driven model in place of the real environment during RL training to estimate a control policy for driving a 2D car and playing DOOM. We will refer to this style of utilizing a data-driven model as a surrogate training ground as "model-based RL".

The learning process of the above-defined model-based RL parallels the classic control design philosophy [135] which is: 1) create a model for control, 2) design a control policy based on the model, and 3) simulate based on a high-fidelity model and repeat if needed. In parallel to these principles, we seek to: 1) *learn* a model – specifically a reduced-order, but still highly accurate model – for control from data, 2) *learn* a control policy based on the model with RL, and 3) validate based on a high-fidelity model.

A key aspect of this approach is the development of an efficient reduced-order model of the process, which we will consider to be completely governed by a state space representation

$$\frac{ds}{dt} = f(s, a), \tag{3.1}$$

or its discrete-time version

$$s_{t+\tau} = F(s_t, a_t), \tag{3.2}$$

where $s \in \mathbb{R}^d$ is the state of the system and $a \in \mathbb{R}^{d_a}$ a time-dependent actuation. Our focus will be the continuous-time case Eq. 3.1, where we will seek a reduced-order representation

$$\frac{dh}{dt} = f(h, a), \tag{3.3}$$

where $h \in \mathbb{R}^{d_h}$ and $d_h \ll d$. Before describing our specific approach, we briefly review other

methods for developing dynamic models from data. A more detailed discussion is given in [123].

If the governing equations of the process are known, one approach is simply to use a direct simulation without any model reduction. Here of course none of the efficiency gains enabled by reducing the dimension of the model are realized. A traditional approach to reducing the dimension of these problems is the Galerkin projection method [84] and other variants like nonlinear Galerkin [68, 93, 133] or post-processing Galerkin [52], which have higher accuracy than the traditional Galerkin approach for given number of unknowns. Additionally, such approaches can be improved by adding data-driven information. For example, Wan et al. [206] approximated the unresolved dynamics in nonlinear Galerkin with an LSTM. The concept of combining first-principles and data-driven models is a widely studied one (see e.g. "physics-informed neural networks" [158]).

In the present work, however, our focus is data-driven models. The classical approach is to seek a linear discrete-time model; a specific example of this approach, which has seen wide use recently even for nonlinear systems, is dynamic mode decomposition (DMD)[107]. We describe and implement an adaption of DMD to closed loop control, known as DMDc, in Section IIIB. As a linear state space model, at long times, trajectories predicted by DMD always evolve toward a fixed point or a quasiperiodic orbit with a discrete set of frequencies, so it cannot faithfully characterize chaotic dynamics, which is our interest here. In principle, for all nonlinear systems there does exist a linear, albeit infinite-dimensional, operator called the *Koopman* operator that describes the evolution of arbitrary observables [14, 108]. We choose not to take this approach because our aim is to reduce the dimension of our model. In any case, effective methods for approximation of chaotic long-time dynamics with data-driven Koopman operator approximations remain elusive.

With the increasing power of neural network representations, nonlinear data-driven modeling methods have received substantial recent attention. Two popular methods include recurrent NN (RNN) [82, 86] and reservoir computing [92, 112]. These approaches are similar

in that a hidden or reservoir state r_t is used along with the current state s_t to predict a future hidden or reservoir state $r_{t+\tau}$. Then a function is used to predict the new state $s_{t+\tau}$ from the hidden state. The two approaches are different in how these functions are approximated. In reservoir computing the functions for evolving the reservoir are chosen a priori, and the mapping back to $s_{t+\tau}$ is learned. In RNNs, specific neural networks are trained to learn all of the functions – two examples are LSTMs [82] and gated recurrent units [27]. Vlachas et al. [202] shows that both these methods perform well at predicting chaotic dynamics. However, because they involve hidden variables, these approaches do not reduce the dimension of the system, but rather increase it: i.e. they do not determine $s_{t+\tau}$ from s_t alone, but rather also use past history stored in r_t to make predictions.

A more direct approach to learning dynamical models is to learn the right-hand side (RHS) of Eq. 3.1 or Eq. 3.2 from data. One way to do this, at least for low-dimensional systems, is using the "Sparse Identification of Nonlinear Dynamics " (SINDy) approach [11]. In this method a dictionary of candidate nonlinear functions are selected to approximate f in Eq. 3.1. Then, by using sparse regression, the dominant terms are identified and kept. Alternately, one could simply represent f with a neural network [63]. If one does not have time-derivative data or estimates, then the neural ODE approach [19, 123] can be used.

Now we turn to the issue of dimension reduction. The dynamics of formally infinite-dimensional partial differential equations describing dissipative flow systems are known to collapse onto a finite-dimensional invariant manifold [189]–a so-called "inertial manifold". In past works, it was shown that one can capture a high-dimensional system's dynamics on this lower-dimensional manifold by using a combination of autoencoders, to identify the manifold coordinates, and Neural ODEs (NODE) [19], to model the dynamics on the manifold in discrete time [122] or continuous time [123]. Our aim is to adapt this data-driven, low-dimensional model framework as a data-efficient surrogate training grounds for model-based RL. Once training is complete, this ROM-based policy is then deployed to the real system for assessment or further fine-tuning. We refer to our method as *Data-driven*

Manifold Dynamics for Reinforcement Learning, or DManD-RL for short.

Unlike the previous implementations of data-driven models in model-based RL [75, 204, 210], which model the evolution of the controlled system in fixed discrete time steps, our method models the vector field of the dynamics on the manifold in the presence of control inputs using neural ODEs [19]. The choice of using neural ODEs has several benefits: 1) it allows one to utilize the vast and already present array of integration schemes for time-stepping, 2) it allows one to train the model from data collected from variable time step sizes as well as make variable time step predictions, which grants one the ability to adjust the transition time step in RL without resorting to recollecting new data and training a new dynamics model, 3) it is a natural formulation for the modeling of the dynamics of our physical flow systems. We highlight that this framework is not limited to fixed time intervals, can utilize unevenly and/or widely spaced data in time, respects the Markovian nature of the dynamics of the systems we are interested in capturing, and respects the Markov environment assumption that underlies RL theory [186], unlike other common data-driven approaches such as recurrent neural networks [82, 86] and reservoir computing [92, 112]. Finally, we do not impose that the underlying relationship between the control input and the dynamics is affine in control (i.e. that $ds/dt = f(s) + g(s)a$ where s is the state and a is the control input), as some past works have done [210]. This is advantageous for application towards complex flow control systems of practical interest as many, such as aircraft [9] and underwater vehicles [54], do not have relationships with this affine structure. We highlight here an important interplay regarding the RL problem and our dissipative systems of interest. In RL the general algorithm seeks to learn a strategy by exploring and optimizing the state-action space. At first glance, this may seem hopeless for flow systems exhibiting high-dimensional ambient spaces. However, an important distinction here is that although our formally infinite-dimensional flow systems may require $\mathcal{O}(10^{6+})$ degrees of freedom to resolve in the ambient space, the underlying dynamics can be much lower dimensional. This indicates that the underlying complexity of the problem is independent

of mesh resolution (assuming the system is already resolved). Assuming we can find a high-fidelity reduced-order representation of the data, the state-action space that we seek to optimize and capture in our method is actually only of complexity *manifold dimension* $\times$ *action dimension*, which leaves an optimistic outlook for our method toward application to larger and more complex flow systems.

The remainder of this chapter is structured as follows: In Section 3.2 we introduce the Kuramoto-Sivashinsky equation, a proxy system for turbulent flows that displays rich spatiotemporal chaos, and the control objective which serves as a drag reduction analog. We conclude this section with an outline of our ROM-based RL framework. In Section 3.3 we examine the performance of our learned reduced-order model, the control strategy extracted from the model using RL, and the dynamical systems relevancy of the strategy. In Section 3.4 we summarize our findings and discuss applications to more complex systems.

3.2 Formulation

3.2.1 Kuramoto-Sivashinsky Equation

The Kuramoto Sivashinsky Equation (KSE) is given by,

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} - \frac{\partial^2 v}{\partial x^2} - \frac{\partial^4 v}{\partial x^4} + \mathcal{F}(x,t). \tag{3.4}$$

Here f is a spatio-temporal forcing term that will be used for control actuation. We consider the KSE in a domain of length $L = 22$ with periodic boundary conditions as this is a dynamically well characterized system [32]. The uncontrolled KSE, $\mathcal{F} = 0$, exhibits rich dynamics and spatio-temporal chaos, which has made it a common toy problem and proxy system for the Navier-Stokes Equations. Spatially localized control is implemented in the KSE with $N = 4$ equally spaced Gaussian "jets" located at $X \in \{0, L/4, 2L/4, 3L/4\}$ as done by Bucci et al. [12] and Zeng and Graham [219], where $a_i(t) \in [-1, 1]$ is the control signal

output by the RL control agent,

$$\mathcal{F}(x,t) = \sum_{i=1}^{4} \frac{a_i(t)}{\sqrt{2\pi\sigma_s}} \exp\left(-\frac{(x-X_i)^2}{2\sigma_s^2}\right). \tag{3.5}$$

The system is time evolved with a time step of $\Delta t = 0.05$ using the same numerical method and code as Bucci et al. [12] with a third-order semi-implicit Runge-Kutta scheme, which evolves the linear second and fourth order terms with an implicit scheme and the nonlinear convective and forcing terms with an explicit scheme. Spatial discretization is performed with Fourier collocation on a mesh of 64 evenly spaced points and in our formulation the state vector u consists of the solution values at the collocation points. To serve as an analogue to energy-saving flow control problems, we are interested in the minimization of the dissipation, D, and total power input, P_f required to power the system and jets of the KSE system, which are described by $D = \langle(\frac{\partial^2 u}{\partial x^2})^2\rangle$ and $P_f = \langle(\frac{\partial u}{\partial x})^2\rangle + \langle u\mathcal{F}\rangle$, respectively. Here $\langle\cdot\rangle$ is the spatial average. These arise from writing an evolution equation of the "kinetic energy" $\frac{1}{2}\langle u^2\rangle$ by multiplying (3.4) by u and integrating over the domain to yield the equation

$$\frac{d}{dt}\frac{1}{2}\langle u^2\rangle = P_f - D. \tag{3.6}$$

3.2.2 Background and Method Formulation

At its core, deep RL is a cyclic learning process with two main components: the RL agent and the environment. The agent is the embodiment of the control policy and generally a neural network, while the environment is the target system for control. During each cycle, the agent makes a state observation of the system, s_t, and outputs an action, a_t. The impact of this action on the environment is then quantified by a scalar quantity, r_t, which is defined by the control objective. During training, the agent attempts to learn the mapping between s_t and a_t that maximizes the cumulative long-time reward and updates accordingly each cycle. This learning cycle is shown in Fig. 3.1.

Deep RL's poor sampling complexity makes this cyclic process costly or even intractable for learning control strategies when simulations or experiments of the environment are expensive to realize. We aim to circumvent this issue in a completely data-driven manner by training the RL agent with a surrogate reduced-order model (ROM) of the target system. Our method can be divided into five steps: 1) obtain ROM training data 2) learn a reduced-order embedding coordinate transformation using an autoencoder 3) learn the RHS of the controlled system's dynamics in the reduced-order coordinates using Neural ODEs, 4) extract a control policy from this ROM using deep RL, 5) deploy and assess the control strategy in the real system. An outline of this process is presented in Fig. 3.2.

Step 1: Training Data Collection: We obtain our training data for our data-driven model by observing actuated trajectories of the target system. As we assume there is neither a model nor control strategy available at this stage, the actuations applied are randomly sampled from a uniform distribution of the available range of control inputs. We comment here that although the ambient space dimensionality for many flow systems can be extremely high, the underlying dynamics are generally much lower dimensional. And equally importantly, we only need to sample the responses of the system to the available actuations, not to random perturbations in every dimension of the ambient state space. Our approach determines the total number of degrees of freedom – 12 in the present case – needed to model the actuated dynamics. This means that when collecting data to sample the state-action space, we are not concerned with the dimensionality of the ambient input configurations and action configurations, but rather we need to the sufficiently sample the combinations of manifold states and actions. In other words, although our formally infinite-dimensional system may be resolved indefinitely by adding more grid points, the underlying dynamics do not change. Thus data sampling complexity is independent of mesh resolution (assuming the system is already resolved and we have a decent reduced-order representation) and the state-action space complexity is of size *manifold dimension* $\times$ *action dimension*.

The observed series of states, s_t, applied actions, a_t, at time t, and the resulting state

after τ time units, $s_{t+\tau}$ are saved as transition snapshots $[s_t, a_t, s_{t+\tau}]$ for ROM training. In our demonstration with the KSE, we take the state observable to be the solution of the KSE at time t, $s_t = u(t)$, and the coinciding control signals to the jet actuations to be the action, $a_t = a(t)$ (where $a_i(t)$ is the signal to the ith jet).

Step 2: Learning the Manifold Coordinate System: We utilize undercomplete neural network autoencoders [64] to learn the coordinate transformation to and from the lower-dimensional manifold of the actuated dynamics of our system. Undercomplete autoencoders are hourglass-shaped neural networks that are composed of two subnetworks, the encoder and decoder, are connected by a size-limiting bottleneck layer that explicitly restricts the number of degrees of freedom that the input data must be represented by. This structure forces the encoder to learn to compress the input data to a representation that fits through the bottleneck and the decoder to learn to reconstruct the original input from the compressed representation while minimizing information loss across the network. In our work, the encoder, $h_t = \chi(s_t; \theta_E)$, is tasked with learning the mapping from the full-state representation, $s_t \in \mathbb{R}^{d_s}$, to the manifold representation, $h_t \in \mathbb{R}^{d_h}$, and is parameterized by weights θ_E. Importantly, $d_h \ll d_s$, such that the bottleneck layer explicitly restricts the number of degrees of freedom representing our data, as our objective is to learn a reduced-order manifold representation. The decoder, $\hat{s}_t = \tilde{\chi}(h_t; \theta_D)$, is tasked with learning the mapping from the manifold representation to the full-state representation and is parameterized by weights θ_D. The autoencoder is trained to minimize the mean squared reconstruction loss (MSE) $\mathcal{L}_{AE} = \langle ||s_t - \hat{s}_t||^2 \rangle$ where $\hat{s}_t = \tilde{\chi}(\chi(s_t))$ using the snapshots of state data obtained from the previous step and $\langle \cdot \rangle$ is the average over a training batch. In our example with the KSE we also perform an intermediary change of basis from the observed state s_t to its projection onto the Principal Component Analysis basis (computed from the training data) prior to input to $\chi(s_t; \theta_E)$ and a return to the full space post output from $\tilde{\chi}(h_t; \theta_D)$. This change of basis does not reduce dimension and although NNs are universal function approximators, we found empirically in [122] that this change of basis results in a lower

reconstruction error indicating that this change of basis eases the training procedure. We identify the dimension of the finite-dimensional manifold, $d_{\mathcal{M}}$, by tracking the MSE performance of the autoencoders as we vary d_h, [122, 123]. We comment that we use a standard autoencoder here as we and others [123, 203] have found empirically that it performs well and yields a manifold representation that is conducive to forecasting tasks. [203] found that variational autoencoders and convolutional autoencoders exhibited no clear advantage over standard feedforward autoencoders for the KSE system as well as the FitzHugh-Nagumo equation. However, for systems with more complex underlying dynamics (e.g. intermittent bursting), a consideration may be warranted for different AE frameworks and latent space regularization mechanisms. For example, in [46] the authors found that systems with multiple time-scales could be better captured via a charts and atlases approach of splitting the manifold dynamics into a collection of autoencoders and time-stepping models. [208] found improvement in forecasting the Lorenz '63 system by promoting orthogonality and isotropy in the latent space while [42] promoted orthogonality with β-VAEs. However, what properties a latent space should possess for improved forecasting is still an open and active research question, we opt for simplicity.

Step 3: Learning the Actuated Dynamics along the Manifold: We next develop a neural-ODE (NODE) model of the dynamics in the manifold coordinates h by learning the RHS (vector field) of the ODE $\dot{h} = g(h_t, a_t; \theta_M)$, where θ_M denotes the neural-network parameters that are to be determined [19]. To make dynamical forecasts given a manifold state, h_t, and input control action, a_t, the evolution of the manifold state τ time units forward, $h_{t+\tau}$, can be computed for a given $g(h_t, a_t; \theta_M)$ with a standard time-integration algorithm. To train the NODE network, the same dataset generated in step 1, $[s_t, a_t, s_{t+\tau}]$ can used. Because the NODE network's purpose is to model the vector field of the dynamics in the manifold coordinates, we must first convert the training data to $[h_t, a_t, h_{t+\tau}]$ using $\chi(s_t; \theta_E)$. This data set can then be used to train the NODE network to minimize the forecasting loss: $\mathcal{L}_{NODE} = \langle ||h_{t+\tau} - \hat{h}_{t+\tau}||^2 \rangle$ where $\hat{h}_{t+\tau}$ is the forecast made by integrating the NODE with

a numerical integrator forward τ time units. In this work we use the Dormand-Prince 45 numerical scheme [38]. The gradient of the loss with respect to the network parameters, θ_M, can be obtained by either performing back-propagation via automatic differentiation through all the steps of the time integration scheme (which can be memory intensive for very large forecasts) or by the adjoint method described by Chen et al. [19]. In this work we opt for the former as we did not encounter memory issues and past works have demonstrated good agreement between both options [123]. We highlight that once training is complete, NODE forecasted trajectories, which are in the manifold coordinates, can be recovered back to the original ambient space, s, at any point using the decoder, $\tilde{\chi}(h_t; \theta_D)$, obtained in step 2.

Step 4: Learning a control strategy from the NODE-ROM with deep RL: We differentiate our DManD-RL method from typical RL with two distinctions. First the RL agent learns by interacting with the learned NODE-ROM, not the true environment. Second, during training the RL agent learns in the manifold coordinate, h, not the observable space, s. We point out that the usual RL nomenclatures for state transitions, s_{t+1} or s', are written here as $h_{t+\tau}$ in the manifold representation and $s_{t+\tau}$ in the ambient representation, to make explicit the fact that the time interval τ is in fact a parameter of the system. To train the RL agent, the NODE-ROM is first initialized with an encoded initial condition. The agent learns by interacting only with the NODE-ROM: given a state in the manifold representation, h_t, the agent attempts to map it to the optimal control action. This action is then passed back to the NODE-ROM and the evolution in h subject to the prescribed actions is obtained by integrating the NODE-ROM forward in time. In this work we aim to lower the total power consumption $D + P_f$ of the KSE dynamics, so we define the immediate reward for the RL algorithm as $r_t = -(D(t) + P_f(t))$. The algorithm seeks to maximize the long time discounted cumulative reward $R_t = \sum_{l=0}^{\infty} \gamma^l r_{t+l\tau}$ where $\gamma = 0.99$. The reward return for this manifold state-action pair ($[h_t, a_t]$) can be estimated by decoding the resulting manifold trajectory using $\tilde{\chi}(h_t; \theta_D)$ and estimating r_t, which is used to update the agent. The learning cycle then repeats. In this work we utilize the Deep Deterministic Policy Gradient (DDPG)

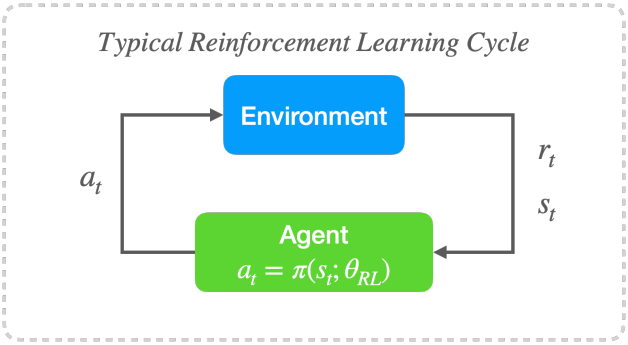

Figure 3.1: Typical reinforcement learning cycle.

RL method [120], but we emphasize that any general RL method can be used with our framework.

Step 5: Deploying and Validating the DManD-RL Control Strategy: Once RL training within the NODE-ROM is complete, the learned DManD-RL policy can be applied to the true system. As the agent was trained in the reduced manifold space, the encoder obtained in step 2 must be inserted between the environment and agent to map state observations to the manifold representation prior to input to the agent. The DManD-RL policy can then be applied in typical closed-loop fashion. If desired, new additional on-policy data can be collected to further improve the model/agent in an iterative fashion or the agent can even be simply fine-tuned with real-system training.

3.3 Results

3.3.1 Neural ODE Model Evaluation

Training data was collected by capturing snapshots of the KSE experiencing jet actuations whose magnitude and direction are randomly sampled from a uniform distribution of the allowable action range described in Section 3.2. We collect 40,000 of these transition snapshots spanning 10,000 time units for our NODE-ROM training, where $\tau = 0.25$ time units. We emphasize here that in this study τ is kept constant for simplicity but in practice τ can vary from snapshot to snapshot. Actuation signals applied during these trajectories and during

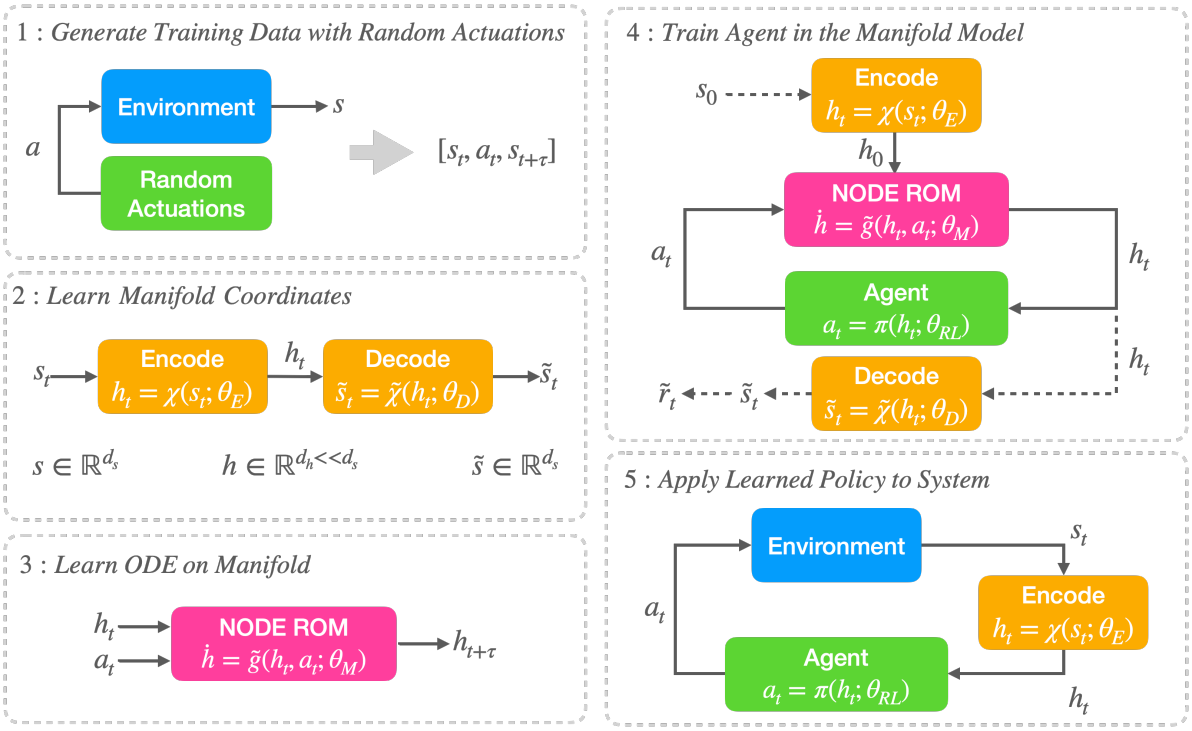

Figure 3.2: Procedure for learning a NODE-ROM from data, combining it with RL to approximate a control policy, and deploying the approximate policy back to the true system.

control are held constant for the duration of τ (i.e. zero-order hold; see Fig. 3.5a for typical actuation trajectories).

As the autoencoder serves as the mapping function to and from the manifold representation h_t, we need to estimate the number of degrees of freedom in the manifold, d_h, to accurately capture the dynamics of the system. As d_h is generally not known a priori for complex systems, we empirically estimate it by tracking the reconstruction error as a function of d_h as done by Linot and Graham [122, 123]. In this work our autoencoder networks utilize encoders and decoders each with a hidden layer of size 500 activated by sigmoid functions.

Before considering actuated data, we first apply the autoencoder analysis to the natural unactuated KSE. For the domain size $L = 22$ considered here, the manifold dimension has been determined to be $d_{\mathcal{M}} = 8$ in past studies [36, 122]. This will aid in validating our procedure and give insight into the results with the actuated system. To validate our procedure with the natural KSE, we collect 40,000 snapshots of the KSE where $a_t = 0$, to capture the unactuated KSE dynamics. Shown in Fig. 3.3a are the mean-squared errors (MSE) of

reconstruction for undercomplete autoencoder networks trained on natural, unactuated KSE snapshots for bottleneck layers of varying dimension d_h. We note that as d_h increases from 7 to 8 the MSE exhibits a sharp drop. Here, $d_h = 8$ corresponds to the minimum number of degrees of freedom required to fully represent the manifold in which the natural KSE data lives, which agrees with previous studies [36, 122]. We denote this manifold dimension of $d_h = 8$ as $d_{\mathcal{M}}$. We note that increasing d_h beyond $d_{\mathcal{M}}$ does not significantly improve reconstruction error, indicating that the data can be effectively represented in $d_h = 8$ and additional dimensions are superfluous [122].

Repeating this analysis with the snapshots collected from the randomly actuated KSE to estimate the dimensionality of the actuated dynamics, we observe a similar phenomenon. Shown in Fig. 3.3b are the mean-squared errors of reconstruction as d_h is changed for undercomplete autoencoders trained with KSE snapshots obtained from trajectories perturbed randomly by the four jet actuators. We note that compared to the Fig. 3.3a, the sharp drop in MSE is delayed to $d_h = 9$, with a lesser drop at $d_h = 11$.

To dynamically explain this, shown in Fig. 3.3c is a cartoon of the $d_{\mathcal{M}} = 8$ embedded manifold $\mathcal{M}$ on which the long time dynamics of the unactuated KSE lives, where states that begin off of the manifold are attracted exponentially to it due to dissipation. When actuations are applied, the KSE produces states and trajectories that live off of the natural attractor, effectively giving the manifold "thickness" in additional dimensions, as shown schematically in Fig. 3.3d. This "thickness" require additional degrees of freedom, i.e. increased $d_{\mathcal{M}}$ from the original system, to accurately capture the dynamics.

To further support this view, shown in Fig. 3.4 is the power spectral density of the unactuated and actuated data sets $u(t)$ used to train the autoencoders shown in Fig. 3.3. We note that in the presence of actuations the data exhibits a broadening of the high-wavenumber tail compared to the unactuated data. Physically, this indicates that there are additional high-wavenumber spatial features in the actuated data set, which require additional degrees of freedom to accurately capture. Finally, we highlight in Fig. 3.3a and

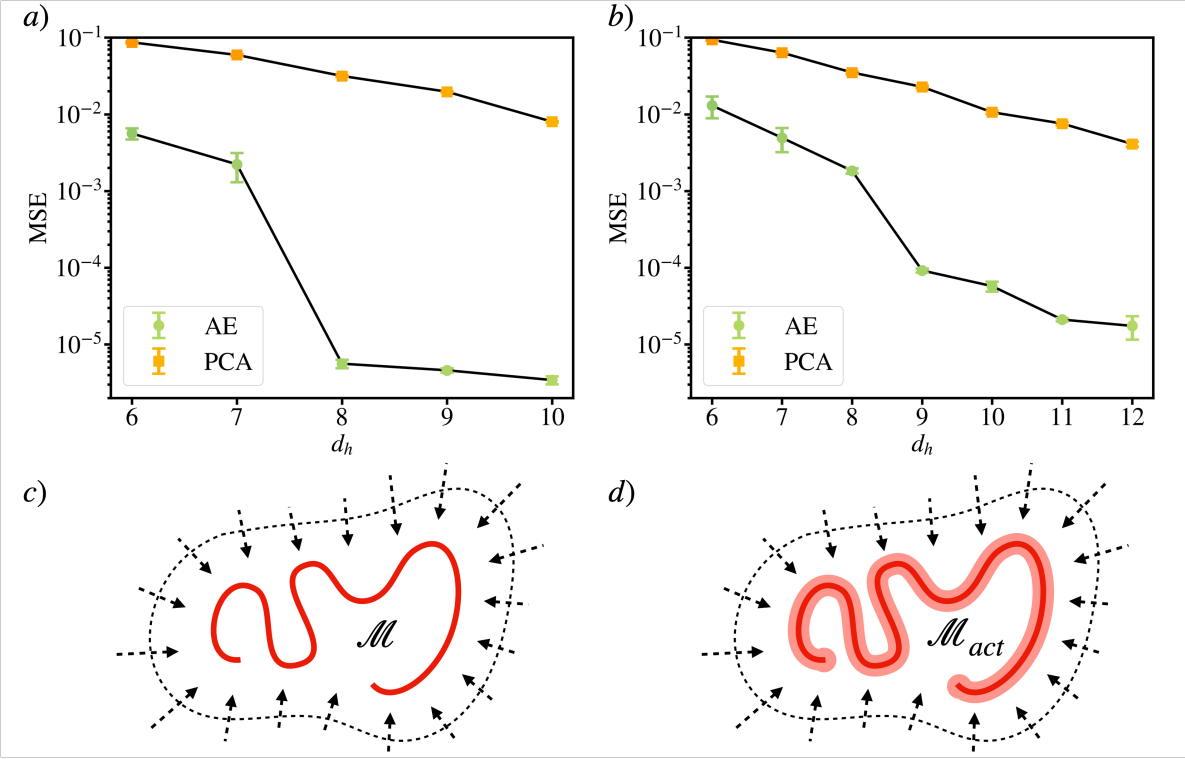

Figure 3.3: Mean squared reconstruction error for autoencoders trained on a) unactuated and b) actuated KSE data vs. manifold representation dimension, d_h. Cartoon illustrations of the inertial manifolds in which the data lives on for the c) unactuated and d) actuated system.

Fig. 3.3b that our autoencoders outperform dimensionality reduction using only Principal Component Analysis in reconstruction error by several orders of magnitude once $d_{\mathcal{M}}$ has been reached.

The autoencoder MSE landscape of the actuated KSE data suggests that a manifold representation of $d_h = 12$ should be sufficient to capture the dynamics of the controlled KSE. Using the $\chi, \tilde{\chi}$ mappings learned by the $d_h = 12$ autoencoder, a NODE network with two hidden layers of size 200, 200 with sigmoid activation was trained to model the actuated dynamics in the $d_h = 12$ manifold. This NODE was trained using the same data used to train the autoencoder by simply converting the collected data of $[s_t, a_t, a_{t+\tau}]$ to $[h_t, a_t, h_{t+\tau}]$ with $h_t = \chi(s_t; \theta_E)$.

To demonstrate the learned NODE-ROM's predictive capability, shown in Fig. 3.5c and Fig. 3.5d are two example KSE trajectories experiencing random jet actuation signal sequences visualized in Fig. 3.5a, Fig. 3.5b, respectively. We also show in Fig. 3.5e and

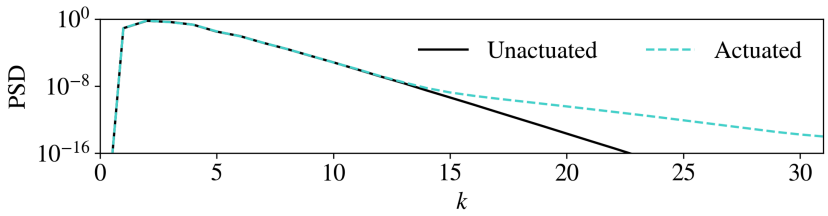

Figure 3.4: Power spectral density vs. wavenumber of unactuated and actuated data.

Fig. 3.5f the decoded $d_h = 12$ trajectory forecast by the NODE-ROM beginning from the same initial condition and following the same actuation signal sequence as its ground truth trajectory, Fig. 3.5c, Fig. 3.5d, respectively. We note that the forecasted trajectories agree qualitatively with the ground truth for about 20-30 times units or 1-1.5 Lyapunov times of the natural system, shown in Fig. 3.5g, Fig. 3.5h. Finally, we show in Fig. 3.5i, Fig. 3.5j the 12 dimensional manifold representation of these trajectories, h. We highlight that the learned representation evolves smoothly in time.

To quantitively compare the ensemble performance of the NODE-ROM and the actuated KSE, an ensemble of 50 actuated forecast/ground truth pairs of trajectories (where the random actuation sequence and initial conditions between each pair are the same) was used to compute the spatial and temporal autocorrelation, shown in Fig. 3.6a and Fig. 3.6b, respectively. We note that the NODE-ROM accurately captures the spatial autocorrelation of the actuated KSE, while the temporal autocorrelation exhibits good agreement with a slight temporal dilation. These results indicate that the NODE-ROM is accurately capturing the distribution of features of the actuated KSE in both space and time.

As the NODE-ROM was trained with only transition snapshots experiencing random actuations, a natural question is: how well does our model capture the underlying natural dynamics, i.e. with $a_t = 0$? Shown in Fig.3.7a and Fig.3.7b are two example natural KSE trajectories where $a_t = 0$. Accompanying the unactuated ground truth trajectories shown in Fig.3.7a and Fig.3.7b are the decoded $d_h = 12$ trajectories forecasted by the same NODE-ROM that was trained on actuated data beginning from the same initial conditions and with zero actuation signal input, shown in Fig.3.7c and 3.7d, respectively. We note that the

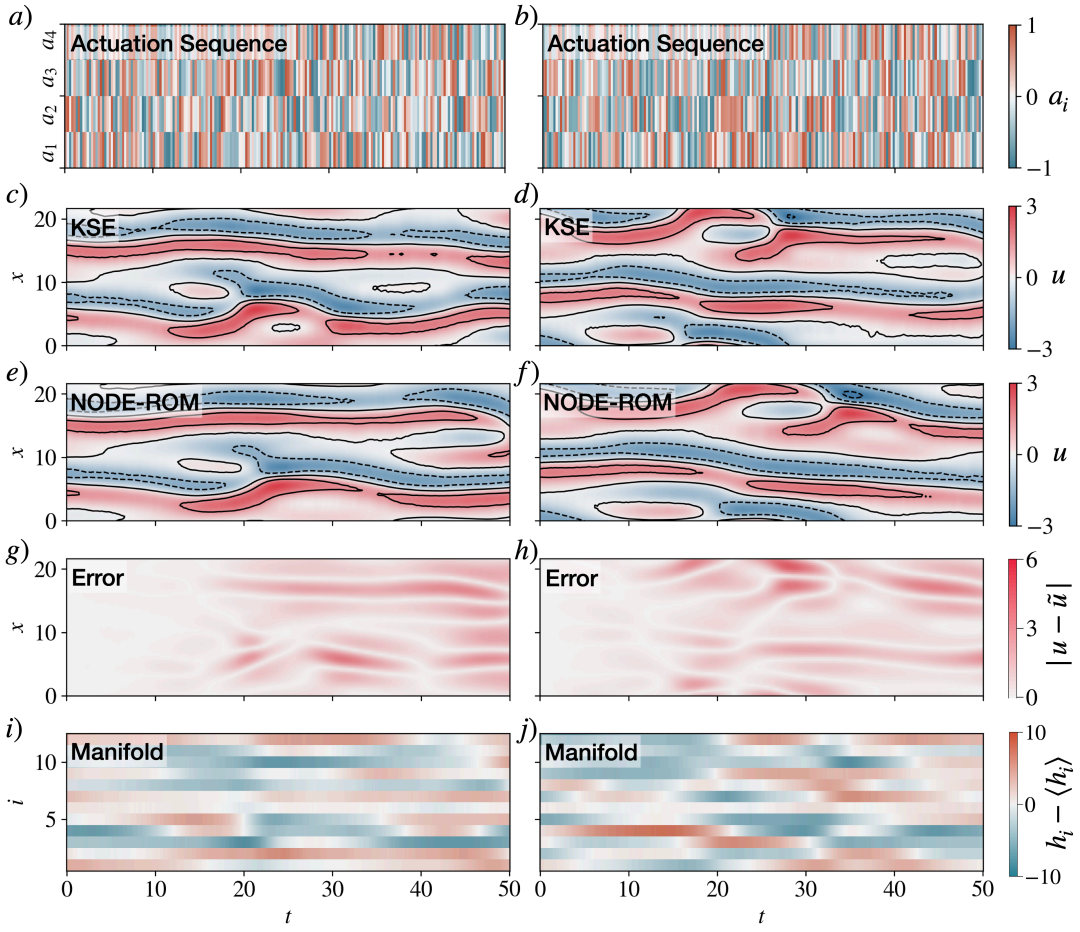

Figure 3.5: Left column: (a) random actuation sequences $a_i(t)$ (c) ground truth KSE trajectory starting from a random initial condition following actuation sequences in (a), (e) the decoded NODE-ROM trajectory following actuation sequence (a) and the same initial condition in (c). The absolute error difference is shown in (g) while the manifold representation, h, is shown in (i). A second example is shown in the right column: (b), (d), (f), (h) and (j), respectively.

forecasted trajectories agree qualitatively with the ground truth for about 1-1.5 Lyapunov times. We again assess the spatial and temporal autocorrelations between an ensemble of 50 pairs of ground truth trajectories and NODE-ROM forecasts, shown in Fig. 3.6c and 3.6d, respectively. We note that the ROM matches the spatial autocorrelation of the KSE very well while the temporal autocorrelation reveals that the ROM exhibits a more pronounced but still quantitatively small temporal dilation. We emphasize that despite the training data for the ROMs were obtained from randomly actuated trajectories and that there is no substantial unactuated data, the ROMs still recovers the unactuated dynamics well.

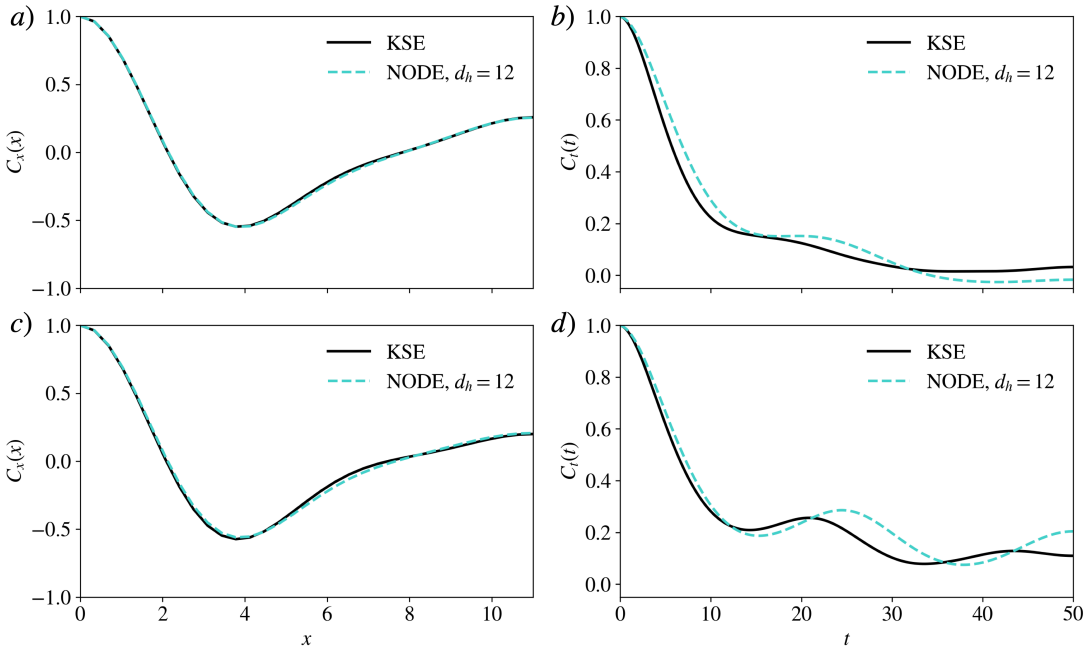

Figure 3.6: a) Spatial and b) temporal autocorrelation computed from an ensemble of trajectories experiencing random jet actuations. NODE-ROM ($d_h = 12$) forecasts were produced using the same initial conditions and jet actuation sequences as each respective trajectory produced from the true KSE. c) Spatial and d) temporal autocorrelation computed from an ensemble of trajectories with actuations set to zero. NODE-ROM ($d_h = 12$) forecasts were produced using the same initial conditions as each respective trajectory produced from the true KSE.

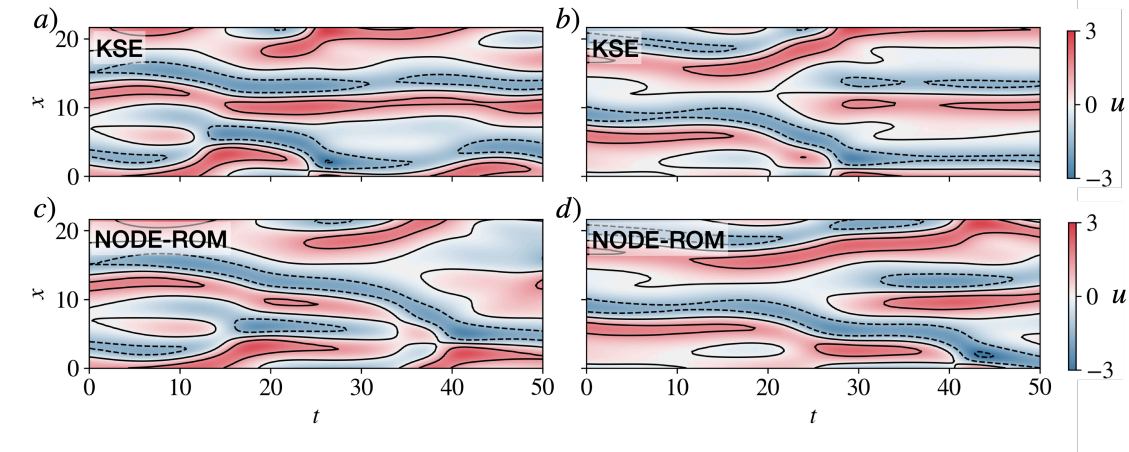

Figure 3.7: Example unactuated trajectories of the KSE in a) and b) and their corresponding decoded NODE-ROM ($d_h = 12$) forecasts starting from the same initial conditions in c) and d), respectively.

3.3.2 Model-Based Control Performance

With this data-driven NODE-ROM, a control agent was trained by interacting only with the NODE-ROM in the manifold space with r_t estimated from the $s = \tilde{\chi}(h; \theta_D)$ decoded state. We set the NODE-ROM model transition time to be $\tau = 0.25$ for all experiments. The control agent was trained with 1000 episodes of 100 time units long (i.e. 400 transitions per episode), with each episode beginning from a random on-attractor initial condition of the natural, i.e. unforced, KSE. Jet actuations implemented by the control agent, a_t, were maintained constantly from s_t to $s_{t+\tau}$. In this work the DDPG actor and critic networks utilized ReLU activated hidden layers of size 128 and 64, respectively, followed by tanh and linear activations to the outputs of size 4 and 1, respectively.

To assess the performance of our DManD-RL policy, the learned control agent was applied to the NODE-ROM, with an example controlled trajectory shown in Fig. 3.8a. We note that after a brief control transient, the control agent navigates the NODE-ROM to an equilibrium(steady) state and stabilizes it. The quantities targeted for minimization, D and P_f, estimated from the predicted trajectory $u(t)$, are shown in Figure 3.8c, revealing that this equilibrium exhibits dissipation much lower than the natural unactuated dynamics. To assess how well this DManD-RL control policy transfers to the original KSE (i.e. the true system), the same policy is applied to the true KSE with the same initial condition, as shown in Fig. 3.8b. We note that the controlled trajectory in the KSE yields not only quantitatively similar transient behavior but also the same low-dissipation equilibrium state as was targeted in the NODE-ROM. The transient behaviors between the two are structurally very similar, although the NODE-ROM displays slightly less strongly damped oscillations as it drives the trajectory to the steady state. The values of D and P_f computed from the true system, shown in Fig. 3.8d, are nearly identical to that of the NODE-ROM in Fig. 3.8c.

To demonstrate the robustness of the DManD-RL policy, shown in Fig. 3.8e are the dissipation trajectories of the true KSE beginning from 15 randomly sampled test initial conditions that the DManD-RL control agent has not seen before. We highlight that the

control agent is able to consistently navigate the system to the same low-dissipation state within ~ 150 time units, with one initial condition requiring ~ 200 time units to converge. Finally we note that although the RL training horizons were only 100 time units long, the control agent is able to generalize to achieve and maintain control well beyond the horizon it was trained in.

Here we emphasize that the DManD-RL policy drives the dynamics to an equilibrium state in both the NODE-ROM and the true KSE, indicating that not only does the NODE-ROM capture this state, but it captures the dynamics leading to it accurately enough such that the RL agent could discover it during training and exploit it in a manner that still translates to the original system. We further emphasize that both the NODE-ROM and agent were never explicitly informed of this low-dissipation state's existence. Finally, we highlight that the discovered low-dissipation state is an unstable state that is stabilized by the control agent. If control is removed, the system returns to the natural chaotic dynamics.

These observations indicate that the RL policy trained on the model transfers very well to the true system. We attribute this performance to the fact that both the NODE-ROM and RL agent operate in Markovian fashion. The agent is not forced to act out a series of actions based on a forecasted trajectory but rather it is reassessing at every action point in the true system. Even if the model has slight inaccuracies, so long as the modeled dynamics are reasonably accurate this does not matter once the agent makes its new state observation. An example of the policy still succeeding far from the training horizon can be seen in Fig. 3.8e.

Returning to the dynamical significance of the low-dissipation equilibrium state discovered and stabilized by the RL agent, a continuation in mean forcing magnitude was performed. To do so, we Newton-solved for equilibrium solutions to the KSE starting with the discovered equilibrium state while gradually decreasing the magnitude of the mean actuation profile to zero, as was done in [219]. Solutions identified by this continuation in forcing magnitude are shown in Fig. 3.10, which reveals that equilibrium state captured by the NODE-ROM and discovered by the RL agent is connected to a known existing solution

of the KSE known as $E1$ [32]; we obtained a similar result with an RL agent trained on interactions with the full system [219]. A similar observation was made for RL control of 2D bluff body flow [117].

We speculate that in systems with complex dynamics, the discovery and stabilization of desirable underlying equilibrium solutions (or other recurrent saddle-point solutions such as unstable periodic orbits) of the system may be a fairly general feature of RL flow control approaches that aim to minimize dissipation while penalizing control action; as was shown in [117, 219]. The nonlinear and exploratory nature of RL algorithms facilitates the discovery of such solutions, and since the dynamics are slow near these solutions, little control action should be required to keep trajectories near them. This is a promising outlook for RL as the dynamics of even more complex chaotic systems, such as turbulent flows governed by the Navier-Stokes equations, are also known to be structured around such solutions [66].

To highlight the importance of the type of data-driven model, we compare to Dynamic Mode Decomposition (DMD), a common data-driven method that has been applied to fluid flows and dynamical systems. Recently, Qin et al. [152] demonstrated active flow control of 2D cylinder flow with a reward signal targeting the minimization of DMD mode amplitudes. Here we specifically investigate the performance of using DMDc [150], the extension of DMD to account for control input, as an alternate data-driven model of the KSE dynamics for RL policy learning. The DMDc model takes the following form,

$$s_{t+\tau} = \mathbf{A}s_t + \mathbf{B}a_t, \tag{3.7}$$

where $\mathbf{A}$ is the state transition matrix and $\mathbf{B}$ is the control input matrix. Using the same training data as our NODE-ROM, $[s_t, a_t, s_{t+\tau}]$, , we obtained a full state DMDc model by simultaneously fitting the two matrices with the $[s_t, a_t, s_{t+\tau}]$ data as described in [150]. This model was then used as the environment for the RL policy training.

Shown in Fig. 3.9a is the DMDc-based RL control agent applied to the DMDc model it

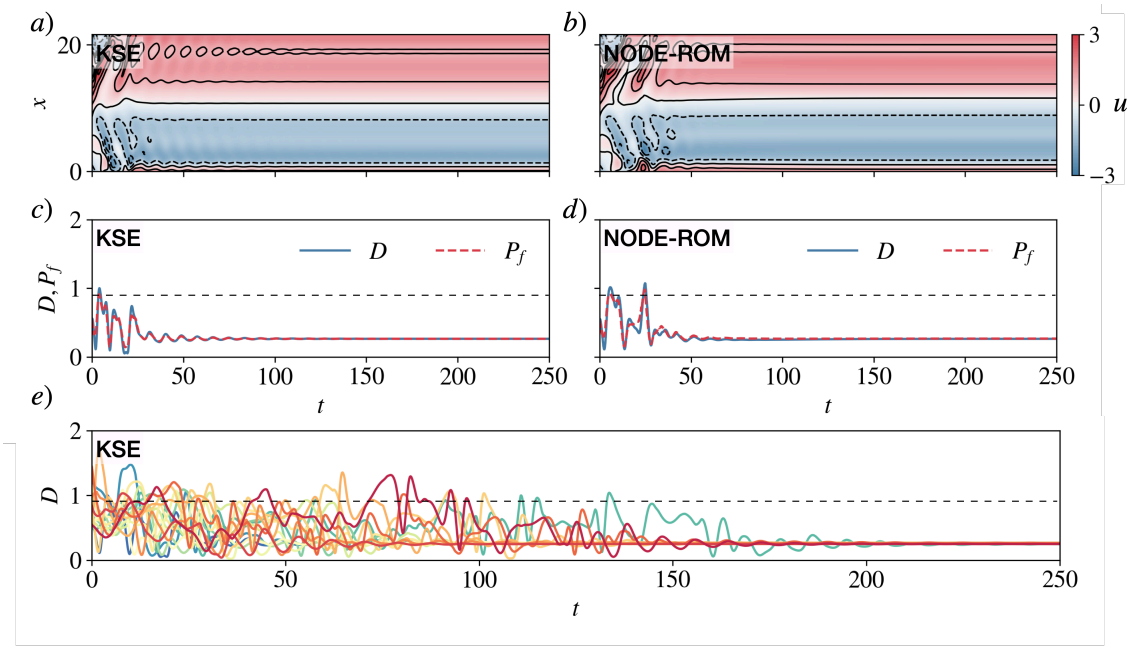

Figure 3.8: ROM-based RL agent applied to the same initial condition in the a) true KSE and b) data-driven reduced-order model (decoded, $d_h = 12$). The corresponding invariant quantities of dissipation and total input power for the c) true KSE and d) learned reduced-order model. The dashed black line represents the system average of the natural KSE dynamics. e) Controlled dissipation trajectories of the true KSE beginning from 15 randomly sampled test initial conditions of the KSE.

was trained in. We show in Fig. 3.9c that the agent is able to rapidly minimize the DMDc model values of D, P_f. We highlight that the DMDc-based strategy accomplishes this by exploiting the unphysical dynamics of the DMDc model. To elaborate, shown in Fig. 3.9b is the same strategy applied to the true KSE with the same initial condition as the DMDc trajectory in Fig. 3.9a. The DMDc-based RL strategy fails to minimize the control objective when applied to the true KSE. Furthermore, the strategy even results in performance that is even worse than the time-averaged uncontrolled system, which can be seen in the above average D, P_f values at long times in Fig. 3.9d, where the controller appears to get trapped in a high energy region of the KSE. This mismatch in performance is due to the inability of the linear model to describe the nonlinear dynamics of the KSE.

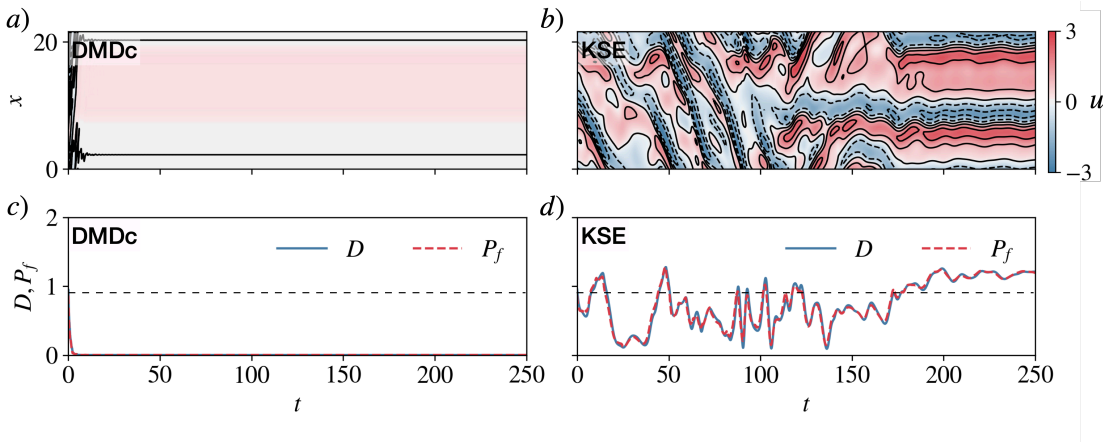

Figure 3.9: DMDc control policy applied to the same initial condition in the (a) DMDc model and (b) True KSE. Corresponding dissipation and total input power for the (c) DMDc model and (d) KSE. The dashed black line represents the average of the natural KSE dynamics.

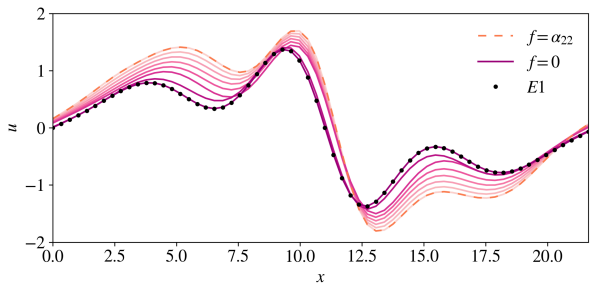

Figure 3.10: Forcing continuation from the forced equilibrium state (under forcing $f = \alpha_{22}$) discovered by NODE-ROM based RL policy (orange, dashed) to the unactuated KSE system (purple). The known equilibrium E1 of the KSE system is also provided (dots).

3.4 Summary

In this chapter, we introduce our NODE-ROM-Based RL method, DManD-RL, that maintains an end-to-end data-driven method of learning control policies from a limited data set. The governing equations and control term do not need to be known or specified–we only assume that the system's actuated dynamics can be represented by some governing system of ODEs in the manifold coordinate. Furthermore, we do not impose any structural relationship between the control input and the dynamics of the systems as previous methods have done in the past. This is advantageous for many complex flow control systems as the relationship between the dynamics and the control inputs are not simple linear relationships [9, 54]. We exploit the notion that many nominally high-dimensional systems actually display low-dimensional dynamics and aim to capture and model these actuated dynamics from data using a combination of autoencoders and Neural ODEs. Then, with deep RL, we extract control policies from our data-driven ROMs. Importantly, our method does not require RL to directly interact with the target system nor does it require great modification of standard RL algorithms.

Our method identifies the lower-dimensional manifold the actuated dynamics live on with autoencoders and models the dynamics with neural ODEs, which are capable of making predictions for arbitrary time intervals [19] and have been demonstrated to produce good predictions of spatiotemporal chaotic systems [123]. Neural ODEs are also a natural formulation and deep architecture for the systems we are attempting to model and control. Our method also respects the Markovian nature underlying many RL frameworks as well as those of our target dynamical systems, unlike other common data-driven methods such as reservoir computing and recurrent neural networks. The use of high-fidelity low-dimensional models allows for faster interaction simulations (or even running multiple surrogate environments in parallel) compared to costly simulations or experiments.

By using a more informative and compact state space representation than a high-dimensional sensed state, the RL policy training load is lessened. In naive applications of RL, a portion

of the agent's network capacity is exhausted in learning to transform the raw inputs to more useful internal representations [75]. Similar to Ha and Schmidhuber [75], in our method, we also explicitly separate the agent's learning of the control policy and the task of learning a useful state representation into two discrete tasks.

We apply our DManD-RL RL method to the KSE, a proxy system for turbulent flows that exhibits rich spatiotemporal chaotic dynamics. From a limited data set generated from trajectories of the KSE under random actuations, we are able to find a reduced-order mapping between the full state and the reduced low-order dynamics of the actuated KSE, as well as to successfully model the dynamics in this reduced space. We find that our NODE-ROM has good forecasting ability and captures ensemble characteristics of the KSE well for not only actuated predictions but also unactuated predictions, for which it was not given training data. In this work, it was found that NODE-ROM training data obtained from snapshots of the KSE experiencing random jet actuations was sufficient. With even more complex systems, it may be possible that this sampling approach is insufficient. If necessary, the NODE-ROM can be iteratively improved by applying the learned control strategy to the true system, collecting additional data, and fine-tuning the NODE-ROM with the new data. Using the updated NODE-ROM, the RL agent can then be updated. Together the control agent and model can be improved in a cyclic training fashion. We highlight that our model could be alternatively trained and used in the "Dyna"-style [186] for improved online learning. We comment that, in the scope of this article, the manifold coordinate is learned from the full state. However, in many systems of interest, this information may not be readily available. In the absence of full-state observations, a combination of sensor measurements and time-delays should suffice so long as they satisfy Whitney's Embedding Theorem. While we demonstrate in this article that this framework can achieve similar performance compared to past studies utilizing direct applications of RL [219], other systems of interest may exhibit much higher dimensional action spaces or greater dynamic complexity, which may require smarter data collection and successive model updates.

Finally, we acknowledge here that although we do not obtain substantial speed up in per-step computation time between the semi-implicit KSE solver and the NODE-ROM, we emphasize that for flow systems of increasing complexity and Re, the time to solve $\mathcal{O}(10^6)$ or more degrees of freedom in a direct numerical simulation is much greater than that of solving a system of ODEs $\mathcal{O}(10^1 - 10^2)$. For example, in [220] we show preliminary results of this method on turbulent plane Couette flow at $\mathrm{Re} = 400$ in a minimal flow unit [77]. In that work, we accurately modeled the dynamics with $\mathcal{O}(10)$ degrees of freedom resulting in simulations running several orders of magnitude faster. Applying reinforcement learning to the full system, with jet actuators on one wall, yields a control policy that can relaminarize the flow. However, learning this policy took 3+ weeks. With the DManD model, an equivalently effective policy was learned in hours. Aside from the computational speed up, we also emphasize that our method greatly reduces the data generation burden, as we successfully obtained a control strategy for the KSE using only 40,000 transition snapshots, while a naive application of RL directly on the KSE required generating approximately 1.6 million transition snapshots [219].

Utilizing this NODE-ROM in place of the environment, the RL agent is able to discover a low-dissipation equilibrium state and learns to exploit it to minimize D, P_f. When the DManD-RL control policy is deployed to the true KSE, we observe that not only are the same equilibrium states targeted, but the performance is nearly indistinguishable from our learned NODE-ROM. This indicates that not only does the NODE-ROM capture the existence of the forced equilibrium, but it also captures the dynamics sufficiently well such that the agent could find it during training and exploit it in a manner that still translates in the original system. We emphasize that we accomplished this with a 12-dimensional NODE-ROM while the full state is 64- dimensional. A continuation in the magnitude of the forcing profile reveals that the RL discovered equilibrium state is connected to an existing equilibrium solution of natural KSE. The naturally occurring RL optimization about underlying solutions of the system has been observed in the KSE [219] as well as bluff-body flows [117], which

is promising, as more complex dissipative systems, such as the turbulent dynamics of the Navier-Stokes equations, are also known to be organized about various types of invariant solutions [66].

4

Model-based reinforcement learning for turbulent Couette flow [1]

The high dimensionality and complex dynamics of turbulent flows remain an obstacle to the discovery and implementation of control strategies. Deep reinforcement learning (RL) is a promising avenue for overcoming these obstacles, but requires a training phase in which the RL agent iteratively interacts with the flow environment to learn a control policy, which can be prohibitively expensive when the environment involves slow experiments or large-scale simulations. We overcome this challenge using a framework we call "DManD-RL" (data-driven manifold dynamics-RL), which generates a data-driven low-dimensional model of our system that we use for RL training. With this approach, we seek to minimize drag in a direct numerical simulation (DNS) of a turbulent minimal flow unit of plane Couette flow at $\mathrm{Re} = 400$ using two slot jets on one wall. We obtain, from DNS data with $\mathcal{O}(10^5)$ degrees of freedom, a 25-dimensional DManD model of the dynamics by combining an autoencoder and neural ordinary differential equation. Using this model as the environment, we train an RL control agent, yielding a 440-fold speedup over training on the DNS, with equivalent control performance. The agent learns a policy that laminarizes 84% of unseen DNS test trajectories

[1] The text of this chapter is adapted from the publication by A. J. Linot, K. Zeng and M. D. Graham *The International Journal of Heat and Fluid Flow* 101, 2023

within 900 time units, significantly outperforming classical opposition control (58%), despite the actuation authority being much more restricted. The agent often achieves laminarization through a counterintuitive strategy that drives the formation of two low-speed streaks, with a spanwise wavelength that is too small to be self-sustaining. The agent demonstrates the same performance when we limit observations to wall shear rate.

4.1 Introduction

Energy loss due to turbulent drag is ubiquitous in many industrial and commercial processes, ranging from air flowing over a plane wing, a ship in the ocean, or oil pumped through a pipe. In total, turbulent drag accounts for 25% of the energy used in industry and commerce, resulting in 5% of all man-made CO_2 emissions [95]. Even small reductions in this drag can yield massive savings in energy, which has long motivated the search for better flow control strategies.

Many control types for reducing turbulent drag exist, including, but not limited to, polymer/surfactant drag reduction [65, 200], riblets [182], wall oscillations [155], plasma actuators [25], and synthetic jets [61]. Due to the complexity of reducing drag, it has been most common to apply these control methods in an open-loop manner where the control policy at any given time is independent of the flow state [24]. However, application of feedback control on a turbulent system could yield far better performance in controlling drag.

Unfortunately, the complexity of the problem has typically limited applications of feedback control to methods based on heuristics. A well-studied heuristic method is opposition control [22]. Here the wall-normal velocity at the wall is set to have the opposite sign as the wall-normal velocity at some detection plane in the channel, a straightforward actuation in simulations. This method has been applied in simulations [22, 28, 78, 89] and experiments (with some modifications) [20, 162], and extensions exist to use just wall observations

[109, 148]. When heuristics are replaced with methods from optimal control theory, like model predictive control (MPC), the drag reduction far outperforms opposition control [6] while using the same actuation scheme. However, the real-time implementation of MPC on DNS still remains infeasible because it involves solving the DNS forward over a time horizon (preferably a long one) and then solving an adjoint backwards in time for every actuation [6].

A potential approach to overcome the high computational cost of real-time optimization of a control strategy is deep reinforcement learning (RL) [186]. Deep RL gained significant traction when it was used to defeat the best professional players in GO [178], DOTA II [144], and Starcraft II [198], in addition to the best engines in GO, Chess, and Shogi [179]. In deep RL, a neural network (NN) control agent is trained through iterative interactions with the environment (i.e. the system to be controlled) to maximize a scalar total reward (i.e. control objective) that includes present as well as discounted future reward values. Once trained, the control agent can be deployed in real time without the need for online optimization.

In recent years, RL has been applied in fluids simulations to reduce the drag experienced in flow around a cylinder [116, 156, 194], to optimize jets on an airfoil [207], and to find efficient swimming strategies [196]. RL has even been applied to experimental flow systems [44]. Recently multi-agent deep RL has been explored for the control of pressure-driven turbulent channel flow [72, 183] in a problem formulation similar to opposition control [22]. In these works, an RL policy is trained to map local detection plane observables to a wall-normal velocity response at the walls to reduce drag. Notably, the same RL policy is locally implemented at each wall grid point. We differentiate the control problem addressed from the previously mentioned works in that we limit the control authority to just two spatially localized jets on a single wall, with a zero-net-flux constraint, as opposed to full spatial control of both walls. We feel that this is much closer to experimental realizability than an approach with control authority everywhere on the wall. For a more thorough review of the application of RL applied to fluid mechanics we refer the reader to Viquerat et al. [199] and

Vignon et al. [197].

In these active flow control problems, deep RL possesses the advantageous property of being completely data-driven, allowing it to discover novel and nontrivial control strategies in complex systems from just data alone without the need to analytically derive or hard-code system properties into the method. However, the training portion of RL is a major bottleneck, requiring a tremendous number of interactions with the target environment to find an approximately optimal policy [40]. Practically speaking, this can correspond to running an enormous number of high-resolution simulations or flow experiments, both of which may be prohibitively expensive.

In the present work, we apply RL to control a minimal flow unit (MFU) (the smallest domain that sustains turbulence) [96] of plane Couette flow at $Re = 400$ using a pair of streamwise-aligned slot jets at one wall, with a zero-net-flux constraint. Therefore there is only one degree of freedom for actuation. We select this system because the unactuated flow isolates the self-sustaining regeneration cycle of wall-bounded turbulence [77, 90]. This case is well-studied for tasks such as reduced-order modeling [55, 125, 138, 205], finding invariant solutions [57, 201], and applying opposition control [89].

In order to overcome the high computational cost of RL training in this environment, in this work we replace the high-resolution simulation with an accurate low-dimensional surrogate model, aiming to dramatically reduce the time required to train the control policy. We showed in [221] that this data-driven model-based RL approach, which we refer to as "Data-Driven Manifold Dynamics" RL (DManD-RL), works well for controlling spatiotemporal chaotic dynamics in the Kuramoto-Sivashinksy Equation. For further discussion on the various types of model-based RL, we refer the reader to Zeng et al. [221]. In Sec. 4.2 we introduce the control environment and the DManD-RL framework. Then, in Sec. 5.3.1 we describe the data used for training the DManD model, the performance of the model, and the results of applying RL to the DManD model and to the DNS environment. Finally, we conclude in Sec. 5.4 with a summary of the key results.

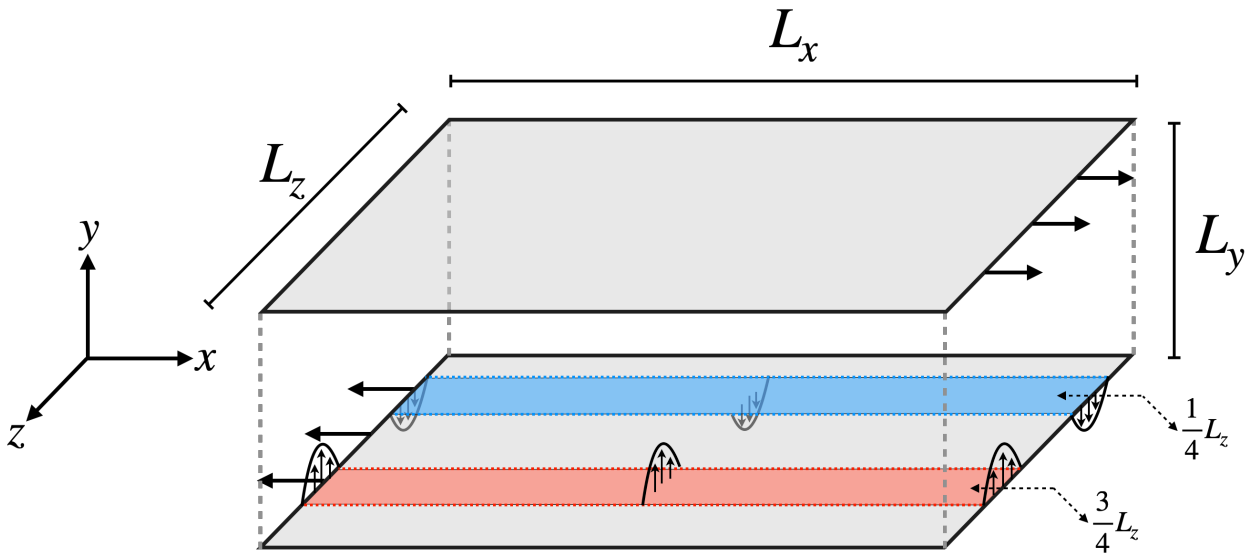

Figure 4.1: Schematic of the Couette flow domain with two slot jets on one wall.

4.2 Framework

4.2.1 Navier-Stokes Equation with Slot Jets

The environment we consider is a direct numerical simulation (DNS) of the Navier-Stokes Equations (NSE)

$$\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} = -\nabla p + \mathrm{Re}^{-1} \nabla^2 \mathbf{u}, \quad \nabla \cdot \mathbf{u} = 0. \tag{4.1}$$

The velocities in the streamwise $x \in [0, L_x]$, wall-normal $y \in [-L_y/2, L_y/2]$, and spanwise $z \in [0, L_z]$ directions are defined as $\mathbf{u} = [u_x, u_y, u_z]$, and the pressure is p. Here we have nondimensionalized velocity by the speed U of the walls, length by the channel half-height ($h = L_y/2$), time with h/U and pressure with ρU^2, where ρ is the fluid density. The Reynolds number is $\mathrm{Re} = Uh/\nu$, where ν is the kinematic viscosity. The boundary conditions for this setup are periodic in x and z ($\mathbf{u}(0, y, z) = \mathbf{u}(L_x, y, z), \mathbf{u}(x, y, 0) = \mathbf{u}(x, y, L_z)$), no-slip boundary conditions at the walls ($u_x(x, \pm L_y/2, z) = \pm 1, u_z(x, \pm L_y/2, z) = 0$), no penetration at the top wall ($u_y(x, L_y/2, z) = 0$), and finally, the actuation on the bottom wall ($u_y(x, -L_y/2, z) = f_a(x, z)$), as we now describe.

The actuation on the bottom wall is in the form of two slot jets that are Gaussian in z

and travel the length of the channel:

$$u_y(x, -L_y/2, z) = f_a(x, z) = a(t)V_{\max}\left(\exp\left(-\frac{(z - L_z/4)^2}{2\sigma^2}\right) - \exp\left(-\frac{(z - 3L_z/4)^2}{2\sigma^2}\right)\right). \tag{4.2}$$

We set $\sigma \approx 0.16$ so that the jets act "locally", and the velocity of the jet is dictated by $a(t)V_{\max}$, where $a(t) \in [-1, 1]$ is the instantaneous actuation amplitude scaled by a maximum velocity $V_{\max} = 0.05$. For perspective, the root-mean-squared wall-normal velocity at the channel centerline for turbulent unactuated flow is ~ 0.063. We chose this small velocity to evaluate how the agent performs with limited control authority. In Fig. 4.1 we show a schematic illustrating this system. Due to only having two jets the zero-net-flux constraint is easily satisfied in this case. However, as it may be useful for practitioners, we present a method for achieving a zero-net-flux constraint with an arbitrary number of jets in the Appendix.

The complexity of the flow increases as the Reynolds number and the domain size L_x and L_z increase. Here we chose the same setup as Hamilton et al. [77], $\mathrm{Re} = 400$ and $[L_x, L_y, L_z] = [1.75\pi, 2, 1.2\pi]$. These parameters isolate the "self-sustaining process" (SSP) that drives wall-bounded turbulence. In the SSP, low-speed streaks that have been lifted from the wall become wavy, this waviness leads to the breakdown of the streaks, generating streamwise rolls, and, finally, these rolls lift low-speed fluid off the wall to regenerate streaks, completing the cycle. By working in this well-studied domain that is dominated by the SSP, we can better identify the means by which a control strategy can disrupt or suppress this process.

In this work, the control strategy is to minimize the turbulent drag averaged between both walls

$$D = \frac{1}{2}\int_0^{L_x}\int_0^{L_z}\left(\left.\frac{\partial u_x}{\partial y}\right|_{y=1} - 1\right) + \left(\left.\frac{\partial u_x}{\partial y}\right|_{y=-1} - 1\right) dxdz, \tag{4.3}$$

subject to a quadratic penalty on actuation amplitude a. (If the relation between the pressure drop and actuation velocity for pumping fluid into/out of the domain is linear, then this

penalty is proportional to the power consumption of the actuation.) Further details are described in Sec. 4.2.2. We report drag in this fashion because this quantity goes to 0 when the flow laminarizes.

We simulate the flow using a Fourier-Chebyshev pseudo-spectral code we implemented in Python [127], which is based on the *Channelflow* code developed by Gibson et al. ([56, 58]). Linear terms are treated implicitly and the nonlinear term explicitly. The specific time integration schemes we use are the multistage SMRK2 scheme [184] for the first two timesteps after every actuation, and the multistep Adams-Bashforth Backward-Differentiation 3 scheme [149] until the next actuation. The multistep scheme is more computationally efficient, but, because actuations change instantaneously, using previous steps with the incorrect boundary condition would lead to incorrect results. For all trials we evolve solutions forward using $\Delta t = 0.02$ on a grid of $[N_x, N_y, N_z] = [32, 35, 32]$ in x, y, and z from random divergence-free initial conditions that we evolve forward 100 time units so initial conditions are near the turbulent attractor.

While most of this approach is standard, here we include some details on the simulation procedure to highlight explicitly how we set the jet actuation boundary condition. At each time step, the approach involves solving the expression

$$\mathrm{Re}^{-1}\frac{d^2\hat{\mathbf{u}}^{i+1}_{k_x,k_z}}{dy^2} - \lambda\hat{\mathbf{u}}^{i+1}_{k_x,k_z} - \hat{\nabla}\hat{p}^{i+1}_{k_x,k_z} = -\hat{\mathbf{R}}^{i}_{k_x,k_z}, \tag{4.4}$$

where i is the timestep and $\hat{\cdot} = \mathcal{F}_{x,z}(\cdot)$ denotes the Fourier transform in x and z. The variable λ includes the timestep Δt and the x and z components of the diffusive term and $\mathbf{R}$ encompasses all the remaining explicit terms (for a multistep method this includes $\hat{\mathbf{u}}_{k_x,k_z}$ multiple steps back). We refer the reader to [56] for a more detailed discussion. Upon taking the divergence of Eq. 4.4, and accounting for incompressibility, we isolate the problem down to 4 sets of one-dimensional Helmholtz equations (for conciseness we suppress indices k_x, k_z, and i):

$$\mathrm{Re}^{-1}\frac{d^2\hat{u}_x}{dy^2} - \lambda\hat{u}_x - \frac{2\pi i k_x}{L_x}\hat{p} = -\hat{R}_x \qquad \hat{u}_x(\pm 1) = \pm\delta_{k_x,0}\delta_{k_z,0} \tag{4.5}$$

$$\mathrm{Re}^{-1}\frac{d^2\hat{u}_z}{dy^2} - \lambda\hat{u}_z - \frac{2\pi i k_z}{L_z}\hat{p} = -\hat{R}_z \qquad \hat{u}_z(\pm 1) = 0 \tag{4.6}$$

$$\mathrm{Re}^{-1}\frac{d^2\hat{u}_y}{dy^2} - \lambda\hat{u}_y - \frac{d\hat{p}}{dy} = -\hat{R}_y \qquad \hat{u}_y(-1) = \mathcal{F}_{x,z}(f_a), \quad \hat{u}_y(1) = 0 \tag{4.7}$$

$$\frac{d^2\hat{p}}{dy^2} - 4\pi^2\left(\frac{k_x^2}{L_x^2} + \frac{k_z^2}{L_z^2}\right)\hat{p} = \hat{\nabla}\cdot\hat{\mathbf{R}} \qquad \frac{d\hat{u}_y}{dy}(\pm 1) = 0. \tag{4.8}$$

These equations can be solved for every wavenumber pair k_x and k_z. The challenge in solving these equations is due to the coupling in Eq. 4.7 and Eq. 4.8. The pressure is coupled to the wall-normal velocity because an explicit boundary condition is unknown. Instead, from incompressibility, we know $d\hat{u}_y/dy(\pm 1) = 0$, which we substitute for the pressure boundary condition. To solve these coupled equations we use the influence matrix method and tau correction developed by Kleiser and Schumann [105]. Although we set the wall-normal boundary condition in Eq. 4.7 by the slot jets in Eq. 4.2, we note that it is simple to replace this boundary condition with any shape of actuation.

4.2.2 Data-driven framework

The objective in deep RL is to train an agent, commonly a neural network, to approximate the optimal control policy $a = \pi^*(s)$, which given a state observation s, outputs the optimal control action a. The optimal policy seeks to maximize the expected long time discounted cumulative reward,

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{l=0}^{\infty}\gamma^l(r_{t+l\tau})\right], \tag{4.9}$$

where $0 < \gamma < 1$ is the discount factor, τ is the time between control actions, and r_t is the reward, a scalar-valued control objective function decided by the user evaluated at time t. As our objective in this work is to minimize the drag of our turbulent Couette system while simultaneously avoiding the use of superfluous control actions, we define the reward function

as the following,

$$r_t = -\left\langle D(t) + c\|a(t)\|^2 \right\rangle_\tau , \tag{4.10}$$

where c is a scalar and $\langle\cdot\rangle_\tau$ is the average from t to $t+\tau$. We note here that our actuation penalty is proportional to the power required for actuation. In deep RL π^* is learned via repeated cyclic interactions between the agent and the environment i.e. the target system. A typical cycle consists of the following: given a state observation of the system at time t, s_t, the control agent outputs its estimated best control response a_t. This control action is then applied to the environment. The system is allowed to evolve for τ time units, and then the impact of the action is quantified by observing the resulting system state, $s_{t+\tau}$, as well as the reward signal, r_t. This iterate of data, [s_t,a_t,r_t,$s_{t+\tau}$], is then stored and used for updating the control agent for the next time interval.

4.2.3 DManD Modeling Framework

Applications of deep RL often require repeating this cycle $\mathcal{O}(10^{6+})$ times. Because deep RL conventionally requires an online realization of the target system during training, the practicality of training an RL agent for systems that are computationally or experimentally expensive to realize online, e.g. a DNS of turbulent channel flow, is especially bottlenecked by the expense of the environment itself [40].

To circumvent this bottleneck, we employ a method denoted "Data-driven Manifold Dynamics for RL" [221], or "DManD-RL" for short, with some modification. This framework consists of two main learning objectives, which can be broken down into five steps, illustrated in Fig. 4.2. The first objective is to obtain an efficient and accurate low-dimensional surrogate model of the underlying dynamics of the turbulent DNS, which we refer to as the DManD model. This objective is achieved via the first three steps outlined in Fig. 4.2: 1) collect data tuples of the target system experiencing random control actions, 2) obtain a low-dimensional representation of the environment's dynamics, 3) model the dynamics of

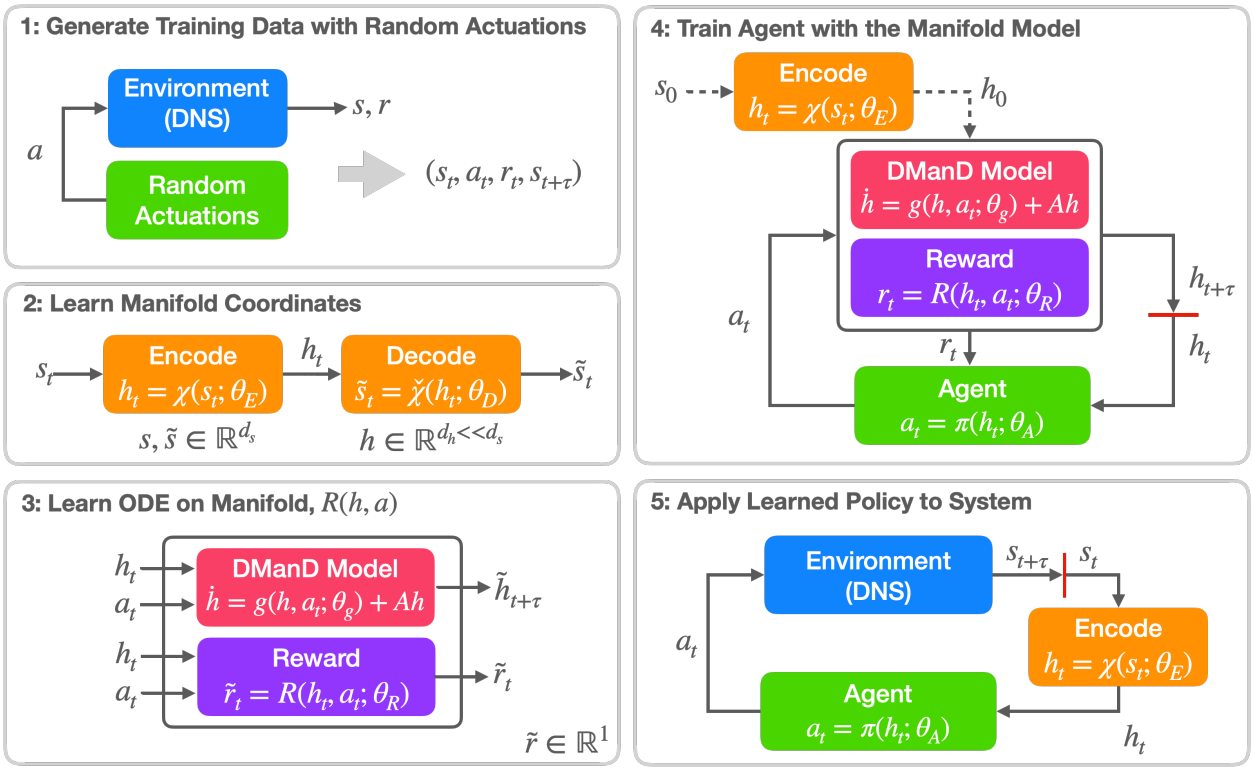

Figure 4.2: Schematic of the DManD-RL framework. After step 3, $\tilde{\cdot}$ is omitted for clarity.

the environment and its response to control inputs. We note here that this first objective is philosophically similar to producing a black-box forward model from data in the field of systems identification [143]. Here we highlight that, unlike methods such as ARMAX, we do not assume any functional form for the dynamics, nor do we assume that the controller is affine. We further distinguish here that we are interested in approximating the RHS of the controlled dynamics without using historical data (i.e. lags or delays).

The second objective is to use this DManD model for RL training to quickly and efficiently obtain an effective control agent. This objective is achieved via the remaining two steps outlined in Fig. 4.2: 4) perform deep RL with the DManD model, and 5) deploy the control agent to the original environment. In the following sections we discuss the details for generating the DManD model in Sec. 4.2.3 and the method for training and deploying the DManD-RL agent in Sec. 4.2.4.

As this framework is completely data-driven, the first step involves collecting sufficient

data to learn an accurate surrogate model. This model must capture the underlying flow system, the response of the dynamics to control inputs, and the impact the control inputs have on the objective. Generating this model requires that we have a large dataset that includes the cycle of data described above ($[s_t,a_t,r_t,s_{t+\tau}]$). In RL training actions are chosen by the policy, however, for training the model we do not necessarily have any policy to generate this data. As such, we instead chose to randomly actuate the flow to generate the original dataset used in training the DManD model. Details on the specifics of the dataset are included in Sec. 4.3.1.

With this data, the second step of the DManD-RL framework involves finding a low-dimensional representation of the state. For many dissipative systems, there is either proof or evidence that the long-time dynamics collapse onto a finite-dimensional invariant manifold[37, 48, 49, 190, 217]. We can define a mapping to coordinates parameterizing this manifold

$$h_t = \chi(s_t), \tag{4.11}$$

where $h_t \in \mathbb{R}^{d_h}$ is the manifold coordinate system and an inverse mapping back to the state

$$s_t = \check{\chi}(h_t). \tag{4.12}$$

When the data lies on a finite-dimensional invariant manifold then the finite-dimensional manifold coordinate representation h_t contains the same information as the state s_t. Thus, if we know χ and $\check{\chi}$ we can simply use h_t in place of s_t for training the RL agent, which requires far fewer degrees of freedom. One subtlety that we gloss over here is that a $d_{\mathcal{M}}$-dimensional manifold may require a set of overlapping local representations called *charts* if one wants to represent the manifold with $d_{\mathcal{M}}$ parameters [47, 50, 110]. However, a manifold with a topological dimension $d_{\mathcal{M}}$ can be embedded in $\mathbb{R}^{2d_{\mathcal{M}}}$ [169, 212]. So, in the worst case, as long as $d_h \geq 2d_{\mathcal{M}}$, a single global coordinate representation can be used, as we do here.

In this work we will approximate χ and $\check{\chi}$ using an undercomplete autoencoder. This

consists of two NNs: an encoder (χ) that reduces the dimension and a decoder ($\check{\chi}$) that expands it. Here we train an autoencoder to find the correction from the linear map given by the proper orthogonal decomposition (POD) [122, 123]. As such we define the state observation s to come from projecting the flow field $\mathbf{u}$ onto a set of POD modes (i.e. s is the POD coefficients). Sec. 4.3.1 includes details on our POD implementation.

For the encoding, we sum the leading d_h POD coefficients with a correction from a NN that is a function of all the POD coefficients:

$$h_t = \chi(s_t; \theta_E) = s_{t,d_h} + \mathcal{E}(s_t; \theta_E), \tag{4.13}$$

where s_{t,d_h} is the first d_h components of s_t and $\mathcal{E}$ is a NN. For the decoding, we want to reconstruct all 500 POD coefficients from the d_h values we have from the encoding. These d_h values are approximately the leading POD coefficients so we can again just add these values to a NN that corrects the leading d_h POD coefficients and reconstructs the remaining POD coefficients:

$$\tilde{s}_t = \check{\chi}(h_t; \theta_D) = [h_t, 0]^T + \mathcal{D}(h_t; \theta_D). \tag{4.14}$$

Here, $[h_t, 0]^T$ represents h_t padded with zeros to the correct size, and $\mathcal{D}$ is a NN. The notation $\tilde{\cdot}$ indicates that this is an approximation of s_t. We refer to this autoencoder structure as a hybrid autoencoder, in contrast to the standard approach of simply treating $\chi(s_t; \theta_E)$ and $\check{\chi}(h_t; \theta_D)$ as NNs. We take this approach because it can achieve lower reconstruction errors [122] than the standard approach and the variables h, and is more interpretable, as a nonlinear correction to POD.

The NNs $\mathcal{E}$ and $\mathcal{D}$ are trained to minimize

$$L = \frac{1}{dK} \sum_{i=1}^{K} ||s_{t_i} - \check{\chi}(\chi(s_{t_i}; \theta_E); \theta_D)||_2^2 + \frac{1}{d_h K} \sum_{i=1}^{K} \xi ||\mathcal{E}(s_{t_i}; \theta_E) + \mathcal{D}_{d_h}(h_{t_i}; \theta_D)||_2^2, \tag{4.15}$$

where $\mathcal{D}_{d_h}$ is the first d_h components of the decoder, ξ is a scalar, and K is the batch size. In

this loss, the first term is the reconstruction mean-squared error (MSE), and the second term promotes the accurate reconstruction of the leading d_h POD coefficients. We include this second term because it must go to zero if the reconstruction is perfect. This is because the modification that the NN makes in the encoder must be removed by the NN in the decoder. In Sec. 4.3.2 we provide details on autoencoder training.

Now that we have a low-dimensional representation of the state observation, in step three, we train a model to predict the evolution of h and the reward r_t. To predict the evolution of h (from h_t to $h_{t+\tau}$) we train a "stabilized" neural ordinary differential equation (ODE) [126]

$$\frac{dh}{dt} = g(h, a; \theta_g) + Ah, \tag{4.16}$$

which can be integrated forward in time to predict

$$\tilde{h}_{t+\tau} = h_t + \int_t^{t+\tau} g(h, a; \theta_g) + Ahdt, \tag{4.17}$$

where g is a NN and A is a matrix that can be learned from data or fixed. We chose to find an ODE instead of a discrete timestepping method because it allows us to evolve this equation to arbitrary times that may not align with the sampling rate of our data. This is a highly desirable trait as it means we can freely vary the time between actions after training the DManD model, which could not be done if we found a discrete time map from h_t to $h_{t+\tau}$.

The linear term in Eq. 4.16 is important for stability. Without this term, small errors in the dynamics can lead to linear growth at long times, which a linear damping term prevents [123, 125, 126]. In this work, we set the linear term

$$A_{ij} = -\beta \delta_{ij} \sigma_i(h), \tag{4.18}$$

where $\beta = 0.1$, δ_{ij} is the Kronecker delta, and $\sigma_i(h)$ is the standard deviation of the ith component of h. This term acts as a damping, preventing trajectories from moving far away

from the training data. In [125] we show in Couette flow without actuation that this term prevents models from becoming unstable. We note that the addition of this damping term does not negatively impact the accuracy of the vector field in the region of state space where there is data, because the damping term is present when we train the NN. This allows the NN to compensate for the damping term in the region where the data lies. We train the NN g to minimize

$$J = \frac{1}{d_h K} \sum_{i=1}^{K} ||h_{t_i+\tau} - \tilde{h}_{t_i+\tau}||_2^2. \tag{4.19}$$

We describe in more detail how the gradient of this loss is computed in [125].

The final piece of the algorithm is computing the reward. We could to this directly by mapping h back to $\mathbf{u}$ and computing the corresponding drag. However, this is undesirable because it is computationally expensive, and, in general, it may not be possible to directly compute the reward from the state. To overcome this difficulty, we use a NN to compute an estimate $\tilde{D}$ of the drag from h:

$$\tilde{D} = \mathcal{R}(h; \theta_R), \tag{4.20}$$

which we train to minimize $J_D = 1/K \sum_{i=1}^{K} ||D_{t_i} - \tilde{D}_{t_i}||_2^2$. Then, we compute the reward $\tilde{r}_t = R(h_t, a_t; \theta_R)$ by inserting $\tilde{D}$ into Eq. 4.10.

4.2.4 Reinforcement Learning using DManD

With our DManD model of the underlying dynamics, we can now proceed to step four and efficiently obtain a control policy by training an RL agent to interact with the low-dimensional DManD model rather than the expensive DNS. Rather than learning $a_t = \pi(s_t; \theta_A)$, our goal shifts to learning

$$a_t = \pi(h_t; \theta_A), \tag{4.21}$$

where θ_A are the network parameters of the RL agent. In this work, we employ the Soft Actor-Critic (SAC) RL algorithm [76] but we emphasize that the DManD-RL framework works with any general RL algorithm. SAC was chosen in this application because it possesses several advantageous characteristics including the ability to output control signals from a continuous range, an off-policy formulation that allows the "reuse" of previously generated data, and twin critic networks to aid the brittleness commonly associated with many off-policy deep RL algorithms. Distinctively, SAC has a stochastic actor with an additional entropy-maximizing formulation. Here, as a slight abuse of notation, we redefine π as the probability distribution of actions, $\sum_a \pi(a|s) = 1$, to match the nomenclature of the SAC framework. This means that during training, actions are not deterministically output, but rather sampled $a_t \sim \pi(\cdot|s)$. In SAC, this is practically achieved by having the agent output the mean, $\mu(s)$, and standard deviation, $\sigma(s)$, of this distribution as a function of the observed state. The output action is sampled with noise $\xi \sim \mathcal{N}(0,1)$: $a_t = \mu(s_t) + \sigma(s_t) \odot \xi$. Post training, a deterministic policy can be recovered by setting σ to zero. This formulation modifies the typical RL objective of Eq. 4.9 to the following

$$\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{l=0}^{\infty} \gamma^l \left(r_{t+l\tau} + \alpha \mathcal{H}(\pi(\cdot|s_t))\right)\right], \tag{4.22}$$

where the entropy of the policy, $\mathcal{H}$, is defined as

$$\mathcal{H}(\pi(\cdot|s_t)) = -\log(\pi(\cdot|s_t)). \tag{4.23}$$

Here α is the trade-off coefficient, which is set to 1.0. This entropy-regularized objective promotes wider state-action exploration and the ability to invest in multiple modes of near-optimal strategies in addition to maximizing the cumulative reward. In this work, the agent is trained stochastically but deployed deterministically during testing. Our implementation of SAC utilizes NNs to approximate the policy function (i.e. agent) π, the two critic functions, Q_1 and Q_2, and the value function V. For more details regarding the derivation and

implementation of SAC, we refer the reader to Haarnoja et al. [76].

With a trained DManD-RL agent, we can proceed to the fifth and final step and deploy the control agent to the original turbulent channel DNS for application. As the RL agent learned its control strategy by observing the low dimensional manifold coordinate system, we must map the high-dimensional state observations made in the DNS to the proper input by using the previously obtained encoder function, χ,

$$a_t = \pi(\chi(s_t)). \tag{4.24}$$

The trained agent can now be deployed in a closed-loop control fashion.

4.3 Results

4.3.1 Description of Data

The first step in DManD-RL is to generate a data set for training models. Our data set consists of 500 different initial conditions, which we evolved forward 300 time units with a random actuation chosen every 5 time units. We sample the random actuations uniformly between -1 and 1. Every 1 time unit we record the velocity field $\mathbf{u}$ and the action a, yielding $1.5 \cdot 10^5$ snapshots of data. We use 80% of this data to train the models, and the remaining 20% to test performance on the data never previously seen by the model. In the training data 2.5% of trajectories laminarize over the 300 time units resulting in $\sim 0.7\%$ of snapshots with a drag below 10. The inclusion of this data allows the DManD model to capture laminarization events.

Due to the high-dimensional nature of the data, learning a manifold coordinate system using the velocity field on the grid as the state is challenging. As such, we first preprocess the data using the proper orthogonal decomposition (POD) to reduce the dimension from $\mathcal{O}(10^5)$ to 500 and treat these 500 POD coefficients as the state observation s. In the POD,

we find modes $\mathbf{\Phi}$, which we project onto to maximize

$$\frac{\left\langle |(\mathbf{u}', \mathbf{\Phi})|^2 \right\rangle}{||\mathbf{\Phi}||^2}, \tag{4.25}$$

which uses the fluctuating velocity $\mathbf{u}' = \mathbf{u} - \langle \mathbf{u} \rangle$ ($\langle \cdot \rangle$ denotes the average). As shown in [83, 181], these modes can be found by solving the following eigenvalue problem:

$$\sum_{j=1}^{3} \int_0^{L_x} \int_{-1}^{1} \int_0^{L_z} \langle u_i'(\mathbf{x}, t) \bar{u}_j(\mathbf{x}', t) \rangle \, \Phi_j^{(n)}(\mathbf{x}') \, d\mathbf{x}' = \lambda_i \Phi_i^{(n)}(\mathbf{x}), \tag{4.26}$$

where $\bar{\cdot}$ is the complex conjugate. Naïve implementation of POD requires solution of a computationally expensive $d \times d$ eigenvalue problem after approximating these integrals. This formulation also fails to respect the streamwise translation invariance of our system.

We account for this translational invariance, and make the problem tractable, by exploiting the fact that in translation invariant directions POD eigenfunctions take the form of Fourier modes [83]. Note that the fixed position of the slot jets breaks translation invariance in z. This turns the eigenvalue problem in Eq. 4.26 into

$$L_x \sum_{j=1}^{3} \int_{-1}^{1} \int_0^{L_z} \langle \hat{u}_i'(k_x, y', z', t) \bar{\hat{u}}_j'(k_x, y', z', t) \rangle \, \varphi_{jk_x}^{(n)}(y', z') \, dy' dz' = \lambda_{k_x}^{(n)} \varphi_{ik_x}^{(n)}(y, z). \tag{4.27}$$

This is a $3N_yN_z \times 3N_yN_z$ eigenvalue problem for every wavenumber k_x. We speed up the computation of this eigenvalue problem by evenly sampling $5,000$ snapshots of training data.

Solving the eigenvalue problem in Eq. 4.27 results in eigenvectors

$$\mathbf{\Phi}_{k_x}^{(n)}(\mathbf{x}) = \frac{1}{\sqrt{L_x}} \exp\left(2\pi i \frac{k_x x}{L_x}\right) \boldsymbol{\varphi}_{k_x}^{(n)}(y, z), \tag{4.28}$$

and eigenvalues $\lambda_{k_x}^{(n)}$. We then project $\mathbf{u}$ onto the leading 305 modes, sorted according to the magnitude of λ, giving us the state observation s. This results in a 500-dimensional state because a majority of the modes are complex. Figure 4.3a shows the sorted eigenvalues. The

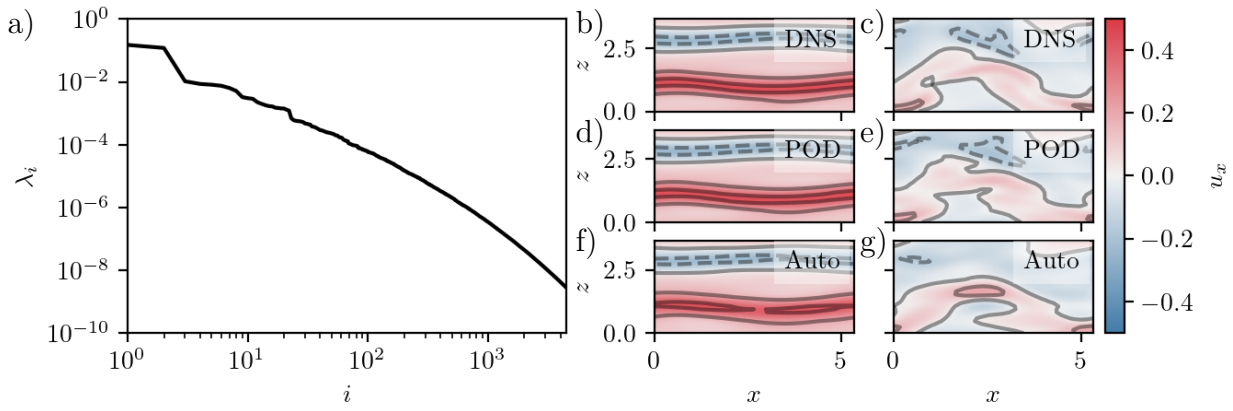

Figure 4.3: (a) Eigenvalues from the POD. (b) and (c) Snapshots of the centerline streamwise velocity ($u_x(y = 0)$) from the DNS at representative low-drag (left) and high-drag (right) instants. (d) and (e) POD reconstruction with 500 modes of the DNS results in (b) and (c). (f) and (g) Autoencoder reconstruction of (d) and (e). Solid contour lines are positive and dotted contour lines are negative.

eigenvalues drop off quickly resulting in the first 305 modes containing 99.8% of the energy.

4.3.2 Manifold Coordinate System

Now that we reduced the dimension of the state observation s, we train an autoencoder as described in Sec. 4.2.3 to find h. In Linot and Graham [125] we varied d_h for the unactuated Couette flow system and found the DManD models to be highly accurate with fewer than 20 degrees of freedom. In the present system, additional degrees of freedom will be excited by the actuations, so we increased d_h slightly, to $d_h = 25$. For training, we first normalized s by subtracting the mean and dividing by the maximum standard deviation. We then trained four autoencoders until the training error stopped improving. We selected the autoencoder based on which yielded the best forecasting performance when coupled to a neural ODE described in the following section. The selected autoencoder had a test MSE of $1.45 \cdot 10^{-4}$. The architecture and parameters of the autoencoders are reported in Table 4.1.

In Figs. 4.3b-4.3g we compare the centerplane streamwise velocity of two flowfields – one in a bursting state (high-drag) and the other in a hibernating (low-drag) state – to

Table 4.1: Architectures of NNs. "Shape" indicates the dimension of each layer, "Activation" the corresponding activation functions, and "sig" is the sigmoid activation."Learning Rate" gives the learning rate for the Adam optimizer [102]. When multiple learning rates are noted, the value was changed from one value to the next at even intervals during training.

Function	Shape	Activation	Learning Rate
$\mathcal{E}$	$500/1000/d_h$	sig/lin	$[10^{-3}, 10^{-4}]$
$\mathcal{D}$	$d_h/1000/500$	sig/lin	$[10^{-3}, 10^{-4}]$
g	$d_h/200/200/200/200/d_h$	sig/sig/sig/sig/lin	$[10^{-2}, 10^{-3}, 10^{-4}]$
$\mathcal{R}$	$d_h/100/100/1$	sig/sig/lin	$[10^{-3}]$
$\mathbb{O}$	$1024/100/100/d_h$	sig/sig/lin	$[10^{-3}]$
π	$d_h/256/128/128/2/1$	ReLU/ReLU/ReLU/lin/Tanh	$[3 \cdot 10^{-4}]$
Q_1	$d_h/256/128/128/1$	ReLU/ReLU/ReLU/lin	$[3 \cdot 10^{-3}]$
Q_2	$d_h/256/128/128/1$	ReLU/ReLU/ReLU/lin	$[3 \cdot 10^{-3}]$
V	$d_h/256/128/128/1$	ReLU/ReLU/ReLU/lin	$[3 \cdot 10^{-3}]$

their reconstruction with 305 POD modes and their reconstruction from the autoencoder with $d_h = 25$. When reconstructing the simpler hibernating state the reconstruction from POD and the autoencoder match almost exactly. In the case of the bursting snapshot, both the POD and the autoencoder still match the full high-dimensional state well, accurately capturing the magnitude and location of the streamwise velocity. In the case of POD, some error is introduced as it distorts the details in the DNS, and then the autoencoder further smooths some of these details. However, considering that we reduced the dimension of the problem from $\mathcal{O}(10^5)$ to 25, the reconstruction is excellent. It is important to note that extending these results to different Reynolds numbers requires training a new model at every Reynolds number. Extending to higher Reynolds numbers, and/or stronger actuations, may require two modifications: 1) The number of latent variables must increase to account for the higher-dimensional manifold on which the data lies and 2) the amount of data must increase to sample this higher-dimensional space. Unfortunately, it remains unclear how the manifold dimension of a minimal flow unit scales with Reynolds number. Now that we can represent the state s in the manifold coordinate system as h, the next step is learning a time evolution model to control.

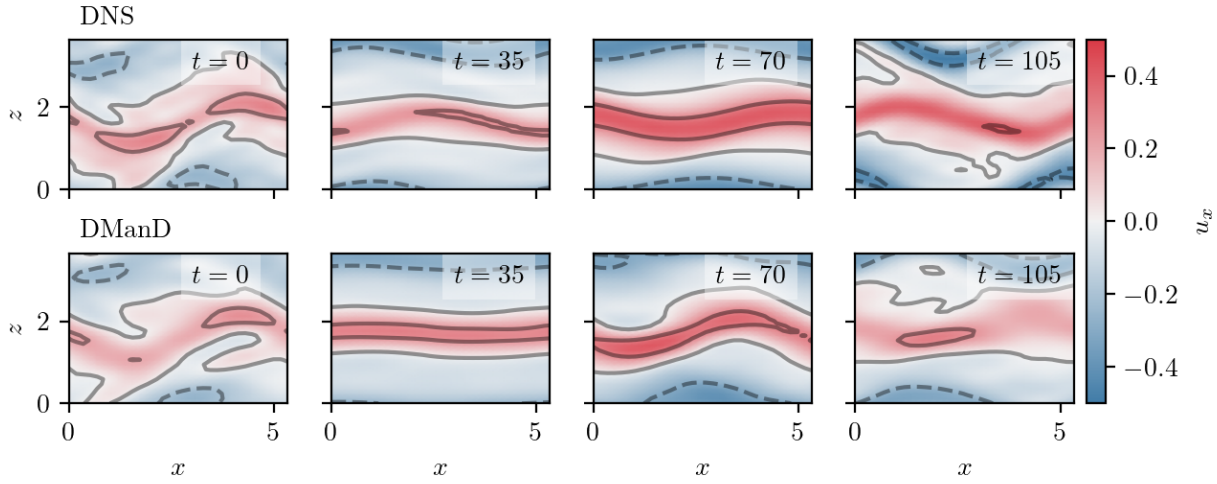

Figure 4.4: Comparison of snapshots from a randomly actuated trajectory from the DNS (top) with the DManD reconstruction (bottom). Times are given at the top right of each image.

4.3.3 DManD Performance

We now describe the neural ODE training and performance of the resulting model. Before training, we normalize h by subtracting the mean and dividing each component by its standard deviation. In Table 4.1 we show the architecture for the neural ODEs used in this section. We trained four neural ODE models for each of the four autoencoders. Then we selected the autoencoder and neural ODE pair that best reconstructed the statistics we report below. Again, we trained the neural ODEs until we no longer saw an improvement in performance.

The first statistics we investigate validate the ability of the model to track the true dynamics over short times. In Fig. 4.4 we show an example of a randomly actuated trajectory from the DNS and reconstructed with the DManD model. In the first two snapshots, the trajectories are in quantitative agreement, after which the trajectories still appear qualitatively quite similar. Both the DNS and the DManD model exhibit the streak breakdown and regeneration cycle over this series of snapshots. At $t = 0$ rolls are forming, at $t = 35$ these rolls lift low-speed fluid off the wall forming streaks, at $t = 70$ the streaks become wavy, and, finally, at $t = 105$ the streaks have broken down.

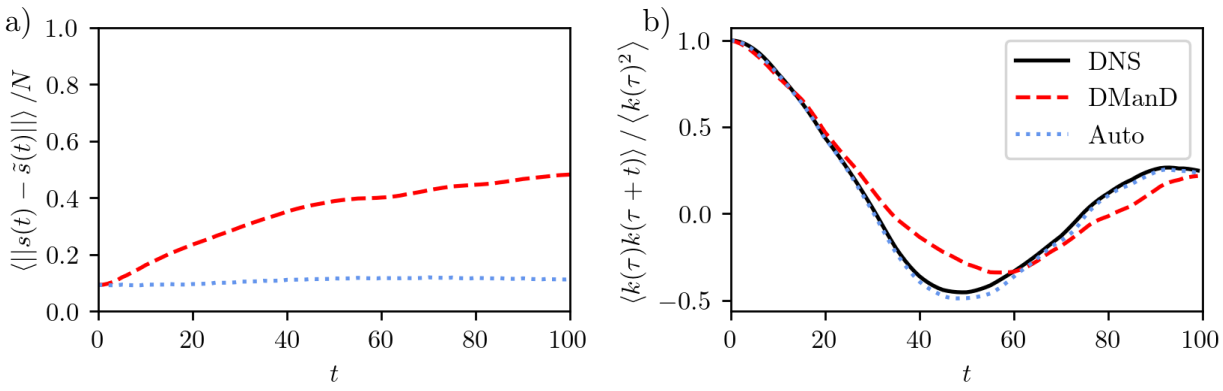

Figure 4.5: (a) Ensemble averaged tracking error and (b) temporal autocorrelation of kinetic energy for the DNS, the DManD model, and the autoencoder. “Auto" represents putting the DNS trajectories through the autoencoder without any time prediction: i.e. this represents the error of just reducing, then expanding the dimension.

Next, we consider model performance when averaged over many trajectories. In all of these cases we show both the error from time evolving with the DManD model, and the error incurred due to the autoencoder. We compute the error due to the autoencoder by inputting the true DNS solution through the autoencoder and computing the relevant statistic. Figure 4.5a is a plot of the normalized ensemble-averaged tracking error as a function of time. We compute this error by finding the difference between $s(t)$ and $\tilde{s}(t)$ from the DManD model for 100 initial conditions. Then, we normalize this error by computing the difference between two states on the attractor at random times t_i and t_j $N = \left\langle ||s_{t_i} - s_{t_j}|| \right\rangle$. With this normalization, the long-time dynamics of two slightly perturbed initial conditions with different random actuation sequences should approach unity. The error at $t = 0$ represents the discrepancy between the full state and its reconstruction when passed through the autoencoder. The ensembled-averaged tracking error rises steadily at one slope for the first 50 time units and then at a lower slope after that. Once the curve levels off, the true and model trajectories have become uncorrelated. Based on these results, the model tracks well for ~ 50 time units. For reference, the Lyapunov time (inverse of the Lyapunov exponent) for the unactuated system is $\tau_L = 48$ time units [90].

In addition to the tracking error, we also check the ability of the DManD model to capture

the temporal autocorrelation of the kinetic energy. To compute this autocorrelation we take the instantaneous kinetic energy of the flow

$$E(t) = \frac{1}{2L_x L_z} \int_0^{L_z} \int_{-1}^{1} \int_0^{L_x} \frac{1}{2} \mathbf{u} \cdot \mathbf{u} \, d\boldsymbol{x}, \tag{4.29}$$

and subtract the mean to yield $k(t) = E(t) - \langle E \rangle$. Figure 4.5b shows this temporal autocorrelation computed from the DNS, the DManD model, and the autoencoder. We see the temporal autocorrelation of the model matches the true temporal autocorrelation closely over the first ~ 30 time units.

Due to the chaotic nature of this system, short-time tracking is limited by the Lyapunov time. Nevertheless, a good model should still be able to capture the long-time statistics of the original system. In Fig. 4.6 we show the four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder. We computed these statistics by averaging over the entire testing dataset (100 trajectories 300 time units in length). For all these quantities, the DManD model is in excellent agreement with the DNS. In the cases of $\langle u_y^2 \rangle$ and $\langle u_z^2 \rangle$, surprisingly, the autoencoder appears to perform worse than the DManD model at matching the DNS. With perfect prediction of the DManD model, it would exactly match the autoencoder. This slight disagreement indicates that some of the states in the manifold coordinates (h) are driven somewhat outside the expected range of values.

All of the statistics shown so far indicate that the DManD model accurately captures the dynamics of the randomly actuated DNS with only $d_h = 25$ degrees of freedom. The last step before using this model in the RL framework is training the reward network as described in Sec. 4.2.3. We trained a NN with the architecture in Table 4.1. In Fig. 4.7a we show a PDF of the parity plot between the true and predicted drag on test data, and report the MSE on the normalized data. The excellent agreement indicates we can compute accurate values of the reward directly from h.

Finally, although we computed the manifold coordinate system directly from the state,

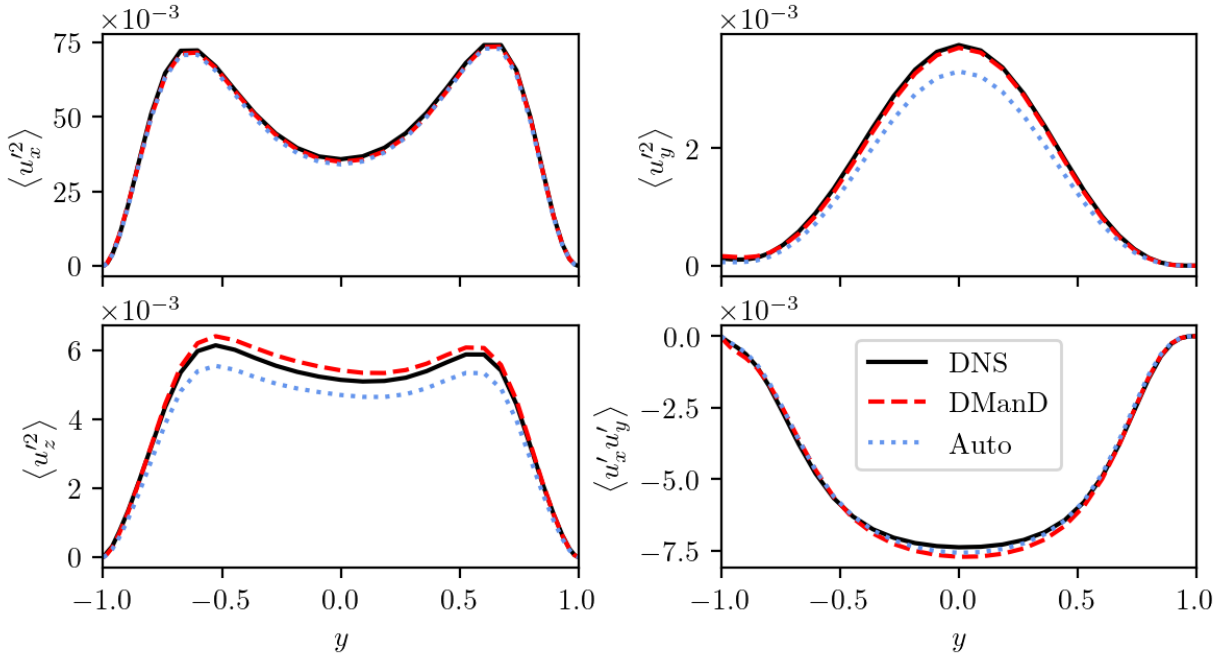

Figure 4.6: Four components of the Reynolds stress for the DNS, the DManD model, and the autoencoder.

a natural extension of this work is to use a more limited set of observations. As a first step in this direction, we trained a NN $\mathbb{O}(\cdot)$ to map 32×32 wall shear rate observations at the bottom wall to the manifold coordinate system:

$$\tilde{h} = \mathbb{O}(\partial_y u_x|_{y=-1}; \theta_O). \tag{4.30}$$

The details of this NN are included in Table 4.1. This mapping allows us to test if it is possible to directly use wall observables that are experimentally realizable with our control policy. In Fig. 4.7b we show the parity plot of reconstructing h from wall observables and report the MSE for normalized data. While the parity plot is not as sharply peaked as we might like, note that it is shown on a logarithmic scale.

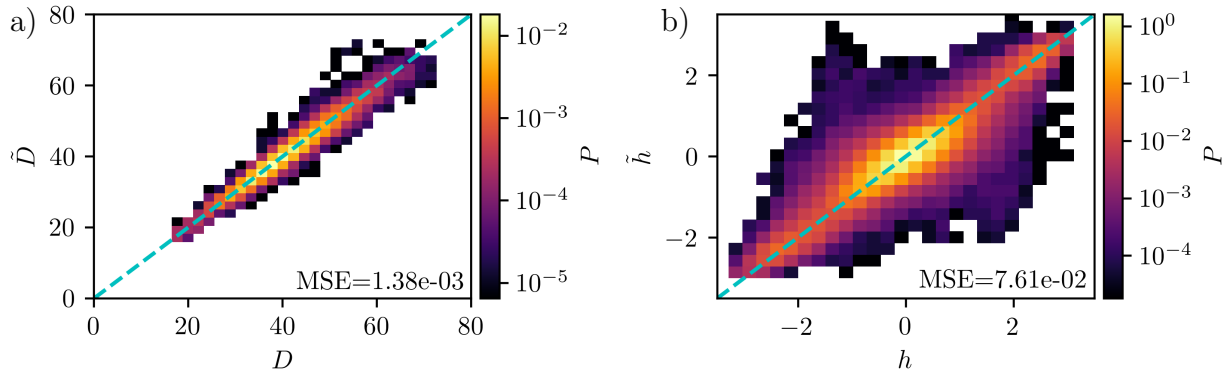

Figure 4.7: (a) Joint PDF of the true (D) and predicted drag ($\tilde{D}$) from the reward network, Eq. 4.20. (b) Joint PDF of the true (h) and predicted ($\tilde{h}$) low-dimensional state from the observation network, Eq. 4.30. Note the logarithmic scales. The cyan line indicates perfect reconstruction, and the MSE is for D and h with the mean subtracted and divided by the standard deviation.

4.3.4 DManD-RL Performance

Now with an efficient and low-dimensional model of the underlying dynamics of the turbulent flow in hand, we can quickly obtain a control agent by performing deep RL on the DManD model rather than the original costly DNS. In this work, we employ the Soft Actor-Critic (SAC) RL algorithm [76], which requires training a policy function (i.e. agent) π, two critic functions, Q_1 and Q_2, and a value function V. The RL networks are trained for 10,000 episodes, with each episode consisting of DManD model trajectories of 300 time units. The initial condition for each episode is selected at random from our training data that consists of $1.2 \cdot 10^5$ time units and snapshots of data. The selection of 10,000 initial states from this dataset ensures the RL has a wide variety of trajectories to learn a general policy. Here we choose the action time to be $\tau = 5.0$ and $\gamma = 0.99$. Accordingly, the discount factor over one Lyapunov time $\tau_L \approx 48$ of the unactuated system is $\gamma^{\tau_L/\tau} \approx 0.9$.

Once trained, we apply the DManD-RL agent to the original turbulent DNS, with which it has never directly seen or interacted. To deploy the agent for control, we insert the already-trained encoder, χ, between the agent and the environment to map state observations of the turbulent DNS to the manifold representation, h, as this is the observation space where the

DManD-RL agent was trained.

Shown in Fig. 4.8 are times series, of length 900, of the drag in various cases from DNS trials using 25 different unseen initial conditions. Here we define the % drag reduction,

$$DR = \frac{D_0 - D}{D_0} \cdot 100\%, \tag{4.31}$$

where D_0, and D is the time averaged drag experienced for the 25 test trajectories over 900 time units under no control and control, respectively. In Fig. 4.8a we plot the drag of the 25 test trajectories in the absence of control and highlight that the system predominately remains turbulent, with a few laminarizations. In Fig. 4.8b we show that utilizing a random jet actuation policy does not reduce drag and even results in increased drag $DR = -4\%$ across the 25 trajectories over the 900 time unit window. In Fig. 4.8c we show that our DManD-RL agent is able to significantly reduce the drag of the turbulent DNS and laminarize 21/25 turbulent initial conditions, yielding $DR = 48\%$. In comparison, we show in Fig. 4.8d that the conventionally trained RL agent, which was trained directly on the turbulent DNS, results laminarization of 23/25 test trajectories with $DR = 56\%$, similar to that of the DManD-RL method. We highlight here, however, that the DManD-RL control agent was obtained at a small fraction of the computational cost compared to its conventional counterpart. For reference, the DManD-RL agent required $\sim$ 3.7s per training episode, while a conventional application of deep RL required $\sim$ 1630s per episode, on a 2.40GHz Intel Xeon CPU E5-2640 v4. This corresponds to a 440 times speedup in training time.

Finally, we now limit the state observation of the DManD-RL agent to information observable at the wall i.e. wall shear rate, by pairing the DManD-RL agent trained for Fig. 4.8c with the observation network ($\mathbb{O}(\partial_y u_x|_{y=-1}; \theta_O)$). We demonstrate in Fig. 4.8e that the agent with only access to wall observations, $a_t = \pi(\mathbb{O}(\partial_y u_x|_{y=-1}; \theta_O); \theta_A)$, performs just as well as its counterparts, with 20/25 test trajectories laminarizing and $DR = 50\%$ over the 900 time unit window. Here we note that DR is influenced by how many and how quickly

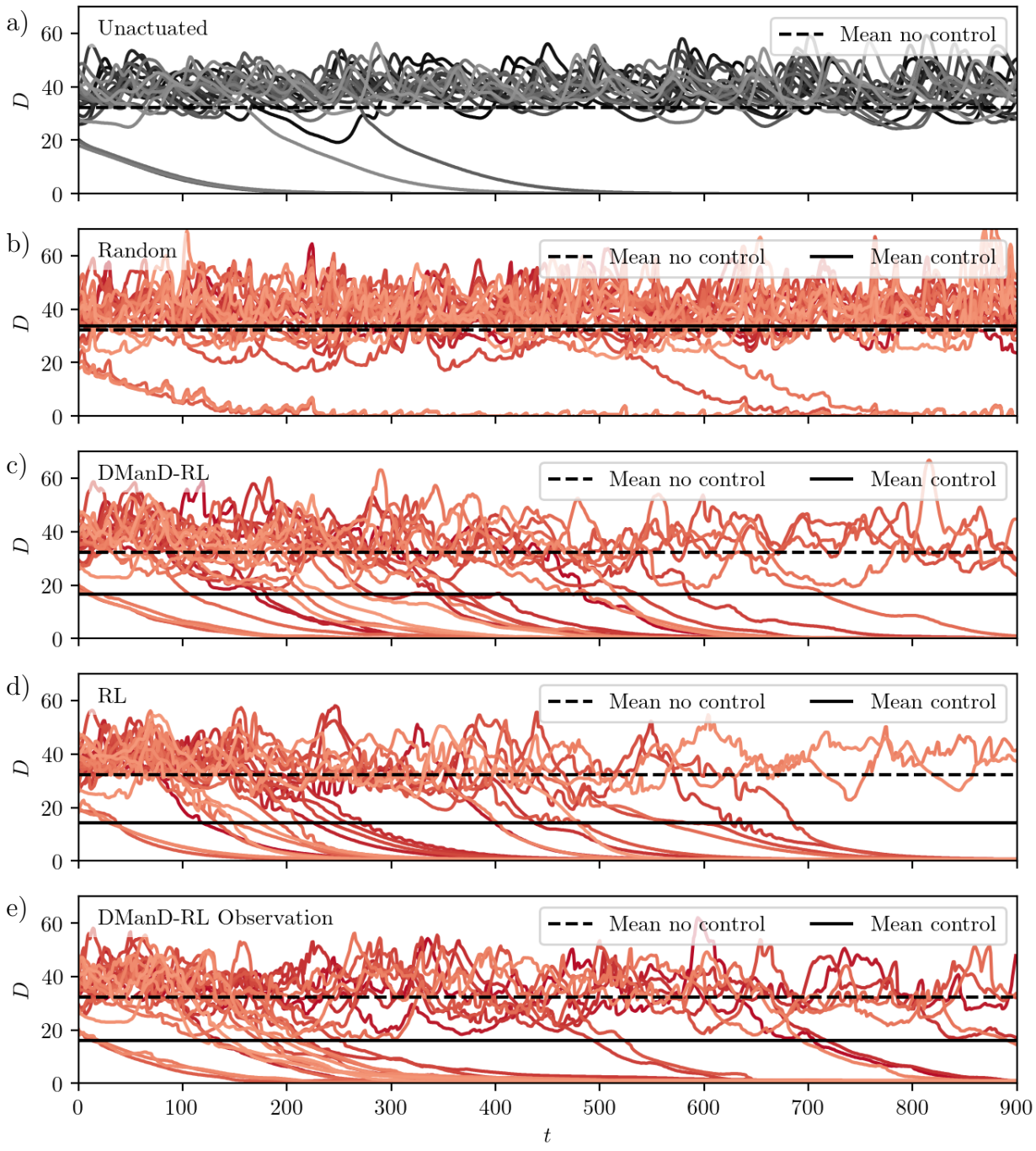

Figure 4.8: Trajectories beginning from test initial conditions with (a) no control, (b) random actuations, (c) DManD-RL control, (d) DNS-based RL control, and (e) DManD-RL control using wall observations. Each figure shows 25 test trajectories.

test trajectories laminarize, as well as how turbulent transient trajectories are attenuated by the various controllers and we emphasize that all of the RL controllers perform similarly. Furthermore, there is nothing particularly notable about the trajectories that failed to laminarize over this time window, and, given sufficient time, the controller would likely laminarize these trajectories too.

Interpretation of the mechanism of drag reduction

Given the observed effectiveness of the control policy discovered by the RL algorithm, it is desirable to understand how the controller is modifying the flow. In this section, we describe two sets of observations that may shed some light on this issue. The first focuses on the control action in the time preceding a laminarization event. Shown in Fig. 4.9a is a time series depicting the drag and left actuator control signal of a DNS controlled by the DManD-RL agent. Fig. 4.9b shows the wall actuation and an isosurface of streamwise velocity illustrating the streamwise streak structure at $t = 149$, indicated by the first red dot on the drag time series in fig:Mechanism1-a. Here the left slot jet is sucking, and the right blowing, drawing the fluid (and low-speed streak indicated by the isosurface) to the left. At $t = 179$ (second red dot and Fig. 4.9c), the streak is now located over the left jet. Once the streak is above the left jet, the agent executes a series of actions that destabilize the low-speed streak, causing it to break down, shown in Fig. 4.9d. In the wake of the collapse, the agent initiates a strong actuation that leads to the formation of *two* low-speed streaks, shown in Fig. 4.9e. This double low-speed streak structure then proceeds to decay to the laminar state, with the agent applying weak, attenuating control actions to expedite the process.

A DNS initialized with this double-streak structure flow field in the absence of control results in the natural laminarization of the flow. In wall-bounded turbulence, streaks take on a characteristic spacing of 100 wall units [180], which is approximately the width of the MFU cell [77, 96]. The two-streak state has a spanwise length scale that is too small and thus too

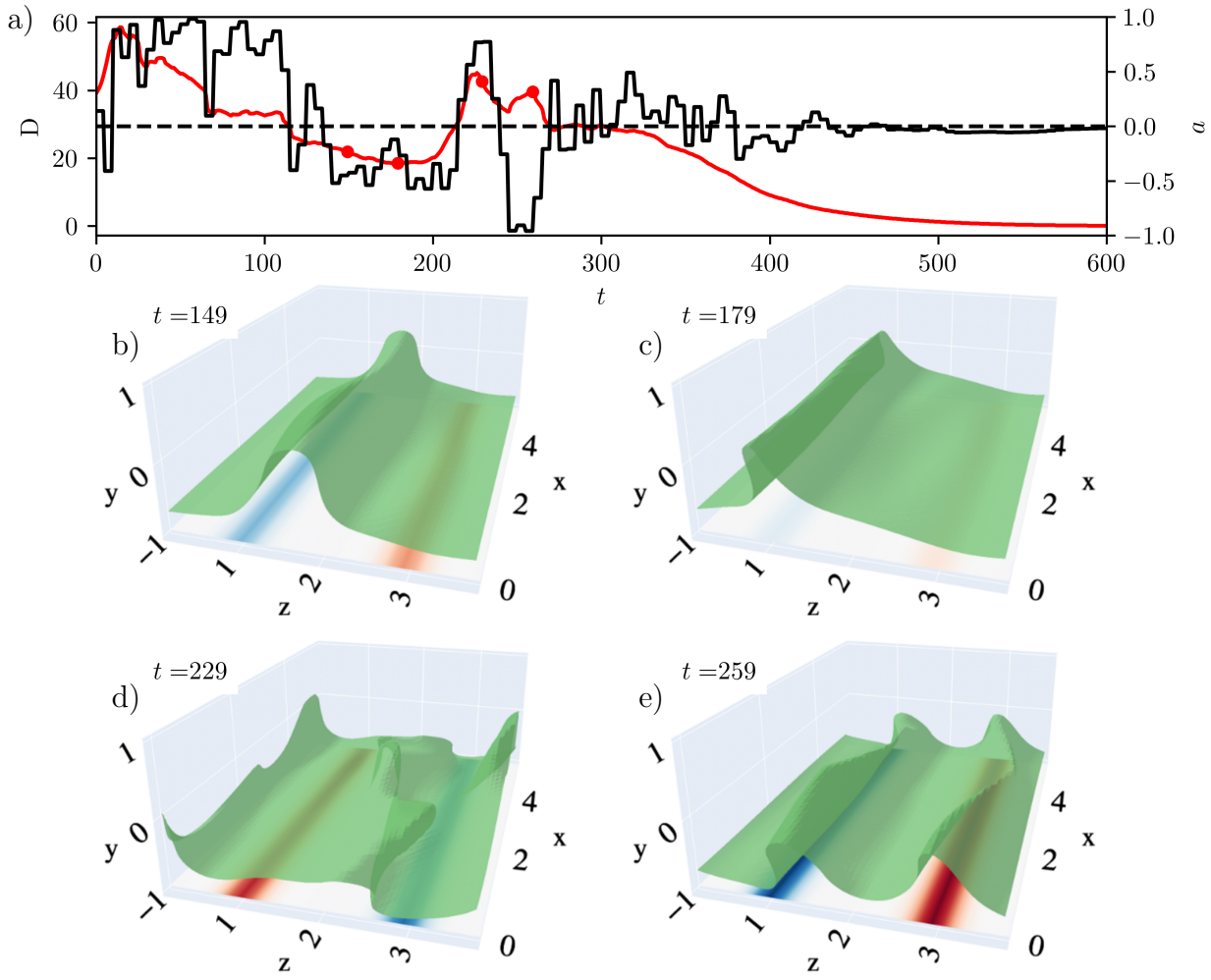

Figure 4.9: (a) Time series of drag (red) and actuation signal of the left jet (black) for an example DNS trajectory controlled by the DManD-RL agent. (b)-(e) snapshots of the trajectory at times marked in (a). These snapshots include an isosurface of $u_x = -0.35$ and the jet actutations (red is fluid injection, blue is fluid suction).

dissipative to self-sustain, leading to breakdown in the SSP and laminarization of the flow. In approximately half of the trials that laminarized, this double-streak structure appeared before laminarization. We comment the reason why every trajectory does not laminarize this way is likely due to the RL agent learning multiple approximately optimal strategies–which is one of the advantages of the SAC framework. While advantageous for performance, The underlying mechanistic strategy behind the other half of the laminarizations is not currently clear.

This is a very interesting and counterintuitive strategy, particularly because it is intrinsically *nonlinear*. At any given instant, the two-jet system can only drive a wall-normal flow with the same fundamental wavelength as the domain. But here the RL agent implements a time-dependent policy that ends up generating a flow structure with, roughly speaking, half the wavelength of the domain. I.e. it has figured out a way to drive structure to a smaller scale, where viscosity can take over and drive the flow to laminar.

The second set of observations we highlight addresses the relationship between the RL control policy and a flow control strategy, widely-studied with simulations, that we mentioned in Sec. 4.1 – opposition control. In opposition control, the entire wall-normal velocity field at both walls is set to have the opposite sign as the wall-normal velocity at an $x - z$ "detection plane" located at some y near the wall. We highlight that this method possesses much greater control authority than in the present work, as it has full spatial control at both walls whereas here we only actuate two spatially localized slot jets on a single wall. Furthermore, our controller holds a constant actuation for multiple time units, whereas opposition control updates on the time scale of a time step.

With these differences in mind, we investigate trajectories controlled by the DManD-RL policy in the context of potential similarities to opposition control. To make this comparison, we first must characterize the wall-normal velocity at the detection plane, but because the jets span the length of the channel this velocity varies. We take a characteristic detection-plane velocity to be the wall-normal velocity averaged over a jet weighted by the shape of

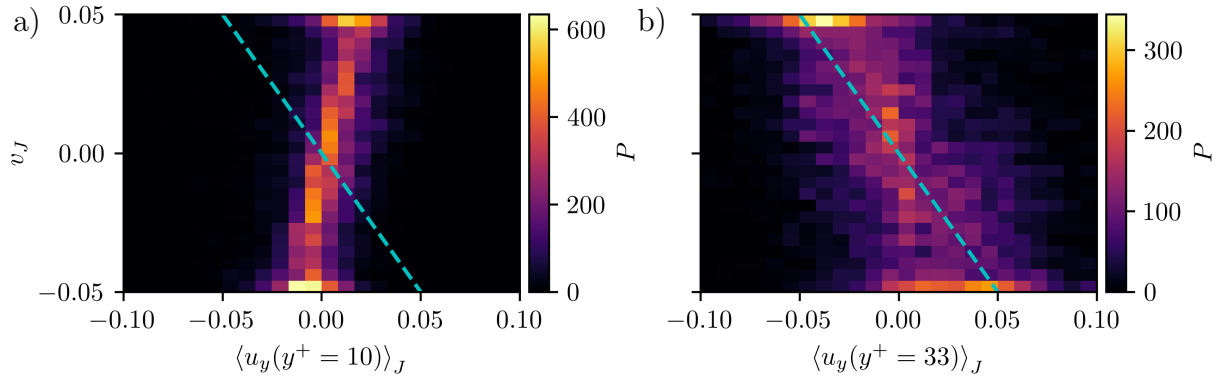

Figure 4.10: Joint PDF of max jet velocity and average wall-normal velocity at (a) $y^+ \approx 10$ and (b) $y^+ = 33$ (the cyan dotted line is $v_J = -\langle u_y \rangle_J$).

the jet, denoted $\langle u_y \rangle_J$.

In Fig. 4.10 we plot a PDF of the maximum jet velocity v_J and the wall-normal velocity $\langle u_y \rangle_J$ (for both jets separately) at a detection plane of $y^+ \approx 10$ (Fig. 4.10a) and $y^+ = 33$ (Fig. 4.10b). We remove laminarization events from this dataset by omitting states with a drag less than 10 because we are interested in the control behavior while the flow is still turbulent. In these plots, we denote opposition control with a unit gain ($v_J = -\langle u_y \rangle_J$) with the cyan line. If the agent performs opposition control, the joint PDF should show a tight distribution with a negative slope. At the detection plane $y^+ \approx 10$ that has been reported to be optimal for opposition control, [22] the RL agent behaves like an anti-opposition controller because there is a positive slope in the PDF. However, at a higher detection plane of $y^+ = 33$, the control agent has a high probability of actuating with sign opposite to $\langle u_y \rangle_J$, as in opposition control. We highlight that the PDF is quite broad, which indicates although the control is opposition-like, the control response is much more complex and diverse than that of opposition control.

Furthermore, we can directly compare our results to Ibrahim et al. [89], who applied standard opposition control to this same Couette domain. In their work, the authors selected a detection plane of $y^+ = 10$ and varied the wall-normal velocity scale $\phi = [0.1, 0.2, 0.5, 1]$. For these parameters, the authors found that the probability of the flow remaining turbulent

after 900 time units of control, the length of our test trajectories shown in Fig. 4.8, to be approximately $[0.77, 0.72, 0.69, 0.42]$. For comparison, with 21/25 trajectories laminarizing, our DManD-RL agent's probability of turbulence is 0.16. This improvement is extremely promising for the future use of RL in controlling turbulent flows, especially as the DManD-RL agent had major and realistic restrictions placed on its control authority when compared to opposition control. We also note that this improvement in control further supports the claim that the agent learned a much more complex and diverse control strategy than opposition control.

In addition to these results we can also compare the dynamics before laminarization between our results and Ibrahim et al. [89]. Notably, Ibrahim et al. [89] found that the phase space portraits of opposition control show much lower power input (proportional to drag) at $\phi = 1$ compared to the unactuated system. Distinctively, opposition control does not appear to exhibit any high-drag bursting events resulting in a 45% reduction in the r.m.s. fluctuation of the power input. In contrast, we highlight that the DManD-RL agent frequently exhibits high-drag behavior prior to laminarization. For example, in Fig. 4.8c, we observe many high-drag "spikes" preceding laminarization. These spikes are of equal or greater drag relative to the uncontrolled flow, shown in Fig. 4.8a. This highlights a major distinction between opposition control and the strategy learned by the DManD-RL agent, which is the ability to captilize on pathways that momentarily increase drag, but lead to laminarization at later times. Opposition control does not exhibit these high-drag excursions on its path to laminarization.

4.4 Summary

In this chapter, we efficiently obtained a control strategy from a limited data set to reduce the drag in a turbulent Couette flow DNS via the control of two streamwise slot jets using the DManD-RL framework. Using a combination of POD and autoencoders, we extracted

a low-dimensional manifold representation of the data, whose dynamics we modeled using a neural ODE. We show that our 25-dimensional DManD model qualitatively captures the turbulent self-sustaining process and has good short-time predictive capabilities, matching the kinetic energy temporal autocorrelation for 30 time units. Furthermore, our DManD model excellently captures long-time statistics such as Reynolds stress. In order for this DManD model to be viable for RL, we added an additional reward network to predict the system drag, D, given the low-dimensional manifold state, h.

We then obtained a control strategy from our DManD model using deep RL, which successfully transferred to and controlled the original DNS. We were able to expeditiously train an RL agent using DManD-RL 440 times faster than a direct application of deep RL to the DNS.

We additionally emphasize here that the data generation and DManD model training in this framework is a fixed one-time cost that is greatly exceeded by the cost of conventional DNS-based RL training. For reference, we found the DNS-based RL training required over a month of training to accomplish 1,000 training episodes, while all steps of the DManD-RL framework (data generation, dynamics model training, 10,000 episodes of RL training) were accomplished within two days.

We find that our DManD-RL agent can consistently drive unseen turbulent initial conditions in the original DNS to the laminar state, despite never having any direct observations or interactions with the DNS. We also demonstrate that there exists a mapping between wall observables and the manifold state, which allowed us to apply the DManD-RL agent with equal effectiveness using only wall shear rate observations.

When investigating the mechanistic nature of the learned control strategy, we observed multiple control strategies executed by the DManD-RL agent. One novel strategy the agent appears to employ consists of manipulating the low-speed streak to a preferred location, causing the breakdown of the streak, and in the wake of the break-down forming two low-speed streaks in its place. These two low-speed streaks are unsustainable within the domain,

breaking the SSP and resulting in laminarization.

When comparing the ensemble behavior of our control agent to opposition control, we find that the controller behaves anti-oppositionally at the commonly-used detection plane location of $y^+ = 10$. At a detection plane of $y^+ = 33$, we find that the ensemble behavior is opposition-like, however, the broadness of the control action distribution, as well as the observation of the two-streak structure, leads us to conclude that the learned controller behavior is much more complex and diverse than a simple opposition feedback rule. We also compare our DManD-RL agent's control performance to that of opposition control and we find that our control agent out-performs opposition control by a notable margin (16% vs. 42% probability of remaining turbulent after 900 time units of control) despite our control set-up and agent possessing much greater restrictions on it spatial and temporal control authority (fixed time intervals of control and localized jets on only the lower wall) compared to opposition control.

Appendix

In this section we present a method for generating a set of actions between -1 and 1 with equal probability that satisfy a zero-net-flux constraint. In order to satisfy this constraint for an arbitrary number of actions a_i from $i = 1, ..., N$, we set $a_N = -\sum_{i=1}^{N-1} a_i$ with the constraint that a_i from $i = 1, ..., N-1$ is drawn from a region that guarantees $a_N \in [-1, 1]$. The region that guarantees this constraint is the set of all points that lie inside and on an $(N-1)$-dimensional cross-polytope [31], which is equivalent to the L_1 ball ($\sum_{i=1}^{N-1} |a_i| \leq 1$) [30]. Thus, if we can uniformly sample points within the cross-polytope we can guarantee that all actions lie between -1 and 1 and we do not introduce bias towards specific actions.

A useful way of viewing this problem is to consider what transformation takes us from uniformly sampling a hypercube of untransformed actions defined by $a'_i \in [-1, 1]$ for $i = 1, ..., N-1$ to the cross-polytope defined by vertices $\{\pm e_i\}$, where e_i is a vector with 1 at

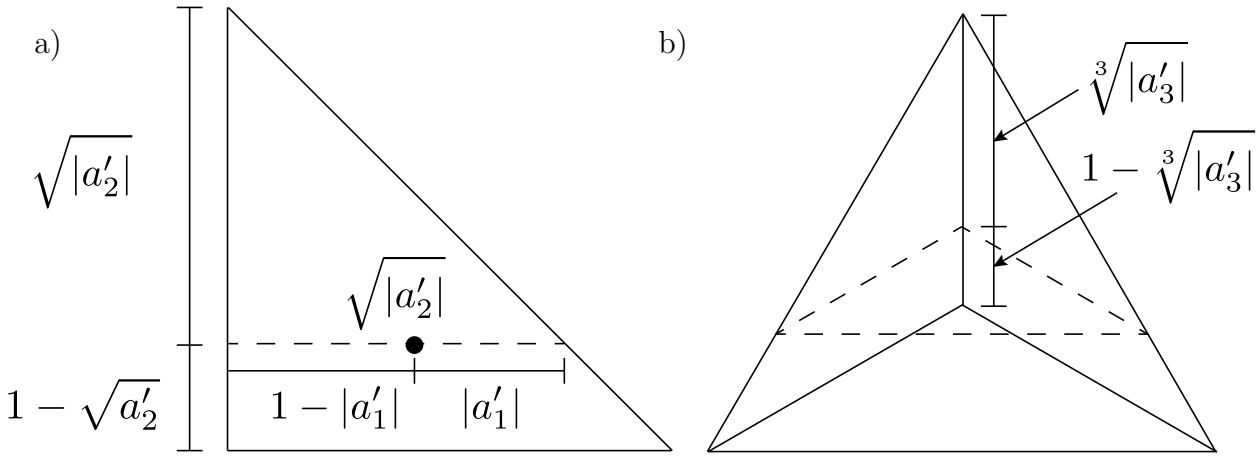

Figure 4.11: The transformations required to uniformly sample a section of a cross-polytope in (a) two dimensions and (b) three dimensions. In three dimensions, you first perform the transformation shown in (b) and then in (a).

the ith location and 0 elsewhere. For 3 actions this corresponds to mapping a'_i uniformly sampled in a square with vertices $\{[1,1],[1,-1],[-1,1],[-1,-1]\}$ to a_i uniformly sampled in a square (diamond) with vertices $\{[1,0],[-1,0],[0,1],[0,-1]\}$. For 4 actions this corresponds to mapping a'_i uniformly sampled in a cube to a_i uniformly sampled in an octahedron. The two-dimensional case is symmetric across four quadrants and the three-dimensional case is symmetric across eight octets, so, without a loss of generality, we discuss how to sample a right isosceles triangle and a trirectangular tetrahedron shown in Fig. 4.11a and 4.11b, respectively. This is equivalent to only considering the absolute values of a'_i

In Fig. 4.11a we present the idea behind uniformly sampling points in the right isosceles triangle. We uniformly sample the right isosceles triangle by first sampling the y-axis with a higher probability of points lying on the bottom than on the top (i.e. $|a_2| = 1 - \sqrt{|a'_2|}$). Then we sample uniformly along the line segment parallel to the base of the triangle with length $\sqrt{|a'_2|}$, such that $a_1 = (1 - |a'_1|)\sqrt{|a'_2|}$. This is a specific case of uniformly sampling points in a triangle which is presented in Glassner [60]. Fig. 4.11b shows how to extend this idea to the trirectangular tetrahedron. In this three-dimensional case, we must sample the base with a higher probability such that $|a_3| = 1 - \sqrt[3]{|a'_3|}$. Instead of a line segment, this results in a right isosceles triangle with two edges of length $\sqrt[3]{|a'_3|}$. Computing the remaining

actions then simply requires performing the same steps as in the two-dimensional case (Fig. 4.11a).

Based on this principle of sampling each dimension one at a time with the appropriate weighting, we can write a general $(N-1)$-dimensional mapping from a'_i to a_i, where we simply multiply by the sign of a'_i to allow all values of the cross-polytope. To perform these operations for $N \geq 3$, first choose $N-1$ untransformed actions between $a'_i \in [-1, 1]$ (e.g. actions output from an agent with tanh activations), then transform these actions such that:

$$a_i = \text{sgn}(a'_i)(1 - |a'_i|^{1/i})\Pi_{j=i+1}^{N-1}|a'_j|^{1/j}, \quad i = 1, ..., N-2 \tag{4.32}$$

$$a_{N-1} = \text{sgn}(a'_{N-1})\left(1 - |a'_{N-1}|^{1/(N-1)}\right) \tag{4.33}$$

$$a_N = -\sum_{i=1}^{N-1} a_i \tag{4.34}$$

This mapping results in a set of actions that satisfy a zero-net-flux constraint and all actions are sampled with equal probability when sampling untransformed actions uniformly.

5

Isolating the manifold with implicit weight-decay autoencoders [1]

While many phenomena in physics and engineering are formally high-dimensional, their long-time dynamics often live on a lower-dimensional manifold. The present work introduces an autoencoder framework that combines implicit regularization with internal linear layers and L_2 regularization (weight decay) to automatically estimate the underlying dimensionality of a data set, produce an orthogonal manifold coordinate system, and provide the mapping functions between the ambient space and manifold space, allowing for out-of-sample projections. We validate our framework's ability to estimate the manifold dimension for a series of datasets from dynamical systems of varying complexities and compare to other state-of-the-art estimators. We analyze the training dynamics of the network to glean insight into the mechanism of low-rank learning and find that collectively each of the implicit regularizing layers compound the low-rank representation and even self-correct during training. Analysis of gradient descent dynamics for this architecture in the linear case reveals the role of the internal linear layers in leading to faster decay of a "collective weight variable" incorporating all layers, and the role of weight decay in breaking degeneracies and thus driving convergence

[1] The text of this chapter is adapted from the prepublication by K. Zeng and M. D. Graham on arXiv, submitted to Machine Learning: Science and Technology, 2023

along directions in which no decay would occur in its absence. We show that this framework can be naturally extended for applications of state-space modeling and forecasting by generating a data-driven dynamic model of a spatiotemporally chaotic partial differential equation using only the manifold coordinates. Finally, we demonstrate that our framework is robust to hyperparameter choices.

5.1 Introduction

Nonlinear dissipative partial differential equations (PDEs) are ubiquitous in describing phenomena throughout physics and engineering that display complex nonlinear behaviors, out-of-equilibrium dynamics, and even spatiotemporal chaos. Although the state space of a PDE is formally infinite-dimensional, the long-time dynamics of a dissipative system are known or suspected to collapse onto a finite-dimensional invariant manifold, which we will denote $\mathcal{M}$. [85, 189, 218]. The same idea holds for high-dimensional dissipative systems of ordinary differential equations (or discretized PDEs), and in any case, data from any system under consideration will be finite-dimensional, so we will consider manifolds of dimension d_m embedded in an ambient space $\mathbb{R}^{d_u}$, where often $d_m \ll d_u$. That is to say, in order to accurately describe the manifold, and thus the underlying dynamics of the system, only d_m independent coordinates are needed (at least locally). In general, no global coordinate representation of dimension d_m is available, but Whitney's theorem guarantees that a global representation with *embedding dimension* $d_e \leq 2d_m$ can be found. Alternately, in principle, an atlas of overlapping charts with dimension d_m can be constructed to provide local d_m-dimensional representations [46, 111]. For the most part, we address the task of learning minimal *global* manifold representations (although we will show that our work can be extended into local representations), and consider cases where $d_e = d_m$.

Obtaining a minimal manifold coordinate description for these systems based on an analysis of data from that system is ideal for a number of dynamical applications such as state-

space identification, reduced-order modeling and control, and system interpretability, as well as many other downstream tasks such as classification. However, estimating the underlying dimensionality of a data set and obtaining the manifold coordinate transformations is generally a nontrivial task. Given access only to data represented in the high-dimensional ambient space of a system, the challenge becomes the following: 1) determining d_m, 2) constructing a coordinate system describing points in $\mathcal{M}$, and 3) obtaining the mapping functions $\mathcal{E} : \mathbb{R}^{d_u} \to \mathbb{R}^{d_m}$ and $\mathcal{D} : \mathbb{R}^{d_m} \to \mathbb{R}^{d_u}$. In the interest of identifying and modeling the underlying core dynamics of these systems, our aim is to address these three challenges using a single framework trained on high-dimensional ambient data alone.

These three challenges have been tackled by an extensive variety of methodologies, but we emphasize that rarely are all three challenges addressed simultaneously in a single framework –often only the first challenge of identifying the manifold dimension is attempted. For complex dynamical systems, many of these methods developed in systems theory rely on high-precision analyses and access to the underlying equations. For example, Yang and Radons [214], Yang et al. [215] estimated the manifold dimension of the Kuramoto-Sivashinsky equation (KSE), a formally infinite-dimensional system with finite-dimensional dynamics, for a range of parameters using covariant Lyapunov vectors, monitoring when the Lyapunov spectrum of the system begins to rapidly fall. Ding et al. [36] corroborated the dimension of invariant manifold containing the long time dynamics of the KSE for a domain size of $L = 22$ via a Floquet mode approach applied to organized unstable periodic orbits identified in the system. These methods require high precision solutions of the governing equations and access to very specific dynamical data (e.g. periodic orbits) that for more complex systems such as the Navier-Stokes equations are nontrivial or even intractable tasks. Furthermore, these methods are not applicable when the governing equations are not known or when data is collected from general time series rather than precisely prescribed trajectories. For these reasons, these methods will not be the focus of this work.

Towards more generalized and data-driven approaches, the task of estimating the number

of degrees of freedom required to represent a set of data without loss of information has been explored in the fields of pattern recognition, information sciences, and machine learning. These methods produce estimates of d_m (or some upper bound) either using global or local analyses of the dataset.

Global approaches tackle this challenge in several ways. Linear global projection methods, such as Principal Component Analysis (PCA) [98] and its variants (e.g. Sparse PCA [223] and Bayesian PCA [7]), determine a linear subspace in which the projection of the data minimizes some projection error. These methods are useful in that not only are they computationally tractable, they also directly provide the mapping functions to the low dimensional representation. However, as they are linear methods, they generally overestimate d_m, since representing data on a curved manifold of dimension d_m will require at least $d_m + 1$ coordinates.

Nonlinear PCA, or deep autoencoders in general, deal with nonlinearity using neural networks tasked with autoassociation [99]. Autoencoders can be used to estimate d_m by tracking the mean squared reconstruction error (MSE) as a function of the bottleneck dimension d_z of the networks. If the MSE significantly drops above a threshold value of d_h, one can infer that the minimum number of degrees of freedom needed to represent the system data is reached. In applications toward complex high-dimensional dynamical systems including discretized dissipative PDEs, Linot and Graham [122, 123] and Vlachas et al. [203] used undercomplete autoencoders to estimate the manifold dimension of data from the KSE this way. However, as system complexity and dimensionality increase, the MSE drop off becomes less and less sharp [16, 94, 122, 124]. Additionally, a practical drawback of this type of approach is it requires training separate networks with a range of d_z.

Towards more automated autoencoder-based frameworks, several works have incorporated the heuristic false-nearest neighbor algorithm (FNN) [100] to target the embedding dimension for state-space reconstructions of a time-series signal that come from systems with manifolds with $d_m = 3$. Specifically, Gilpin [59] incorporated an additional loss based

on the FNN metric to penalize the encoder outputs. This formulation, however, penalizes both redundant latent variables as well as those capturing the manifold, leading to high sensitivity to the regularization term [209]. To address this, Wang and Guet [209] incorporated an attention map to explicitly mask superfluous latent variables based on the FNN metric. Practically, these frameworks require repeatedly computing Euclidean distances between training data points for a range of embedding dimensions at each iteration of training, which is not ideal for systems of increasing complexity and dimensionality. Furthermore, the FNN targets the embedding dimension, which is often higher than the manifold dimension.

Several notable methods of dimensionality reduction tools utilize local computations. A large portion of these methods belong to the class of methods known as multidimensional scaling (MDS), which are concerned with preserving some local or pairwise characteristics of the data. These include Laplacian Eigenmaps [5], t-distributed stochastic neighbor embedding [193], ISOMAP [191] and Locally Linear Embeddings [166]. However, a major distinction between these methods and the goals of this chapter is these methods require choosing a manifold dimension beforehand to embed the data into, and are generally applied towards data visualization applications. ISOMAP [191], while capable of providing an "eyeballed" estimate of d_m via error curves, struggles to handle higher dimensionality data [15]. Several other principled d_m estimation methods, such as the Levina-Bickel method [114] and the Little-Jung-Maggioni method (multiscale SVD) [128], estimate d_m by averaging estimates made over neighborhoods of data points. Multiscale SVD and the Levina-Bickel methods are further discussed below. Importantly, all of these local methods lack one or more of the following features: the ability to estimate d_m, project new out-of-sample data points into manifold coordinates, or provide a coordinate system for the d_m-dimensional representation.

In this work, we address the three aforementioned challenges using a deep autoencoder framework that drives the rank of the covariance of the data in the latent representation to a minimum. This rank will be equal to the dimension d_m of the manifold where the data lies. Our framework utilizes two low-rank driving forces. The first is known as implicit

regularization, which is a phenomenon observed in gradient-based optimization of deep linear networks (i.e. multiple linear layers in series) leading to low-rank solutions [74]. Although a series of linear layers is functionally and expressively identical to a single linear layer, the learning dynamics of the two are different. The mechanisms of this phenomenon are an ongoing area of research with primary focus in matrix [2, 74] and tensor factorization [161]. Importantly, it has been observed that implicit regularization does not occur for unstructured datasets such as random full-rank noise [2], indicating that the phenomenon depends on the underlying structure of the data. Recently, implicit regularization has been extended to autoencoders (Implicit Rank Minimizing autoencoders, IRMAE) to learn low-rank representations, improving learning representations for image-based classification, and generative problems by Jing et al. [97], whose observations form the foundation in this work.

The second low-rank driving force is L_2 regularization, often referred to by its action when combined with gradient descent: weight-decay. Weight-decay is a popular weight regularization mechanism in deep learning that forces the network to make trade-offs between the standard loss L of the learning problem with properties of the weights of the network, θ,

$$\mathcal{L} = L + \frac{\lambda}{2}\|\theta\|_p^2. \tag{5.1}$$

Recently, Mousavi-Hosseini et al. [139] showed that in two-layer neural-networks the first layer weights converge to the minimal principal subspace spanned by a target function only when online stochastic gradient descent (SGD) is combined with weight decay. The authors found that weight-decay allowed SGD to avoid critical points outside the principal subspace. Here we demonstrate a similar synergistic result when weight-decay is combined with implicitly-regularized autoencoders.

The goal of the present work is to demonstrate that implicit regularization combined with weight-decay in deep autoencoders, an approach we call *Implicit Rank Minimizing Autoencoder with Weight-Decay* or *IRMAE-WD*, can be applied toward datasets that lie on

a manifold of $d_m < d_u$, to 1) estimate the dimension of the manifold on which the data lie, 2) obtain a coordinate system describing the manifold, and 3) obtain mapping functions to and from the manifold coordinates. We highlight that IRMAE-WD produces by construction an orthogonal manifold coordinate basis organized by variance, and does not rely on extensive parameter sweeps of networks [122, 203] or external estimators [59, 209] – only a good upper-bound guess of the manifold dimension is needed. (And if this guess is not good, the results of the analysis will indicate so.) These properties make the IRMAE-WD framework a natural first step for data-driven reduced-order/state space modeling and many other downstream tasks.

The remainder of this chapter is organized as follows: In Sec. 5.2 we describe the IRMAE-WD framework. In Sec. 5.3.1 we apply it to a zoo of datasets ranging from synthetic data sets to physical systems that exhibit complex chaotic dynamics including the Lorenz system and the Kuramoto-Sivashinsky equation. In Sec. 5.3.2 we overview performance sensitivity to hyperparameters. In Sec. 5.3.3 we compare the framework's ability to estimate the underlying dimensionality of complex datasets against several state-of-the-art estimators. In Sec. 5.3.4, we demonstrate how this framework can be naturally extended for downstream tasks such as state-space modeling and dynamics forecasting in the manifold coordinates. Finally, in Sec. 5.3.5, we examine the training dynamics of IRMAE-WD to isolate the origins of low-rank in both "space" (i.e. how the data representation is transformed as it passes through the architecture) and "time" (i.e. how the data representation is transformed as training progresses). We glean insight into network learning and, with an analysis of a special case of a linear autoencoder, provide a rationale for how implicit regularization and weight decay achieve a synergistic effect. Appendix 5.5 provides a summary of our architectures, Appendix 5.6 details an application to the MNIST handwriting dataset, and Appendix 5.7 contains the analysis of the linear autoencoder that provides some theoretical understanding of the observed performance of the method.

5.2 Formulation

Our proposed framework uses an autoencoder architecture. Autoencoders are composed of two subnetworks, the encoder and decoder, which are connected by a latent hidden layer. For dimensionality reduction problems, this latent hidden layer is often a size-limiting bottleneck that explicitly restricts the number of degrees of freedom available to represent the input data. This architecture, which we will denote as a standard autoencoder, forces the encoder network, $z = \mathcal{E}(u;\theta_E)$, to compress the input data, $u \in \mathbb{R}^{d_u}$, into a compact representation, $z \in \mathbb{R}^{d_z}$, where $d_z < d_u$. The decoder, $\tilde{u} = \mathcal{D}(z;\theta_D)$, performs the inverse task of learning to reconstruct the input, $\tilde{u} \in \mathbb{R}^{d_u}$, from the compressed representation, z. The autoencoder is trained to minimize the mean squared error (MSE) or reconstruction loss

$$\mathcal{L}(u;\theta_E,\theta_D) = \langle ||u - \mathcal{D}(\mathcal{E}(u;\theta_E);\theta_D)||_2^2 \rangle \tag{5.2}$$

Here $\langle\cdot\rangle$ is the average over a training batch and θ_i corresponds to the weights of each subnetwork. We then deviate from the standard autoencoder architecture by adding an additional linear network, $\mathcal{W}(\cdot;\theta_W)$, between the encoder network and decoder: i.e. $z = \mathcal{W}(\mathcal{E}(u;\theta_E);\theta_W)$, where $\mathcal{W}(\cdot;\theta_W)$ is composed of n trainable linear weight matrices denoted as W_j (i.e. linear layers) of size $d_z \times d_z$ in series, as was done in Jing et al. [97]. Although $\mathcal{W}$ adds additional trainable parameters compared to a standard autoencoder, it does not give the network any additional expressivity, as linear layers in series have the same expressivity as a single linear layer. Thus, the effective capacity of the two networks are identical. Importantly we train the framework with weight-decay shown in Fig. 5.1a with the autoassociation task,

$$\mathcal{L}(u;\theta_E,\theta_W,\theta_D) = \langle ||u - \mathcal{D}(\mathcal{W}(\mathcal{E}(u;\theta_E);\theta_W);\theta_D)||_2^2 \rangle + \frac{\lambda}{2}\|\theta\|_2^2. \tag{5.3}$$

Here we contrast IRMAE-WD from typical autoencoders tasked with finding minimal or low-dimensional representations with two distinctions. First, rather than parametrically sweep d_z, as is usually done with standard autoencoders, we instead guess a single $d_z > d_m$, and

rely on implicit regularization and weight-decay to drive the latent space to an approximately minimal rank representation. If this rank is found to equal d_z, then d_z can be increased and the analysis repeated.

Once the regularized network is trained, we perform singular value decomposition on the covariance matrix of the latent data matrix Z (i.e. the encoded data matrix) to obtain the matrices of singular vectors U and singular values S, shown in Fig. 5.1b. Here, the number of significant singular values of this spectrum gives an estimate of d_m, as each significant value represents a necessary coordinate in representing the original data in the latent space. (More precisely, we get an estimate of d_e, although as we illustrate below, the analysis can be performed on subsets of data to find d_m in the case $d_m < d_e$.)

Shown in Fig. 5.1c, we can naturally project z onto U^T to obtain $U^T z = h^+ \in \mathbb{R}^{d_z}$ where each coordinate of h^+ is orthogonal and ordered by contribution. As $UU^T = I$, we can recover the reconstruction of z, $\tilde{z}$, by projecting h^+ onto U. Importantly, as the framework automatically discovers a latent space in which the encoded data only spans d_m (reflected in the number of significant singular values), the data only populates the latent space in the directions of the singular vectors corresponding to those significant singular values. In other words, the encoded data does not span in the directions of the singular vectors whose corresponding singular values are approximately zero and $UU^T z \approx \hat{U}\hat{U}^T z$ holds, where $\hat{U}$ are the singular vectors truncated to only include those whose singular values are not approximately zero.

This observation allows us to isolate a minimal, orthogonal, coordinate system by simply projecting z onto $\hat{U}$ to obtain our minimal representation $\hat{U}^T z = h \in \mathbb{R}^{d_m}$, which we refer to as the manifold representation, shown in Fig. 5.1d. As $UU^T z \approx \hat{U}\hat{U}^T z$ and $u \approx \mathcal{D}(\mathcal{W}(\mathcal{E}(u;\theta_E);\theta_W);\theta_D)$, we can transform between between our manifold representation, h, and the ambient representation, u with minimal loss.

To glean insight into the learning mechanism of autoencoders with implicit regularization and weight-decay in a tractable manner, in Appendix 5.7 we analyze the dynamics of gradient

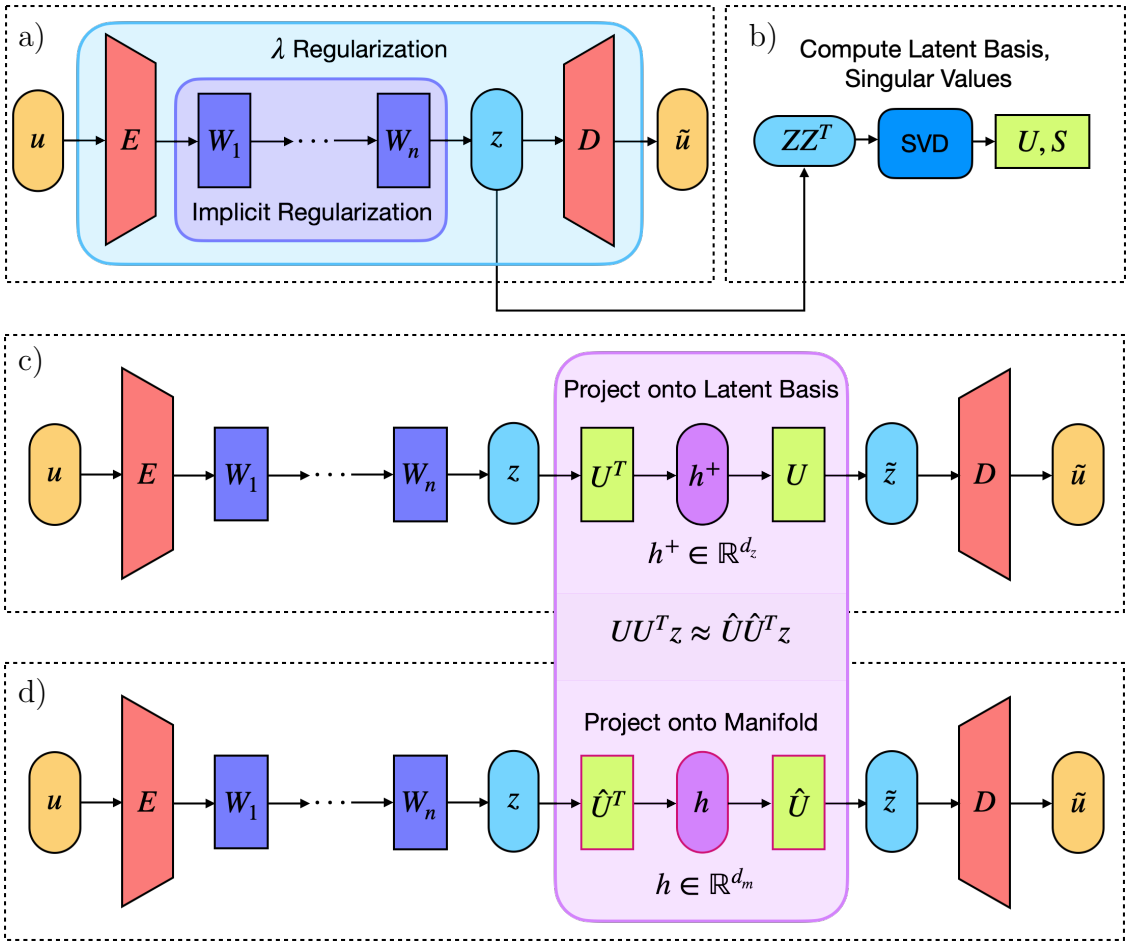

Figure 5.1: Our implicit and λ weight-decay regularized deep autoencoder framework a) network architecture with regularization mechanisms, b) singular value decomposition of the covariance of the learned latent data representation Z, c) projection of latent variables onto manifold coordinates d) isolated projection of latent variables onto manifold coordinates.

descent for a *linear* autoencoder acting on data whose covariance has rank $r(= d_m)$. For this case, there is a family of solutions for the weight matrices in which they all have rank r. For gradient descent near convergence to these low-rank solutions, implicit regularization drives the weights exponentially fast toward a low-rank solution. The analysis shows that there is a "collective" mode of decay toward the low-rank solution family in which all of the weight matrices are coupled. The decay rate for this mode scales as $2 + n$, where n is the number of internal (square) linear layers. In the absence of weight decay, there are directions with eigenvalues of zero that do not decay with training. When weight decay is added, these formerly zero eigenvalues become negative, allowing decay from all directions to the low-rank solution. In Sec. 5.3.5, we empirically observe the gradient updates and weight matrices of the linear layers of our nonlinear network exhibiting this behavior.

An important practical detail during application of the present method is the choice of optimizer for the SGD process. We found that it is very important to use the AdamW optimizer [131] rather than the standard Adam optimizer. This distinction is important because direct application of weight decay (L_2 regularization) in the commonly used Adam optimizer leads to weights with larger gradient amplitudes being regularized disproportionately [131]. AdamW decouples weight decay from the adaptive gradient update. We have found that the usage of the base Adam optimizer with L_2 regularization can lead to high sensitivity to parameters and spurious results.

5.3 Results

5.3.1 Manifold Dimension Estimates: Example Systems

We now investigate IRMAE-WD applied to a zoo of datasets of increasing complexity, ranging from linear manifolds embedded in finite-dimensional ambient spaces to nonlinear manifolds embedded in formally infinite-dimensional ambient spaces.

Data linearly embedded in a finite-dimensional ambient space

We first benchmark IRMAE-WD against a simple data set consisting of 5-dimensional noise linearly embedded in an ambient space of 20 dimensions. Because this dataset exactly spans 5 orthogonal directions and is linearly embedded, Principal Component Analysis (PCA) is able to extract d_m from the data, which can be identified via the singular value spectrum of covariance of the data matrix. Shown in Fig. 5.2a are the singular values obtained from PCA, from the learned latent variables of IRMAE without and with weight-decay, and a standard AE that is architecturally identical to IRMAE-WD without any regularization (i.e. no W and $\lambda = 0$). For the standard autoencoder, while the singular values σ_i drop slightly for index $i > 5$, the spectrum is broad and decays slowly, indicating that the learned latent representation is essentially full-rank. In other words, the standard autoencoder, when given excess capacity in the bottleneck layer, will utilize all latent variables available to it. In contrast, for IRMAE-WD, the singular values for $i > 5$ drop to $\sim 10^{-16}$, just as in the case of PCA. This indicates that IRMAE-WD is able to automatically learn a representation that isolates the minimal dimensions needed to represent the data.

We further highlight here two important observations: 1) an autoencoder with weight decay alone is insufficient in learning a sparse representation – it behaves very similarly to the standard autoencoder, and 2) an autoencoder with implicit regularization alone, as applied in Jing et al. [97], yields a sharp drop in σ_i for $i > 5$, but not nearly so dramatic as when both linear layers and weight decay are implemented. This phenomenon is addressed in Sections 5.3.5 and Appendix 5.7.

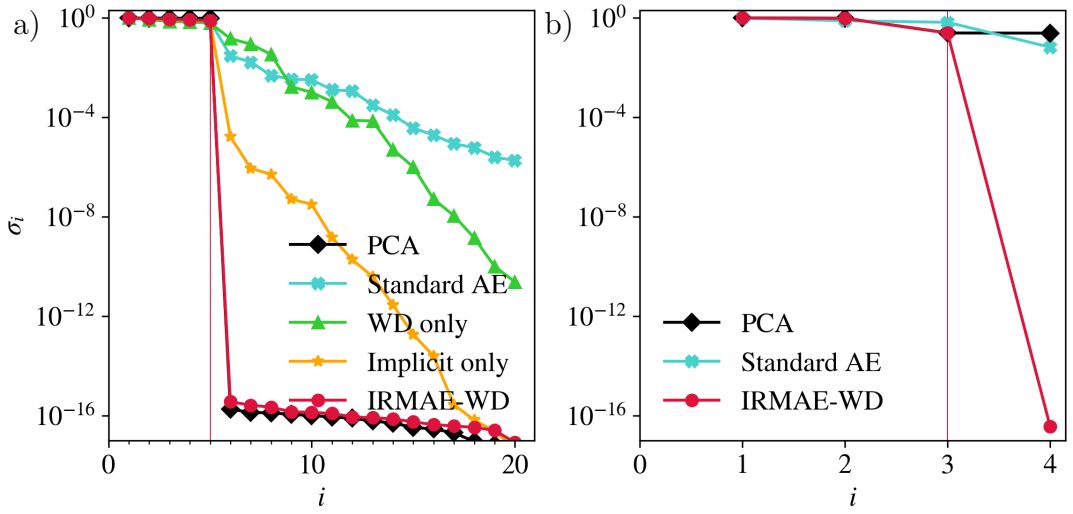

Figure 5.2: Normalized singular values, σ_i, of latent data covariances of various AE methods applied to a) a 5-dimensional linear manifold embedded in $\mathbb{R}^{20}$ and b) a 3-dimensional nonlinear manifold embedded in $\mathbb{R}^4$. The spectra obtained from PCA and a standard AE with no regularization are provided. The value of d_m is marked by the vertical red guide line.

Nonlinearly embedded finite-dimensional system: The Archimedean Spiral Lorenz

We now turn our attention to data from nonlinear finite-dimensional dynamical systems with nonlinear embedded manifolds. Specifically, we take the Lorenz '63 system [130],

$$\begin{aligned} \dot{x} &= \sigma(y - x) \\ \dot{y} &= x(\rho - z) - y \\ \dot{z} &= xy - \beta z \end{aligned} \tag{5.4}$$

which exhibits chaotic dynamics in $\mathbb{R}^3$ and embed this system nonlinearly in $\mathbb{R}^4$ by wrapping the data set around the Archimedean spiral using the following mapping:

$$[x, y, \alpha z \cos(\alpha z), \alpha z \sin(\alpha z)] \to [u_1, u_2, u_3, u_4],$$

with $\alpha = 0.2$.

For parameters $\sigma = 10, \rho = 28, \beta = 8/3$, the Lorenz '63 exhibits chaotic dynamics. In

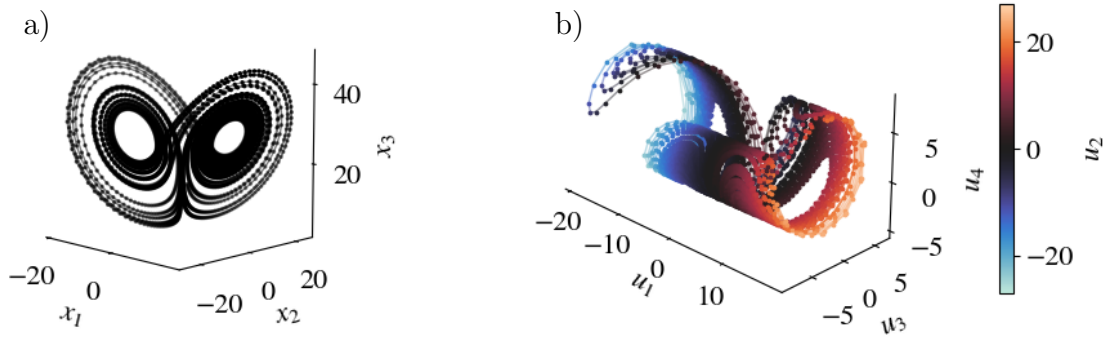

Figure 5.3: Dynamics of the a) 3-dimensional Lorenz '63 equation and b) the 4-dimensional Archimedean Lorenz equation. The color corresponds to the variable, u_2 in the embedding, while the spatial coordinates correspond to u_1, u_3, and u_4.

other words, the underlying dynamics of this system live on a nonlinear 3-dimensional manifold that is nonlinearly embedded in a 4-dimensional ambient space. We show in Fig. 5.2b that IRMAE-WD correctly determines that this system can be minimally represented by 3 latent variables. In contrast, the application of PCA fails to identify the underlying structure of the data. Here, the PCA spectrum does not give a correct estimate of d_m because inherently a linear method cannot minimally capture the nonlinearity/curvature of the manifold. Finally, a standard AE with $d_z > d_m$ also fails to automatically learn a minimal representation as it finds a full-rank data covariance in the latent space.

Global manifold estimates vs local estimates: quasiperiodic dynamics on a 2-torus

We now turn our attention to a trajectory in $\mathbb{R}^3$ traversing the surface of a 2-torus with poloidal and toroidal speeds that lead to quasiperiodic dynamics, as visualized in Fig. 5.4a. Given infinite time, the particle will densely cover the surface of the torus. Although this system lives on a two-dimensional manifold, the topology of this manifold is nontrivial and a single global representation is not possible to obtain [46]. Here we apply IRMAE-WD to this dataset, which consists of snapshots of the three coordinates along a trajectory.

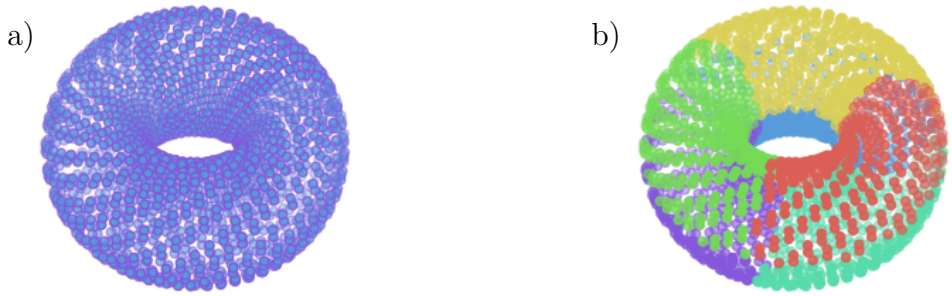

Figure 5.4: Quasiperiodic dynamics on a torus: a) global b) local patches.

In this example, we also use an overcomplete network design, $z \in \mathbb{R}^{10}$, to highlight that even when $d_z > d_u$, excess degrees of freedom are still correctly eliminated. We show in Fig. 5.5a that when IRMAE-WD is tasked with learning a global representation by training over the entire dataset, it (correctly) obtains a 3-dimensional latent space – the embedding dimension of the manifold is $d_e = 3$. However, as described by Floryan and Graham [46], by decomposition of the manifold into an atlas of overlapping charts, the intrinsic dimension of the manifold containing the data can be captured. In Fig. 5.5b, we show IRMAE-WD applied to the same dataset after being divided into patches found using k-means clustering, illustrated in Fig. 5.4b. We show that for each subdomain, IRMAE-WD automatically learns a minimal 2-dimensional representation of the data while simultaneously discarding the remaining superfluous degrees of freedom. In this manner, IRMAE-WD can be deployed on local regions of data to make estimates of the intrinsic dimension.

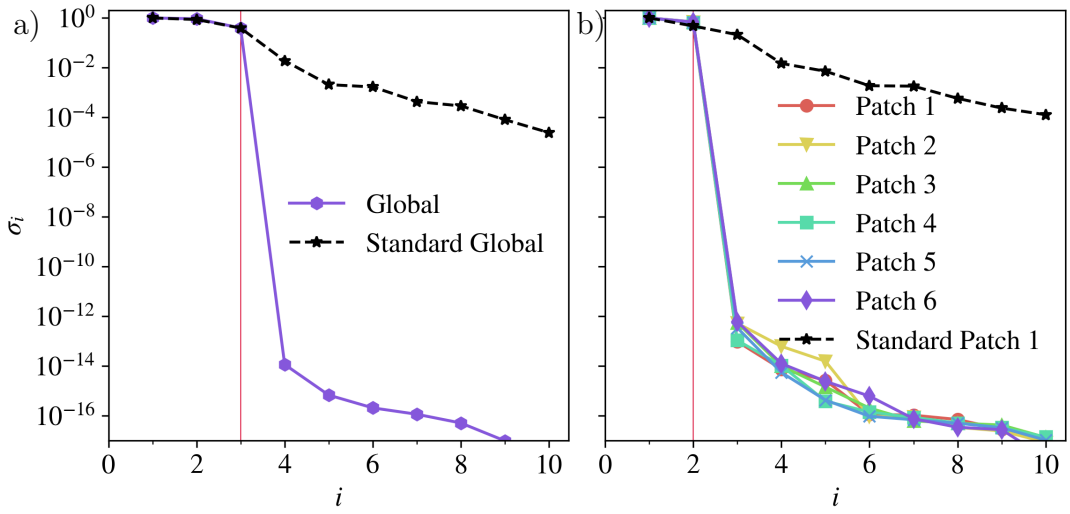

Figure 5.5: Normalized singular values, σ_i, of learned latent spaces from IRMAE-WD applied to the a) global torus dataset and b) local patches of the torus dataset. Results for a standard AE latent space is shown in black. The value of d_m is marked by the vertical red guide line.

Nonlinear manifold in an "infinite-dimensional" system: The Kuramoto-Sivashinsky Equation

We now turn our attention to dissipative nonlinear systems that are formally "infinite" dimensional. Here, we investigate the 1D periodic Kuramoto-Sivashinsky equation (KSE):

$$\frac{\partial v}{\partial t} = -v\frac{\partial v}{\partial x} - \frac{\partial^2 v}{\partial x^2} - \frac{\partial^4 v}{\partial x^4} \tag{5.5}$$

in a domain of length L with periodic boundary conditions. For large L, this system exhibits rich spatiotemporal chaotic dynamics which has made it a common test case for studies of complex nonlinear systems. To analyze this formally "infinite" dimensional system, state snapshots will consist of sampled solution values at equidistant mesh points in the domain. We apply IRMAE-WD to extract the dimension of the underlying manifold for dynamics for a range of domain sizes, focusing first on $L = 22$, which exhibits spatiotemporal chaotic dynamics and has been widely studied. An example trajectory of this system is shown in Fig. 5.6a. This system, although formally infinite-dimensional, has dynamics dictated by a

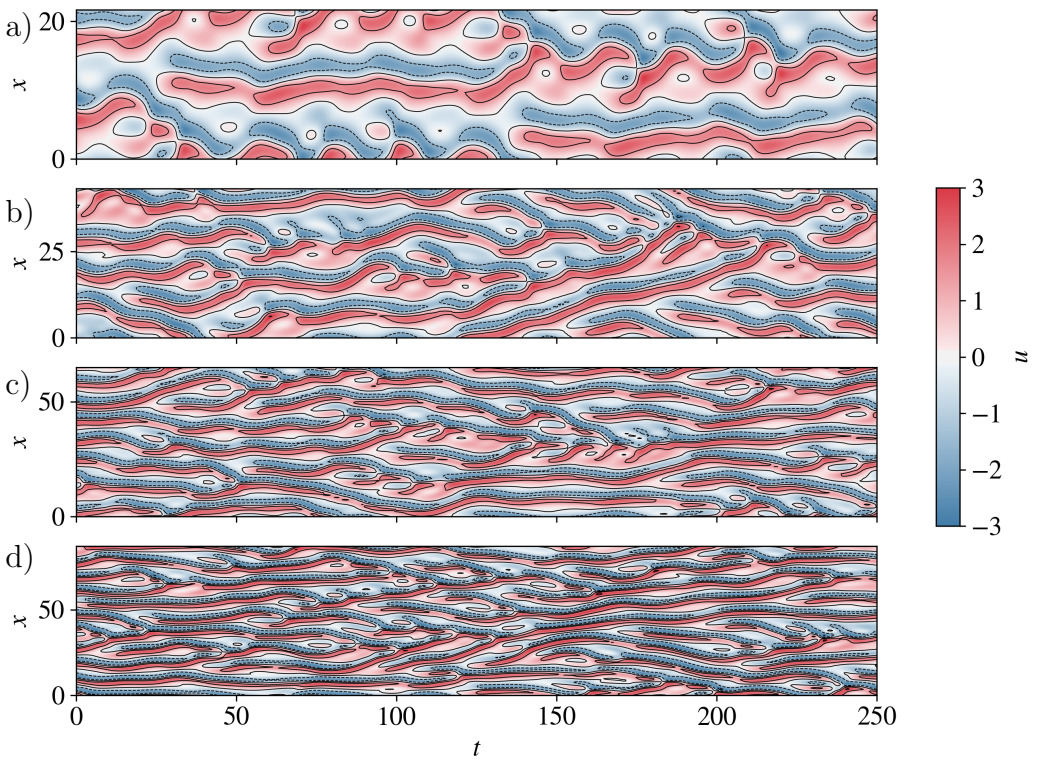

Figure 5.6: Typical evolutions for the KSE in domain sizes of a) $L = 22$, b) $L = 44$, c) $L = 66$, and d) $L = 88$.

nonlinearly embedded 8-dimensional manifold, as indicated by a variety of methodologies [36, 122, 188, 215]. Using a data set comprised of 40,000 snapshots sampled on 64 mesh points, and choosing a bottleneck layer dimension $d_z = 20$, we show in Fig. 5.7a that the singular values coming from IRMAE-WD drop dramatically above an index of 8, indicating that we have automatically and straightforwardly learned a latent space of dimension $d_m = 8$. By contrast, neither PCA nor a standard AE leads to a substantial drop in singular values over the whole range of indices.

For increasing domain sizes of of the KSE, the spatiotemporal dynamics of the system increases in complexity. Fig. 5.6b-d show space-time plots of the dynamics for $L = 44, 66$, and 88, sampled on a uniform spatial mesh of 64, 64, and 128 points, respectively. Fig. 5.7b-c show

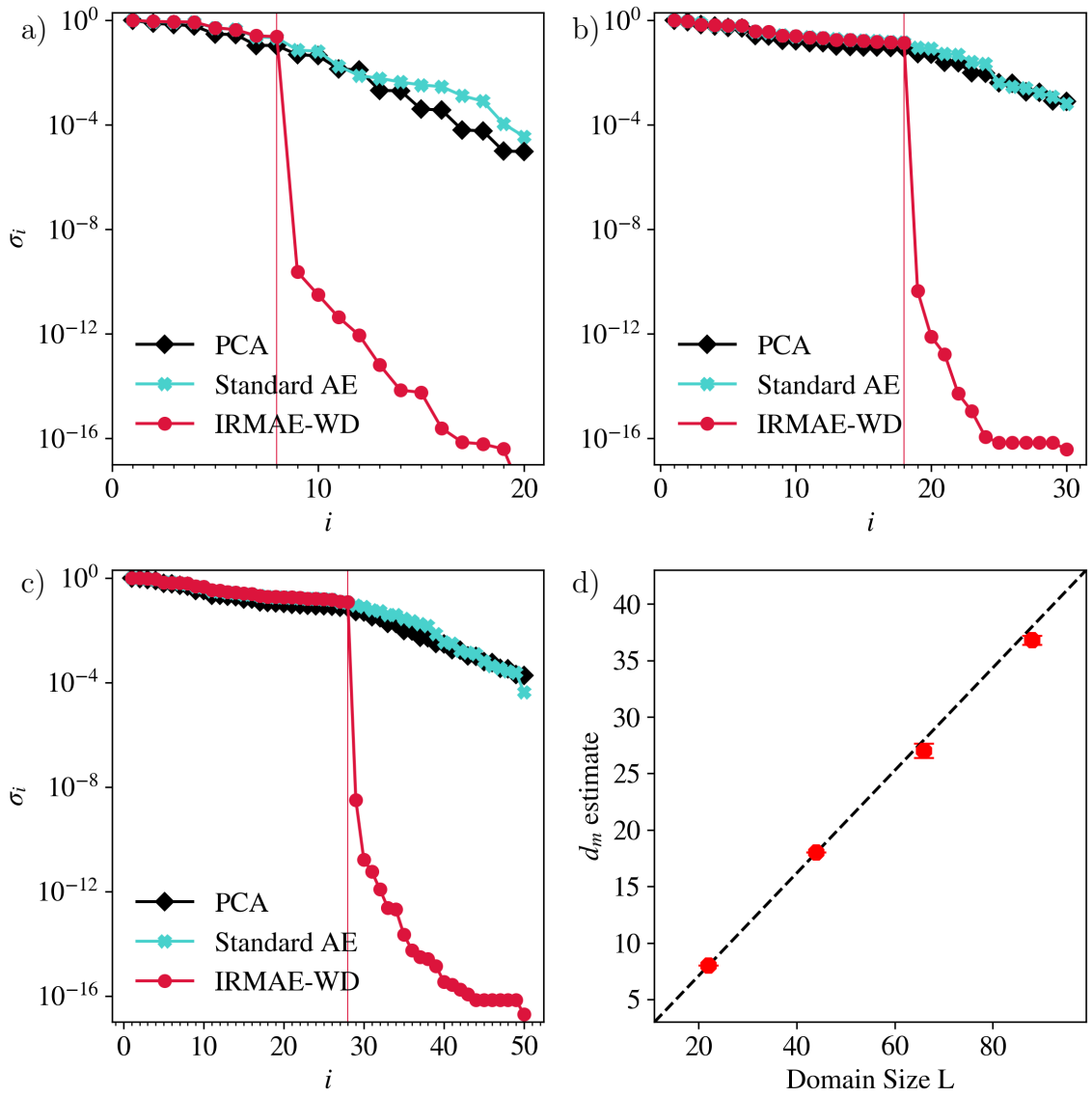

Figure 5.7: Singular values, σ_i, of IRMAE-WD learned latent spaces for the KSE a) $L = 22$, b) $L = 44$, c) $L = 66$, and d) estimate of d_m averaged over 5 randomly initialized models as a function of L, with the standard deviations represented by the error bars. In a)-c), the spectra obtained from PCA and a standard AE with no regularization are also shown, and the value of d_m is marked by the vertical red guide line.

the singular value spectra of the latent space covariances for $L = 44$ and 66, again showing a drop of > 10 orders of magnitude at well-defined index values, indicating manifold dimensions $d_m = 18$ and 28, respectively. We highlight here that previous autoencoder methods [122, 123], using the trend in MSE with d_z to estimate d_m, struggle to make distinctions in the manifold dimension for these domain sizes, while IRMAE-WD yields a well-characterized value. Prior works relying on high-precision analyses of the dynamics based on detailed and complex trajectory analyses have suggested that the manifold dimension for the KSE scales linearly with the domain length L [188, 215]. In Fig. 5.7d we show the trend in d_m vs. L as determined with IRMAE-WD: we are able to very straightforwardly recover the linear scaling without access to the underlying governing equations or periodic solutions.

5.3.2 Robustness and Parameter Sensitivity

In the following section we overview parametric robustness of IRMAE-WD, focusing on the KSE $L = 22$ dataset. We choose this dataset as it comes from a nonlinear, high-dimensional system governed by dynamics on a nonlinear manifold and is considerably more complex than typical benchmark systems.

We first investigate the accuracy of the estimate of d_m, where the correct value, based on consistent results from many sources, is taken to be $d_m = 8$. Fig. 5.8a shows the dimension estimate as a function of number of linear layers n and weight decay parameter λ, with the bottom row of the plot corresponding to the case $n = 0$ of a standard autoencoder with L_2 regularization. We highlight that for a broad range of n and λ the framework is capable of accurately estimating d_m. It is not until there is significant regularization in terms of both n and λ that the framework begins to fail. In the absence of implicit regularization with linear layers the autoencoder cannot predict d_m at all. Shown in Fig. 5.8b is the same parameter sweep characterized by test MSE performance. This quantity is also relatively insensitive to choice of parameters, and the regularized models operating with effectively d_m degrees of freedom in the representation achieve comparable reconstruction errors to

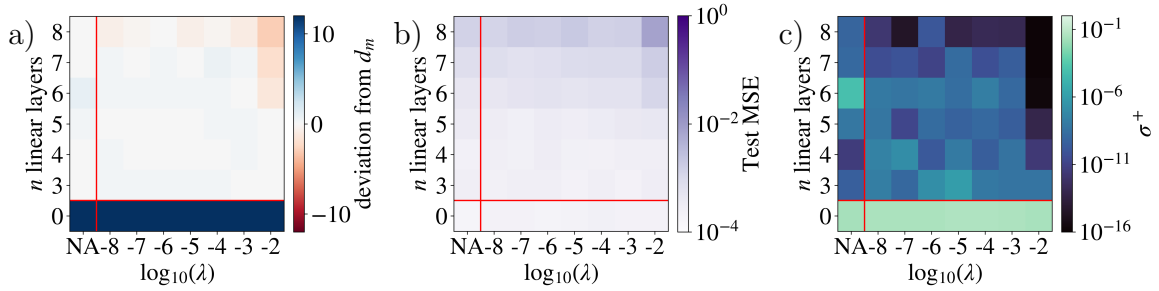

Figure 5.8: Parametric sweep in n and λ of models trained over the KSE $L = 22$ dataset with various degrees of implicit and weight regularization. The colors correspond to: a) average deviation from d_m over 3 models, b) average test MSE over 3 models, c) lowest fraction of trailing singular values of 3 models (lower is better). In the leftmost column, labeled NA, $\lambda = 0$. The lower left corner in each plot corresponds to a standard autoencoder.

standard autoencoders (bottom left corner). Finally, for an ideal regularized model, singular values with indices greater than d_m are zero, but practically this is not the case. In Fig. 5.8c, we quantify the fraction of total variance in the representation coming from singular values from the tail of the spectrum, i.e. with index greater than d_m: $\sigma^+ = \sum_{i=d_{m+1}}^{d_z} \sigma_i / \sum_{j=1}^{d_z} \sigma_j$. We highlight here that for a broad range of n and λ, the trailing singular values contribute on the order of 10^{-9} of the total variance or energy, while the unregularized models contribute a nontrivial 10^{-1}. Finally, we comment that we did not observe strong dependence of the choice of d_z on the results, as long as $d_z > d_m$.

5.3.3 Comparison to other methods

In this section, we compare IRMAE-WD to two state-of-the-art estimators: Multiscale SVD (MSVD) [128] and the Levina-Bickel method [114] as these methods are designed to provide a direct estimate of the manifold dimension from data. We first compare to the MSVD method, as it is also completely data-driven and also relies on analyzing singular value spectra of the data. MSVD estimates d_m by tracking the ensemble average of singular value spectra obtained from a collection of local neighborhoods of data as a function of the size of

the neighborhood radius, r. Development of gaps in the singular value spectra as r increases coincides with a separation between directions on the manifold and those due to curvature. A gap in the spectrum is presumed to indicate the manifold dimension, as PCA does for data on a linear manifold. We revisit the KSE $L = 22$ and $L = 44$ datasets as they are nontrivial complex systems with nonlinear manifolds. In Fig. 5.9a and Fig. 5.9b we show the MSVD method applied to these datasets. We highlight here that MSVD, given these datasets, is unable to unambiguously identify d_m; Rather than one gap, there are multiple gaps in the spectra, as indicated by the arrows. This is likely due to a key limitations of MSVD, which is that it requires data in small enough r neighborhoods to accurately approximate the highly nonlinear manifold as flat. I.e. in order to work in the limit of very small neighborhoods, MSVD requires an ensemble of data points to have a sufficient number of neighboring points at very small r. In our MSVD application, we were unable to access small values of r without encountering neighborless point cloud samples. Many complex dynamical systems do not uniformly populate their underlying manifolds, resulting in regions of high and low density – indeed, data points on a chaotic attractor will be fractally, rather than uniformly, distributed. As a result, it is difficult to collect dynamical data in which the manifold is represented with uniform density or to collect enough data such that low probability regions are dense when natural occurrences in these regions are low. IRMAE-WD does not suffer from these limitations.

We also apply the Levina-Bickel method to these same datasets. This method utilizes a maximum likelihood framework in estimating the dimensionality of the data from local regions [114]. In our application we fix the number of neighbors, as suggested by Levina and Bickel [114], rather than fixing the neighborhood radius. This method also fails to provide reliable estimates given our datasets. We summarize this section with our findings in Table 5.1. For the datasets considered, the Levina-Bickel method appears to underestimate the dimensionality while MSVD tends to give ambiguous estimates.

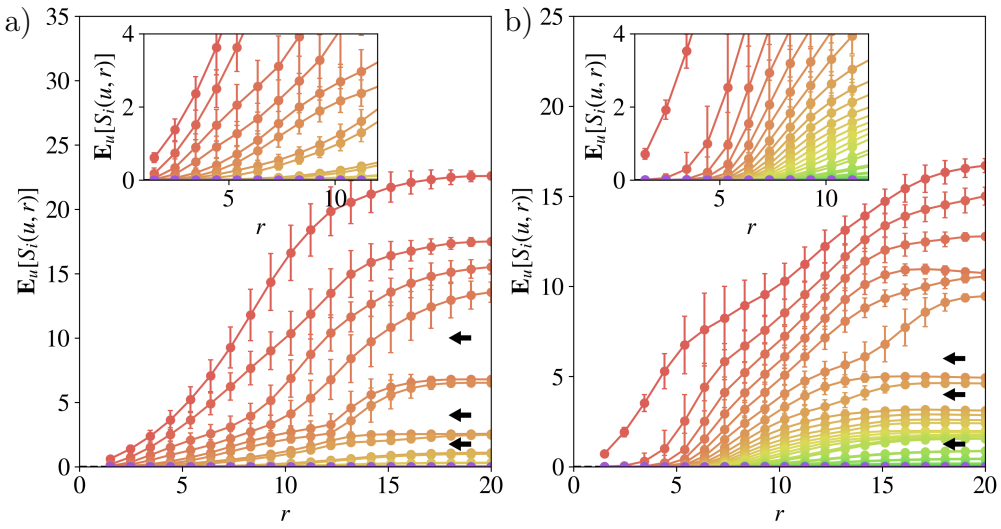

Figure 5.9: Ensemble MSVD singular values, S_i, as a function of sampling neighborhood radius r on the KSE a) $L = 22$ dataset and b) $L = 44$ dataset. The color of the lines corresponds to the modal index of the spectra. The arrows mark the gaps that appear in the spectra, providing an estimate of underlying dimensionality.

Table 5.1: Estimates of d_m with various methods.

Dataset	d_m	Multiscale SVD	Levina-Bickel	IRMAE-WD
Arch. Lorenz	3	2	2.09	3
KSE $L = 22$	8	6-8	3.99	8
KSE $L = 44$	18	8-20	7.00	18

5.3.4 Reduced-order state-space forecasting in the manifold coordinates

As noted above and illustrated in Fig. 5.1, projection of the latent space data z onto the first d_m singular vectors of its covariance yields the manifold representation $h \in \mathbb{R}^{d_m}$. We can map data snapshots in the ambient space to this manifold coordinate representation by simply extending our definitions of encoding and decoding to h:

$$\begin{aligned} h &:= \mathcal{E}_h(u; \theta_E, \theta_W, \hat{U}^T) = \hat{U}^T \mathcal{W}(\mathcal{E}(u; \theta_E); \theta_W) \\ \tilde{u} &:= \mathcal{D}_h(h; \theta_D, \hat{U}^T) = \mathcal{D}(\hat{U} h; \theta_D) \end{aligned} \tag{5.6}$$

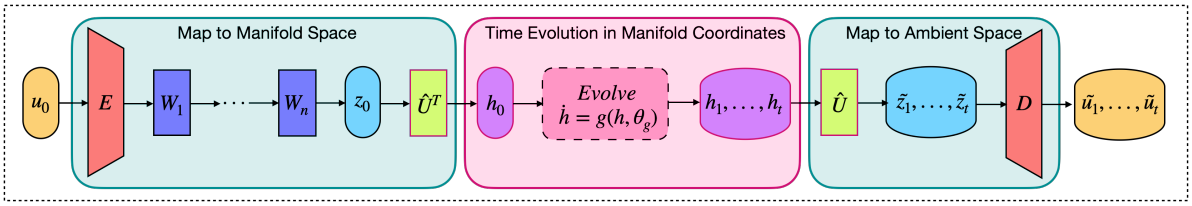

Figure 5.10: Schematic for extending the IRMAE-WD framework for forecasting in the manifold coordinate system using a Neural ODE (pink section).

where $\mathcal{E}_h$ and $\mathcal{D}_h$ simply subsume the intermediate linear transformations required to map between h and u. With our extended definitions of encoding and decoding, we now have 1) found an estimate of d_m, 2) obtained the coordinate system h parameterizing the manifold, and 3) determined the explicit mapping functions $\mathcal{E}_h$ and $\mathcal{D}_h$ back and forth between the ambient space and data manifold. With access to these three, a natural application is state-space modeling and forecasting. We show a schematic of this extension in Fig. 5.10; the pink internal box contains a time-evolution module to integrate an initial condition u_0 that has been transformed into manifold coordinate representation h_0 forward in time.

We briefly demonstrate this natural application using the KSE, where we train a neural ODE [19], $\dot{h} = g(h; \theta_g)$, to model the time evolution of h as done by Linot and Graham [123]. In other words, we simply insert a forecasting network trained to evolve the dynamics of the system in the manifold coordinate representation, h. Shown in Fig. 5.11a, is an example trajectory from the KSE. Fig. 5.11b is the $\mathcal{E}_h$ encoded manifold representation of the same trajectory. From a single encoded initial condition in the ambient space, we can perform the entire systems forecast in the manifold space. This forecasted trajectory, for the same initial condition used to generate Fig. 5.11a, is shown in Fig. 5.11d. Naturally, the ambient representation of this trajectory can be completely recovered via $\mathcal{D}_h$, shown in Fig. 5.11c. Comparison of the top and bottom rows shows that the time-evolution prediction in the manifold coordinate system is quantitatively accurate for nearly 50 time units. We emphasize that because the KSE is a chaotic dynamical system, the ground truth and forecast will eventually diverge. Nevertheless, the relevant time scale (the Lyapunov time) of this system

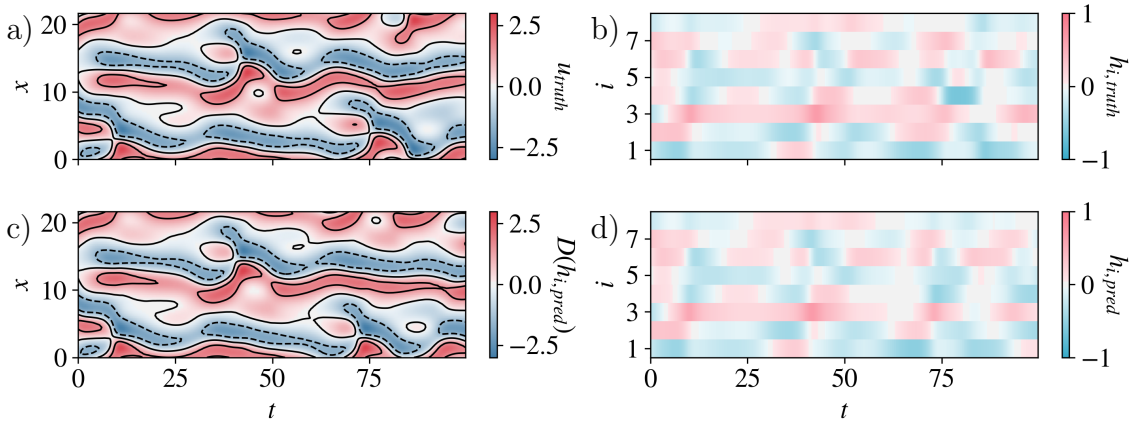

Figure 5.11: Example ground truth trajectory of the KSE in the a) ambient space and b) projected onto the learned manifold coordinate representation. A time series forecast made using a Neural ODE in the d) manifold coordinate beginning from the same initial condition. c) The ambient space reconstruction decoded from the Neural ODE forecasted manifold trajectory.

is $\sim$ 20 time units and we achieve quantitative agreement for about two Lyapunov times. From this result, we highlight that our learned manifold coordinate system is conducive for forecasting, and our mapping functions produce good ambient space reconstruction. This approach to development of data-driven reduced-order models will be further applied and assessed in future work.

5.3.5 The Dynamics of Low-Rank Representation Learning

We now turn our attention towards understanding the automatic learning of an approximately minimal representation. We glean insights by framing our network as a dynamical system, where "space" corresponds to layer depth in the network and "time" corresponds to training epoch/iteration. In this manner, we elucidate "when" and "where" low rank behavior appears in our network.

More precisely, we will compute and track the singular value spectra for a range of intermediate latent representations, weight matrices, and update gradients as a function of model layer and epoch. We will use these spectra to estimate the rank (based on the position

of a substantial gap in the singular value spectrum of the matrix under investigation). The following analyses are performed on a framework with $n = 4, \lambda = 10^{-6}$, trained on the KSE $L = 22$ dataset, which has a 64-dimensional ambient space with a nonlinear invariant manifold with $d_m = 8$.

We first define several key weights, W_j, and representations, z_j, in the model from input to output, where j is a placeholder for the position in the network. Starting from the encoder, we define the nonlinearly-activated representation immediately output from the *nonlinear* portion of the encoder, $\mathcal{E}_N$, as $z_{\mathcal{E}N} = \mathcal{E}_N(u)$. This representation is then mapped to $\mathbb{R}^{d_z}$ by a linear layer $W_\mathcal{E}$ to result in representation $z_\mathcal{E} = \mathcal{E}(u) = W_\mathcal{E}\mathcal{E}_N(u)$. From here, the representation passes through n square linear layers: $W_1, ..., W_n$. The representation output after each of these layers is then $z_1, ..., z_n$. Note that z_n is equal to z in the nomenclature of the previous sections. Finally, before arriving at the nonlinear decoder, $\mathcal{D}_N$, z_n is mapped via $W_\mathcal{D}$ to the proper size: $\mathcal{D}(z_n) = \mathcal{D}_N(W_\mathcal{D} z_n)$. To summarize, a fully encoded and decoded snapshot of data is $\tilde{u} = \mathcal{D}(\mathcal{W}(\mathcal{E}(u))) = \mathcal{D}_N(W_\mathcal{D} W_n ... W_1 W_\mathcal{E} \mathcal{E}_N(u))$.

We first perform space-time tracking of the rank of the latent representation, shown in Fig. 5.12, by computing the singular spectrum of the covariance of the data representation, z_j, at various intermediate layers of the network and various epochs during training. As we traverse our model in space (layer), we find that the nonlinear encoder produces a full-rank representation and is not directly responsible for transforming the data into its low-rank form, shown in Fig. 5.12a. However, we observe that as the data progresses from the nonlinear encoder and through the non-square linear mapping to W_1, the learned representation is weakly low-rank, shown in Fig. 5.12b. As the data progresses through each of the square linear blocks $W_1, ..., W_n$, we observe that the unnecessary singular values/directions of the representation are further attenuated (equivalently the most essential representation directions are amplified), transforming the latent representation towards a true minimal-rank representation.

As we traverse our model in time (i.e. epoch), we observe that the rank of the learned

representation for $z_{\mathcal{E}N}$ is stagnant. In contrast, we observe for each of the sequential linear layers the rank of the representation begin as essentially full-rank, but then collectively decay into low-rank representations. We note that the representation during the early epochs "over-correct" to a representation that is too low of a rank to accurately capture the data, but the network automatically resolves this as training progresses.

To further understand what is happening, we now perform a similar space-time analysis of our model to track the rank of the gradient updates of the weights at each layer, $\mathcal{J}_j = \nabla_{W_j}\mathcal{L}$, shown in Fig. 5.13. Here we follow the same layer indexing convention described above. We observe in Fig. 5.13 that in early training the sequential linear layers begin with update gradients that adjust all directions in each of the latent representations. As training progresses, the singular values of the update gradients begin to decay in unnecessary directions, shifting the latent space towards a low-rank representation. Once this is achieved, the gradient updates are essentially only updating in the significant directions needed for reconstruction. From the analysis of the gradient updates, we can conclude that the framework collectively adjusts all linear layers.

As linear layers in sequence can be subsumed into a single linear layer by simply computing the product of the sequence, we also investigate the rank of the *effective* layer weight matrix itself, $W_{j,\text{eff}}$ (e.g. $W_{2,\text{eff}} = W_2W_1W_{\mathcal{E}}$) in space and time, shown in Fig. 5.14. We show in Fig. 5.14 as the linear layers compound deeper into the network, the effective rank of the layers approaches d_m. This coincides with the observation made in Fig. 5.12. We conclude here that the sequential linear layers work together to form an effective rank d_m weight matrix, projecting the data onto a space of dimension d_m. We highlight here that while the network automatically learns a linearly separated d_m representation, the manifold of the original dataset is nonlinear in nature and is nonlinearly embedded in the ambient space–this feature is captured by the nonlinear encoding and decoding blocks. Finally, we comment that when weight-sharing is implemented across the linear blocks W_j (i.e. W_j are equal) we lose regularization as weight-sharing decreases the effective number of linear layers.

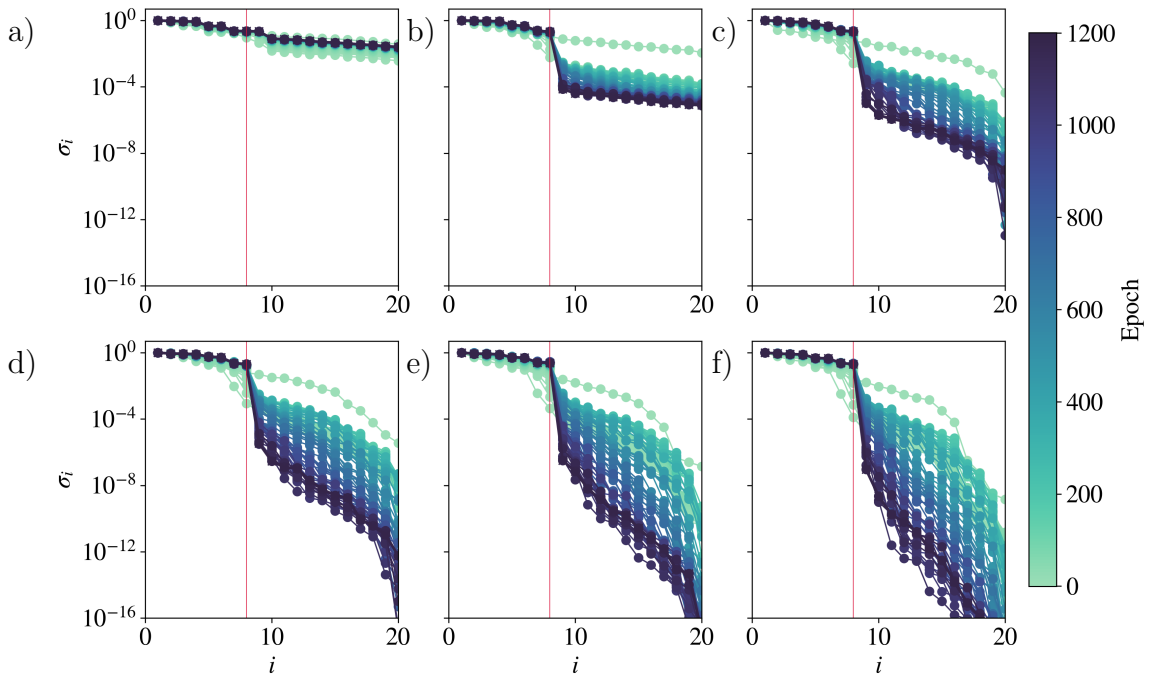

Figure 5.12: "Space-Time" tracking of the singular spectra of the covariance of the representation of the data, z_j, trained on the KSE $L = 22$ dataset: a) $z_{\mathcal{E}N}$ b) $z_{\mathcal{E}}$ c) z_1, d) z_2, e) z_3, and f) z_4 (i.e. z) as a function of training epoch. Note that the spectra for a) and b) are truncated for clarity. The d_m of the dataset is denoted by a vertical red line.

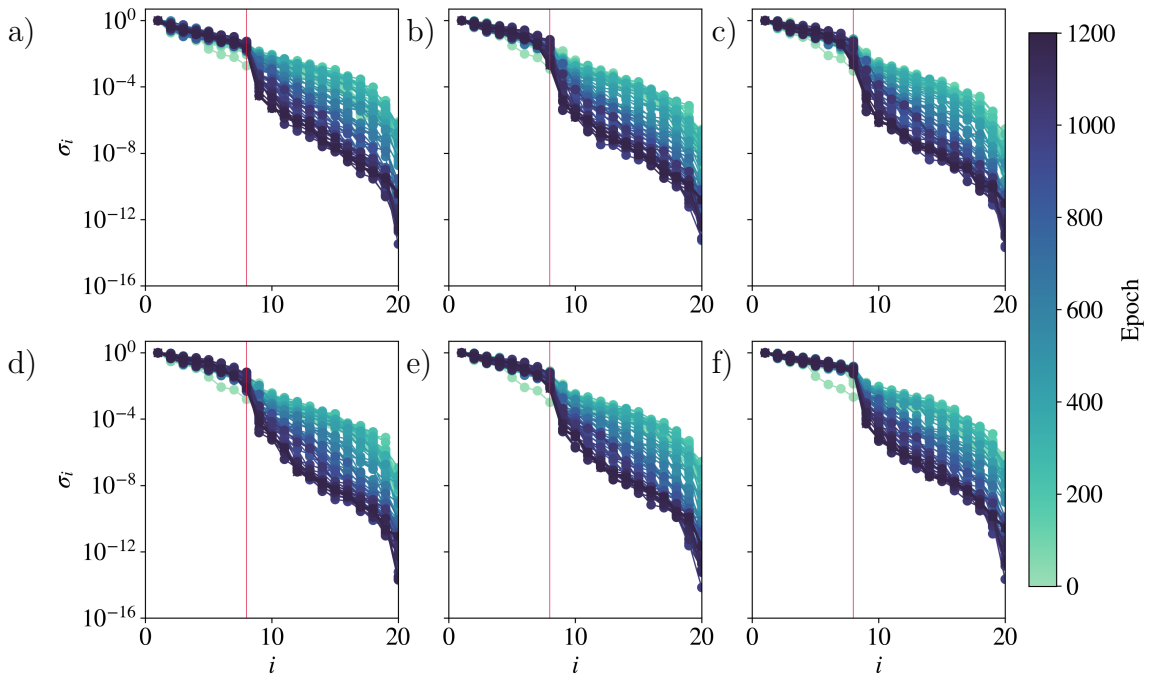

Figure 5.13: "Space-Time" tracking of the singular spectra of the update gradient, $\mathcal{J}_j$, for a model trained on the KSE $L = 22$ dataset for the a) $\mathcal{J}_\mathcal{E}$ b) $\mathcal{J}_1$, c) $\mathcal{J}_2$, d) $\mathcal{J}_3$, e) $\mathcal{J}_4$, and f) $\mathcal{J}_\mathcal{D}$ as a function of training epoch. Note that the spectra for a) and f) are truncated for clarity. The d_m of the dataset is denoted by a vertical red line.

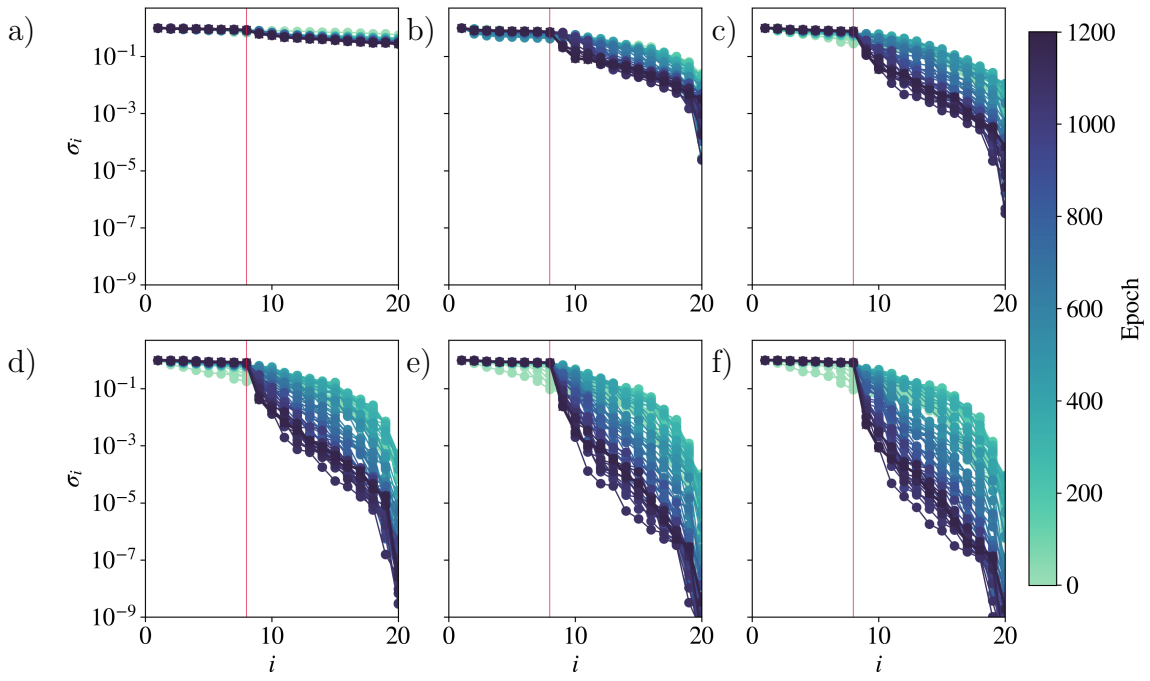

Figure 5.14: "Space-Time" tracking of the singular spectra of the effective linear layer, $W_{j,\text{eff}}$, for a model trained on the KSE $L = 22$ dataset. The singular spectra for the effective weight matrix a) $W_{\mathcal{E},\text{eff}}$ b) $W_{1,\text{eff}}$, c) $W_{2,\text{eff}}$, d) $W_{3,\text{eff}}$, e) $W_{4,\text{eff}}$, and f) $W_{\mathcal{D},\text{eff}}$ as a function of training epoch. Note that the spectra for a) and f) are truncated for clarity. The d_m of the dataset is denoted by a vertical red line.

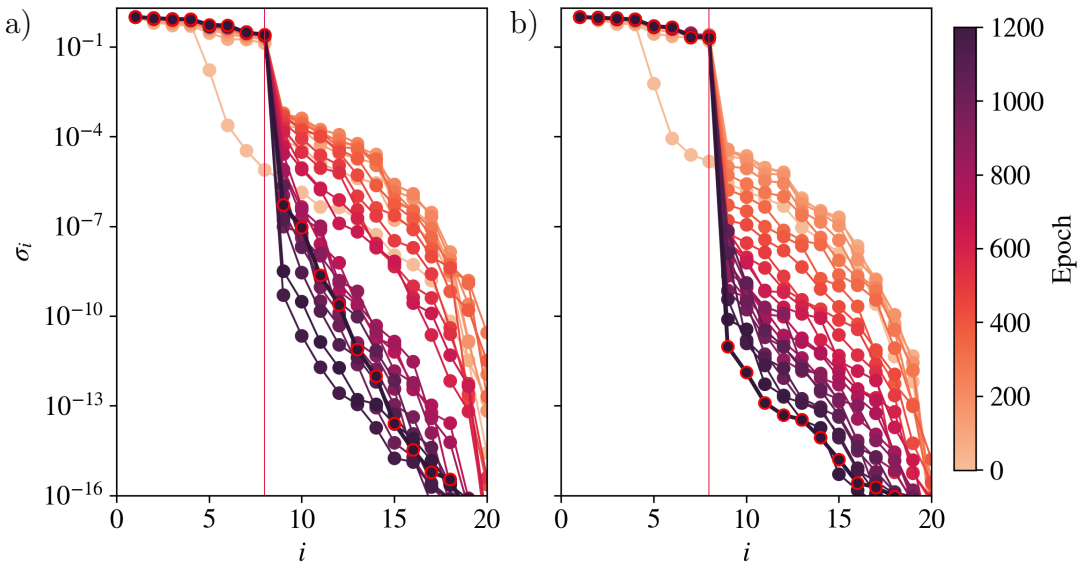

Figure 5.15: "Space-Time" tracking of the singular spectra of the covariance of the representation of the data, z, trained on the KSE $L = 22$ dataset: a) an AE with only implicit regularization (IRMAE) b) an AE with implicit regularization and weight-decay (IRMAE-WD). The d_m of the dataset is denoted by a vertical red line and the final learned latent spectrum is outlined in red markers.

We conclude this section with a comparison between our proposed framework IRMAE-WD, which utilizes implicit regularization and weight-decay, and one that only utilizes implicit regularization, IRMAE. We show in Fig. 5.15 the learning dynamics of the data covariance of the latent representation for each. Fig. 5.15a shows the dynamics in the absence of weight decay where we observe that the trailing singular values first drift upward in the first 100 epochs, followed by decay and then growth again as training proceeds. The addition of weight decay, as shown in Fig. 5.15b, leads to monotonic decay of the trailing singular values. These observations are consistent with the linear IRMAE-WD analysis in Appendix 5.7, which, in the absence of weight-decay predicts directions with eigenvalues at zero in which the training dynamics will drift. Adding weight decay makes these eigenvalues negative, aiding convergence.

5.4 Summary

In this chapter, we build upon observations made by Jing et al. [97] and present an autoencoder framework, denoted IRMAE-WD, that combines implicit regularization with internal linear layers and weight decay to automatically estimate the underlying dimensional d_m of the manifold on which the data lies. This framework simultaneously learns an ordered and orthogonal manifold coordinate representation as well as the mapping functions between the ambient space and manifold space, allowing for out-of-sampling projections. Unlike other autoencoder methods, we accomplish this without parametric model sweeps or relying on secondary algorithms, requiring only that the bottleneck dimension d_z of the autoencoder satisfies $d_z > d_m$.

We demonstrated our framework by estimating the manifold dimension for a series of finite and (discretized) infinite-dimensional systems that possess linear and nonlinear manifolds. We show that it outperforms several state-of-the-art estimators for systems with nonlinear embedded manifolds and is even accurate for relatively large manifold dimensions, $d_m \approx 40$. However, the ambient dimensions of our test systems are still small relative to the demands of many industrially relevant applications, such as turbulent fluid flows where the ambient dimension d_u (number of Fourier modes or grid points) can easily exceed 10^6 and d_m is suspected to increase very strongly with Reynolds number (flow strength). We aim with future work to efficiently extend IRMAE-WD to these high-dimensional systems.

We demonstrate that our framework can be naturally extended for applications of state-space modeling and forecasting with the Kuramoto-Sivashinsky equation. Using a neural ODE, we learned the dynamics of the dataset in the manifold representation and showed that the ambient space representation can be accurately recovered at any desired point in time.

Our analyses of the training process in "space" (layer) and "time" (epoch) indicate that low-rank learning appears simultaneously in all linear layers. We highlight that the nonlinear encoder is not directly responsible for learning a low-rank representation, but rather each

of the sequential linear layers work together by compounding the approximately low-rank features in the latent space, effectively amplifying the relevant manifold directions and equivalently attenuating superfluous modes. Analysis of a linear autoencoder with the IRMAE-WD architecture illustrates the role of the linear layers in accelerating collective convergence of the encoder, decoder, and internal layers as well as the role of weight-decay in breaking degeneracies that limit convergence in its absence. On the theoretical side, while the linear autoencoder analysis presented in Appendix 5.7 provides some insight, it is quite limited, and further, more sophisticated studies are necessary to better understand the method in the fully nonlinear setting.

Finally, we demonstrate that our framework is quite robust to choices of L_2 regularization (weight decay) parameter λ and number of linear layers n. We show that in a large envelope of regularization parameters we achieve accurate estimations of d_m without sacrificing accuracy (MSE). We also show that λ can help reduce the contribution of superfluous singular directions in the learned latent space.

While the present work is motivated by complex *deterministic* dynamical systems, we acknowledge that many practical systems of interest are stochastic or noisy and the data may only lie *near*, but not precisely on a finite-dimensional manifold and we aim to robustly extend IRMAE-WD to these systems in future work.

Acknowledgments

This work was supported by Office of Naval Research grant N00014-18-1-2865 (Vannevar Bush Faculty Fellowship).

5.5 Model Architecture and Parameters

Table 5.2: Here we list the architecture and parameters utilized in the studies of this chapter. For brevity, the decoders, $\mathcal{D}$, of each architecture is simply mirrors of the encoder, $\mathcal{E}$ with activations ReLU/ReLU/lin. Each network has n sequential linear layers with shape $d_z \times d_z$ between the encoder and decoder. Learning rates were set to 10^{-3} and with mini-batches of 128.

Dataset	$\mathcal{E}$	Activation	d_z	n	λ
5D Noise	20/128/64/20	ReLU/ReLU/lin	20	4	10^{-2}
Arch. Lorenz	4/128/64/4	ReLU/ReLU/lin	4	4	10^{-6}
2Torus	3/256/128/10	ReLU/ReLU/lin	10	4	10^{-2}
KSE $L = 22$	64/512/256/20	ReLU/ReLU/lin	20	4	10^{-6}
KSE $L = 44$	64/512/256/30	ReLU/ReLU/lin	30	4	10^{-6}
KSE $L = 66$	64/512/256/50	ReLU/ReLU/lin	50	4	10^{-6}
KSE $L = 88$	20/512/256/80	ReLU/ReLU/lin	80	4	10^{-6}

5.6 Application to the MNIST Handwriting Dataset

Here we apply IRMAE-WD to the MNIST dataset and compare to Jing et al. [97]. We utilize the same convolutional autoencoder architecture parameters that they used, with the following parameters and architecture: 4×4 kernel size with stride 2, padding 1, a learning rate of 10^{-3}, and $\lambda = 10^{-6}$. Here Conv, ConvT, and FC correspond to a convolutional layer, transposed-convolutional layer, and fully connected (not activated) layer, respectively.

In Fig. 5.16 we show that IRMAE-WD, which utilizes both implicit and weight regularization, learns a $d_m = 9$ representation for the MNIST handwriting dataset while Jing et al. [97], which only utilizes implicit regularization, learns a $d_m = 10$ representation. We further highlight that the trailing singular values from our model sharply decays several orders of magnitude lower than the Jing et al. [97] model. We finally note that the latent space from the Jing et al. [97] model exhibits a broader tail, especially near the significant singular values. We find that despite our model utilizing one fewer degree of freedom to model the MNIST data, it produces an MSE that is comparable to Jing et al. [97] when trained using their parameters ($1.0 \cdot 10^{-2}$ vs. $9.5 \cdot 10^{-3}$).

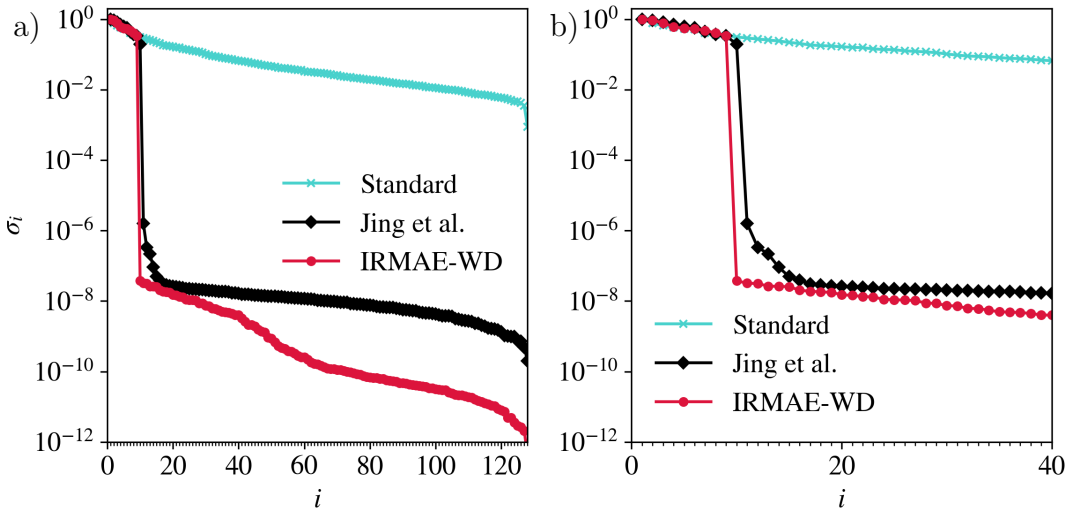

Figure 5.16: Singular value spectra obtained from models trained over the MNIST handwriting dataset a) full spectra and b) zoomed in spectra.

Table 5.3: Convolutional autoencoder and architecture for MNIST Handwriting Dataset

Encoder	Decoder
$x \in \mathbb{R}^{32\times32\times1}$	$z \in \mathbb{R}^{128}$
$\rightarrow \text{Conv}_{32} \rightarrow \text{ReLU}$	$\rightarrow \text{FC}_{4096}$
$\rightarrow \text{Conv}_{64} \rightarrow \text{ReLU}$	$\rightarrow \text{reshape}_{8\times8\times64}$
$\rightarrow \text{Conv}_{128} \rightarrow \text{ReLU}$	$\rightarrow \text{ConvT}_{64} \rightarrow \text{ReLU}$
$\rightarrow \text{Conv}_{256} \rightarrow \text{ReLU}$	$\rightarrow \text{ConvT}_{32} \rightarrow \text{ReLU}$
$\rightarrow \text{flatten}_{1024}$	$\rightarrow \text{ConvT}_{1} \rightarrow \text{Tanh}$
$\rightarrow \text{LC}_{128} \rightarrow z \in \mathbb{R}^{128}$	$\rightarrow \hat{x} \in \mathbb{R}^{32\times32\times1}$

5.7 Analysis of linear autoencoders with internal linear layers and weight decay

5.7.1 Formulation

To gain some insight into the performance of autoencoders with additional linear layers and weight decay, we present here an analysis of gradient descent for an idealized case of a linear autoencoder with one or more internal linear layers. We begin with the formalism with a single internal linear layer. The input is denoted $u \in \mathbb{R}^{d_u}$, encoder $E \in \mathbb{R}^{d_z \times d_u}$, decoder

$D \in \mathbb{R}^{d_u \times d_z}$, internal linear layer $W \in \mathbb{R}^{d_z \times d_z}$ and output $\tilde{u} = DWEu \in \mathbb{R}^{d_u}$. We define the latent variable preceding the linear layer as $h = Eu \in \mathbb{R}^{d_z}$ and the one following it as $w = Wh = WEu \in \mathbb{R}^{d_z}$. For a conventional autoencoder, $W = I^{d_z d_z}$, where the notation I^{mm} denotes the $m \times m$ identity matrix. We will consider the simple loss function

$$\mathcal{L} = \langle ||\tilde{u} - u||_2^2 \rangle + \lambda_E(||E||_F^2 + ||D||_F^2) + \lambda_W ||W||_F^2,$$

where $\langle \cdot \rangle$ denotes ensemble average (expected value). First the converged equilibrium solution of the minimization problem for the loss will be considered, and then the convergence of the solution to the minimum.

We are particularly interested in the case where the data lies on an r-dimensional subspace of $\mathbb{R}^{d_u}$, or equivalently $\text{rank}\langle uu^T \rangle = r$, and we assume that the dimension m of the hidden layers is chosen so that $m > r$.

We can write the loss as

$$\mathcal{L} = \langle u^T(E^T W^T D^T DWE - (DWE + E^T W^T D^T))u \rangle + \langle u^T u \rangle + \lambda_E(\text{tr}EE^T + \text{tr}DD^T) + \lambda_W \text{tr}WW^T.$$

In index notation we can write

$$\begin{aligned}\mathcal{L} = &\langle u_k E_{kl}^T W_{lm}^T D_{mn}^T D_{no} W_{op} E_{pq} u_q \rangle \\ &- \langle u_k (D_{kl} W_{lm} E_{mn} + E_{kl}^T W_{lm}^T D_{mn}^T) u_n \rangle \\ &+ \langle u_k u_k \rangle + \lambda_E(E_{kl}E_{kl} + D_{kl}D_{kl}) + \lambda_W W_{kl} W_{kl}.\end{aligned}$$

Taking partial derivatives yields

$$\frac{\partial \mathcal{L}}{\partial E_{ij}} = \langle 2W^T_{im} D^T_{mn} (DWE - I)_{no} u_o u_j \rangle + 2\lambda_E E_{ij} = \langle 2W^T_{im} D^T_{mn} (\tilde{u}_n - u_n) u_j \rangle + 2\lambda_E E_{ij},$$
$$\frac{\partial \mathcal{L}}{\partial W_{ij}} = \langle 2D^T_{ik} (DWE - I)_{kl} u_l E_{jm} u_m \rangle + 2\lambda W_{ij} = \langle 2D^T_{ik} (\tilde{u}_k - u_k) h_j \rangle + 2\lambda_E W_{ij},$$
$$\frac{\partial \mathcal{L}}{\partial D_{ij}} = \langle 2(DWE - I)_{ik} u_k (WE)_{jl} u_l \rangle + 2\lambda_E D_{ij} = 2(\tilde{u}_i - u_i) w_j + 2\lambda_E D_{ij}.$$

Without loss of generality, we can work in coordinates where $\langle uu^T \rangle = \sigma^2 I^{rd_u d_u}$, where I^{rpq} is an $p \times q$ matrix (with $p, q > r$) whose first r diagonal elements are unity and all others are zero. Now

$$\frac{\partial \mathcal{L}}{\partial E_{ij}} = 2\sigma^2 W^T_{im} D^T_{mn} (DWE - I)_{no} I^{rd_u d_u}_{oj} + 2\lambda_E E_{ij},$$
$$\frac{\partial \mathcal{L}}{\partial W_{ij}} = 2\sigma^2 D^T_{ik} (DWE - I)_{kl} E_{jm} I^{rd_u d_u}_{lm} + 2\lambda_W W_{ij},$$
$$\frac{\partial \mathcal{L}}{\partial D_{ij}} = 2\sigma^2 (DWE - I)_{ik} (WE)_{jl} I^{rd_u d_u}_{kl} + 2\lambda_E D_{ij}.$$

In matrix-vector notation this becomes

$$\frac{\partial \mathcal{L}}{\partial E} = 2\sigma^2 W^T D^T (DWE - I) I^{rd_u d_u} + 2\lambda_E E,$$
$$\frac{\partial \mathcal{L}}{\partial W} = 2\sigma^2 D^T (DWE - I) I^{rd_u d_u} E^T + 2\lambda_W W,$$
$$\frac{\partial \mathcal{L}}{\partial D} = 2\sigma^2 (DWE - I) I^{rd_u d_u} (WE)^T + 2\lambda_E D.$$

5.7.2 Equilibrium solutions

At convergence, these derivatives vanish. For the moment, we set $\lambda_E = 0$. We first consider the solution in absence of the internal linear layer: i.e. when $W = I^{d_z d_z}$. Now $\frac{\partial \mathcal{L}}{\partial E}$ and $\frac{\partial \mathcal{L}}{\partial D}$ will vanish when

$$(DE - I) I^{rd_u d_u} = DEI^{rd_u d_u} - I^{rd_u d_u} = 0.$$

This has "full rank" solution $E = D^T = I^{d_z d_u}$, which satisfies $DE - I = 0$, as well as "rank r" solution $E = D^T = I^{r d_z d_u}$. This does not satisfy $DE - I = 0$, but does satisfy $DEI^{r d_u d_u} - I^{r d_u d_u} = 0$. If we include a nontrivial linear layer W, we than have

$$(DWE - I)I^{r d_u d_u} = DWEI^{r d_u d_u} - I^{r d_u d_u} = 0.$$

it is clear that the rank r solution $E = D^T = I^{r d_u d_z}$, along with the rank r choice $W = I^{r d_z d_z}$ continues to be a solution, as does the full rank solution with $E = D^T = I^{d_z d_u}$ with $W = I^{d_z d_z}$.

In the presence of weight decay the situation is more complex, and we will only consider the case $\lambda_E = \lambda_W = \lambda$. Defining a new parameter $\zeta = \lambda/\sigma^2$, and taking this parameter to be small, a perturbation solution of the form

$$E = I^{r d_z d_u}(1 + \alpha\zeta + O(\zeta^2)), W = I^{r d_z d_z}(1 + \beta\zeta + O(\zeta^2)), D = I^{r d_u d_z}(1 + \gamma\zeta + O(\zeta^2)) \quad (5.7)$$

can be found. Plugging into the equilibrium conditions $\frac{\partial \mathcal{L}}{\partial E} = 0, \frac{\partial \mathcal{L}}{\partial W} = 0, \frac{\partial \mathcal{L}}{\partial D} = 0$ and neglecting terms of $O(\zeta^2)$ yields in each case

$$\alpha + \beta + \gamma + 1 = 0.$$

Thus there is a whole family of solutions to the equilibrium problem with weight decay. For future reference we will write this solution (up to $O(\zeta)$) as

$$\begin{aligned} E &= aI^{r d_z d_u}, \quad a = 1 + \alpha\zeta, \\ W &= bI^{r d_z d_z}, \quad b = 1 + \beta\zeta, \\ D &= cI^{r d_u d_z}, \quad c = 1 + \gamma\zeta. \end{aligned} \quad (5.8)$$

5.7.3 Convergence of gradient descent

Dynamic model

Now we turn to the issue of convergence of gradient descent to an equilibrium solution. Here we consider a very simple ordinary differential equation model of the process, where t is a pseudotime representing number of gradient descent steps:

$$\begin{aligned}
\frac{dE}{dt} &= -\frac{1}{2\sigma^2}\frac{\partial \mathcal{L}}{\partial E} = -W^T D^T (DWE - I) I^{rd_u d_u} - \zeta E, \\
\frac{dW}{dt} &= -\frac{1}{2\sigma^2}\frac{\partial \mathcal{L}}{\partial W} = -D^T (DWE - I) I^{rd_u d_u} E^T - \zeta W, \\
\frac{dD}{dt} &= -\frac{1}{2\sigma^2}\frac{\partial \mathcal{L}}{\partial D} = -(DWE - I) I^{rd_u d_u} (WE)^T - \zeta D.
\end{aligned} \tag{5.9}$$

This is a high-dimensional and highly nonlinear system; to make progress we consider only the dynamics near convergence, linearizing the system around the converged solution (5.7). That is, we set

$$\begin{aligned}
E &= aI^{rd_z d_u} + \epsilon \hat{E}, \\
W &= bI^{rd_z d_z} + \epsilon \hat{W}, \\
D &= cI^{rd_u d_z} + \epsilon \hat{D},
\end{aligned} \tag{5.10}$$

insert these expressions into (5.9), and neglect terms of $O(\epsilon^2)$, to yield

$$\begin{aligned}
\frac{d\hat{E}}{dt} &= -bc\left[abI^{rd_z d_u}\hat{D}I^{rd_z d_u} + acI^{rd_z d_z}\hat{W}I^{rd_z d_u} + bcI^{rd_u d_z}\hat{E}I^{rd_z d_z}\right] - \zeta\hat{E}, \\
\frac{d\hat{W}}{dt} &= -ab\left[abI^{rd_z d_u}\hat{D}I^{rd_z d_z} + acI^{rd_z d_z}\hat{W}I^{rd_z d_z} + bcI^{rd_z d_z}\hat{E}I^{rd_u d_z}\right] - \zeta\hat{W}, \\
\frac{d\hat{D}}{dt} &= -ac\left[abI^{rd_u d_z}\hat{D}I^{rd_z d_z} + acI^{rd_u d_z}\hat{W}I^{rd_z d_z} + bcI^{rd_u d_z}\hat{E}I^{rd_u d_z}\right] - \zeta\hat{D}.
\end{aligned} \tag{5.11}$$

We can now make some important general statements about the solutions. First, observe that the terms in the square brackets will always yield matrices for which only the upper left $r \times r$ block is nonzero. Furthermore for any nonzero ζ, all terms outside this block will be driven to zero. Finally, observe that (5.11) will have time-dependent solutions of the form $\hat{E}(t) = \mathcal{E}(t)I^{rd_z d_u}, \hat{W}(t) = \mathcal{W}(t)I^{rd_z d_z}, \hat{D}(t) = \mathcal{D}(t)I^{rd_u d_z}$, where $\mathcal{E}, \mathcal{D}$, and $\mathcal{W}$ are *scalar*

functions of time. The evolution equation for these quantities is

$$\begin{aligned}\frac{d\mathcal{E}}{dt} &= -bc\left[ab\mathcal{D} + ac\mathcal{W} + bc\mathcal{E}\right] - \zeta\mathcal{E},\\ \frac{d\hat{W}}{dt} &= -ab\left[ab\mathcal{D} + ac\mathcal{W} + bc\mathcal{E}\right] - \zeta\mathcal{W},\\ \frac{d\hat{D}}{dt} &= -ac\left[ab\mathcal{D} + ac\mathcal{W} + bc\mathcal{E}\right] - \zeta\mathcal{D}.\end{aligned} \tag{5.12}$$

Hereinafter, we will consider solutions in this invariant subspace, where a fairly complete characterization of the linearized dynamics is possible.

Linear layers speed collective convergence of weights

The situation is simplest when there is no weight decay: $\zeta = 0$. Now $a = b = c = 1$ and (5.12) simplifies to

$$\begin{aligned}\frac{d\mathcal{E}}{dt} &= -\left[\mathcal{D} + \mathcal{W} + \mathcal{E}\right],\\ \frac{d\mathcal{W}}{dt} &= -\left[\mathcal{D} + \mathcal{W} + \mathcal{E}\right],\\ \frac{d\mathcal{D}}{dt} &= -\left[\mathcal{D} + \mathcal{W} + \mathcal{E}\right].\end{aligned} \tag{5.13}$$

Adding these equations together yields that

$$\frac{d}{dt}(\mathcal{D} + \mathcal{W} + \mathcal{E}) = -3(\mathcal{D} + \mathcal{W} + \mathcal{E}).$$

So the “collective" weight $\mathcal{C} = \mathcal{D} + \mathcal{W} + \mathcal{E}$ decays as $e^{-\rho_1 t}$ where $\rho_1 = 3$. More generally ,we can write (5.13) in matrix-vector form

$$\frac{d}{dt}\begin{bmatrix}\mathcal{E}\\ \mathcal{W}\\ \mathcal{D}\end{bmatrix} = A\begin{bmatrix}\mathcal{E}\\ \mathcal{W}\\ \mathcal{D}\end{bmatrix}, \qquad A = -\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}. \tag{5.14}$$

This has general solution

$$\begin{bmatrix} \mathcal{E}(t) \\ \mathcal{W}(t) \\ \mathcal{D}(t) \end{bmatrix} = C_1 e^{-3t} v_1 + C_2 v_2 + C_3 v_3$$

with

$$v_1 = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \qquad v_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \qquad v_3 = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix},$$

and $C_i = [\mathcal{E}(0), \mathcal{W}(0), \mathcal{D}(0)]^T v_i$. Therefore, while the collective weight $\mathcal{D} + \mathcal{W} + \mathcal{E}$ decays as e^{-3t}, the quantities $\mathcal{D} - \mathcal{E}$ and $\mathcal{W} - \mathcal{E}$ do not decay at all, because of the two zero eigenvalues of the matrix G. This fact will limit the performance of gradient descent in the absence of weight decay. (We see below that weight decay breaks the degeneracy of the dynamics.)

Now we proceed to the question of how the number of internal linear layers affects convergence. To consider the case of no internal linear layers, we simply set $\hat{W}$ and thus $\mathcal{W}$ to zero — the matrix W is simply fixed at the identity. Now (5.14) reduces to

$$\frac{d}{dt} \begin{bmatrix} \mathcal{E} \\ \mathcal{D} \end{bmatrix} = - \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \mathcal{E} \\ \mathcal{D} \end{bmatrix}. \tag{5.15}$$

Now the collective weight variable $\mathcal{D} + \mathcal{E}$ decays as e^{-2t}, rather than e^{-3t} when we had an internal linear layer – this added layer accelerates convergence.

What if we add additional linear layers, for a total of n, by replacing W with a product $W_n W_{n-1} W_{n-2} \cdots W_1$? Without loss of generality we can take the converged value of each of these matrices (in the absence of weight decay) to be $I^{r d_z d_z}$. In considering the linearized

dynamics we use the result

$$\begin{aligned} W_n W_{n-1} W_{n-2} \cdots W_1 &= (I^{rd_z d_z} + \epsilon \hat{W}_n)(I^{rd_z d_z} + \epsilon \hat{W}_{n-1})(I^{rd_z d_z} + \epsilon \hat{W}_{n-2}) \cdots (I^{rd_z d_z} + \epsilon \hat{W}_1) \\ &= I^{rd_z d_z} + \epsilon(\hat{W}_n + \hat{W}_{n-1} + \hat{W}_{n-2} + \cdots + \hat{W}_1) + O(\epsilon^2). \end{aligned}$$

Taking $\hat{W}_i = \mathcal{W}_i I^{rd_z d_z}$ and following the same process as above yields the following set of equations for the linearized dynamics:

$$\frac{d}{dt} \begin{bmatrix} \mathcal{E} \\ \mathcal{W}_n \\ \mathcal{W}_{n-1} \\ \vdots \\ \mathcal{W}_1 \\ \mathcal{D} \end{bmatrix} = - \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ 1 & 1 & 1 & \cdots & 1 & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & 1 & 1 & \cdots & 1 & 1 \end{bmatrix} \begin{bmatrix} \mathcal{E} \\ \mathcal{W}_n \\ \mathcal{W}_{n-1} \\ \vdots \\ \mathcal{W}_1 \\ \mathcal{D} \end{bmatrix} \tag{5.16}$$

By adding these equations together we find that the collective weight for this case $\mathcal{C} = \mathcal{D} + \sum_{i=1}^{n} \mathcal{W}_i + \mathcal{E}$ decays as $e^{-\rho_n t}$, with the decay rate ρ_n for an n layer network given by

$$\rho_n = 2 + n. \tag{5.17}$$

The origin of this increase in convergence rate for the collective weight variable $\mathcal{C}$ is the basic autoencoder loss structure – for every layer, the combination $DWE - I$ appears, so the gradients for all layers will have a common structure containing the collective weight $\mathcal{C}$. The more internal linear layers, the faster this collective weight converges.

Now there are $n + 1$ zero eigenvalues indicating directions where gradient descent does not act: $\mathcal{D} - \mathcal{E}$ and $\mathcal{W}_i - \mathcal{E}, i = 1, \ldots n$. Thus, while addition of internal linear layer speeds convergence of the collective weight, these degenerate directions remain, limiting the overall performance of the gradient descent process.

Weight decay breaks degeneracy and leads to asymptotic stability

Addition of weight decay complicates the analysis considerably, as illustrated in the results for the equilibrium solutions presented above. Therefore we will limit ourselves to a perturbative treatment of the dynamics of the case $n = 1$ when ζ is small. Inserting the expressions for a, b and c into (5.12) and collecting like powers of ζ leads to the equation

$$\frac{d}{dt}\begin{bmatrix}\mathcal{E}\\ \mathcal{W}\\ \mathcal{D}\end{bmatrix} = (A + \zeta B)\begin{bmatrix}\mathcal{E}\\ \mathcal{W}\\ \mathcal{D}\end{bmatrix}, \tag{5.18}$$

where A is as in (5.12) and

$$B = -\begin{bmatrix}2(\beta+\gamma)+1 & \gamma-1 & \beta-1\\ \gamma-1 & 2(\alpha+\gamma)+1 & \alpha-1\\ \beta-1 & \alpha-1 & 2(\alpha+\beta)+1\end{bmatrix}.$$

Seeking solutions of the form $ve^{\xi t}$ leads to the eigenvalue problem

$$(A + \zeta B)v = \xi v.$$

This can be solved perturbatively for small ζ [81]. Expressing eigenvectors $v = v^{(0)} + v^{(1)} + O(\zeta^2)$ and eigenvalues $\xi = \xi^{(0)} + \zeta\xi^{(1)} + O(\zeta^2)$ leads to the leading order problem

$$Ax^{(0)} = \xi^{(0)}x^{(0)} \tag{5.19}$$

and the $O(\zeta)$ problem

$$(A - \xi^{(0)}I)v^{(1)} = (B - \xi^{(1)}I)v^{(0)}. \tag{5.20}$$

The leading order problem (5.19) is precisely the no-weight-decay case described above, with eigenvalues $\xi_1^{(0)} = -\rho_1 = -3, \xi_2^{(0)} = 0, \xi_3^{(0)} = 0$ and eigenvectors

$$v_1^{(0)} = \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \qquad v_2^{(0)} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix}, \qquad v_3^{(0)} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 \\ 1 \\ 0 \end{bmatrix},$$

The $O(\zeta)$ problem is an inhomogeneous linear system with a singular left-hand side. For a given eigenvalue-eigenvector pair $\xi_i^{(0)}, v_i^{(0)}$, this will only have solutions if the right-hand side lies in the range of $(A - \xi_i^{(0)} I)$, or equivalently is orthogonal to the nullspace of $(A - \xi_i^{(0)} I)^T$. Since $(A - \xi_i^{(0)} I)$ is symmetric, for eigenvalue $\xi^{(0)} = \xi_i^{(0)}$, the nullspace of $(A - \xi_i^{(0)} I)$ is spanned by $v_i^{(0)}$, and solutions exist if $\left(v_i^{(0)}\right)^T (B - \xi^{(1)} I) v_i^{(0)} = 0$. The $O(\zeta)$ correction $\xi_i^{(1)}$ to the ith eigenvalue is determined by solving this equation:

$$\xi_i^{(1)} = \frac{\left(v_i^{(0)}\right)^T B v_i^{(0)}}{\left(v_i^{(0)}\right)^T v_i^{(0)}}. \tag{5.21}$$

Evaluating this yields $\xi_1^{(1)} = 3, \xi_2^{(1)} = \xi_3^{(1)} = -1$ (for any choice of α, β, γ that satisfies $\alpha + \beta + \gamma + 1 = 0$), so with an error of $O(\zeta^2)$ we have

$$\xi_1 = -3 + 3\zeta, \quad \xi_2 = \xi_3 = -\zeta. \tag{5.22}$$

Addition of weight decay has a very small detrimental effect on the collective convergence rate $-\xi_1$, but more importantly converts the eigenvalues at zero to negative eigenvalues, leading to decay toward the equilibrium in all directions – the equilibrium solution becomes asymptotically stable.

6

Conclusions

6.1 General summary

In this thesis, we developed deep reinforcement learning and learning representation frameworks for the application of control and reduced-order modeling of dissipative PDEs with complex dynamics such turbulent flows. Motivated by the open challenges in automated and generalizable frameworks for the control and reduced-order modeling of these systems, we leveraged our understandings of dynamical systems, deep reinforcement learning, and learning representations to develop two RL-based control frameworks, *symmetry-reduced RL* and *DManD-RL*, as well as one manifold learning framework, *IRMAE-WD*.

In Chapter 2 we developed the first control framework, symmetry-reduced RL, which we demonstrated on the KSE system. Although many systems of interest possess natural symmetries, we show that standard RL algorithms do not learn equivariant policies. The framework we developed not only guarantees equivariant behavior, but also significantly improves data efficiency and performance. The core idea in this framework is reducing the redundant symmetries from the representations the agent learns and operates in or equivalently the learning problem is moved to a symmetry-reduced subspace.

In Chapter 3 we developed the second control framework, DManD-RL, which we again demonstrate on the KSE system. The DManD-RL framework is a model-based RL frame-

work that utilizes surrogate data-driven manifold models to efficiently train RL agents. In this framework, we isolate the underlying dynamics of our target system and its dependencies to control input from entirely off-policy generated data. We demonstrate that an agent trained on these low-dimensional isolated dynamics via our DManD model leads to excellent performance in the true system despite the agent never having directly interacted with it.

In Chapter 4 we extended DManD-RL to our white-whale system–turbulent plane Couette flow with slot jets located on a single wall. We again demonstrate that an agent, trained by interacting only with a low-dimensional DManD model, performs just as well as performing RL directly on the main system. In this chapter, we highlight the accelerated training speed gains we achieved $\sim 400\times$. Additionally, we dramatically out-perform opposition control in terms of drag reduction despite having significantly limited control authority.

In Chapter 5 we pivoted our attention from control to data representation, motivated by one of the less rigid steps in DManD modeling and reduced-order modeling in general–identifying and isolating the manifold in an automated fashion from just data. In this chapter we introduce our regularized autoencoder framework which automatically 1) estimates the underlying dimensionality of the data 2) provides an orthogonal coordinate system for the manifold, 3) learns the mapping functions between the ambient representation and the manifold representation. In this chapter we demonstrate the viability of this framework for a zoo of systems, its advantages over other estimators, and we shed light on the origins low-rank representations in a deep learning context.

In the following sections, we deliver several future directions of research. While there are many natural directions for control and control applications (i.e. different systems etc.), we choose to focus on addressing the challenge of large domains. The remainder of future directions then focus upon designing latent spaces that leverage the graphical nature of dynamical systems data. These directions focus upon developing less agnostic latent or manifold representations and incorporating more task-relevant properties.

6.2 Future work

In this section we outline several potential extensions of the work described in this thesis. These ideas and preliminary results are motivated by remaining challenges in the field, recent advances in adjacent fields, or are just *interesting*.

6.2.1 Distributed Hierarchical Controllers

Distributed and templated controllers

In many industrially relevant settings, such as the surface of an airplane wing or hull of a tanker ship, it is infeasible to train a single global RL agent to control the countless number of sensor arrays and actuators needed. One natural exploitation of turbulent flows (and many other flows) is that the structures/dynamics at the small/local scale are similar regardless of global location (assuming the same flow conditions). One natural extension of the work in this thesis is that a policy developed in a minimal working unit can be replicated or templated in space to form a distributed system, where each spatial patch is controlled by the *same* policy. One could imagine training an array of locally distributed agents (that share the same policy) in the large domain, but often times even this is not feasible.

Here we demonstrate in Fig. 6.1 that our KSE symmetry-reduced agent trained in a small domain ($L = 22$) when extended to control a larger KSE system ($L = 66$), is capable of controlling the flow and reducing the dissipation/power-input cost. This is quite promising as these collective clone agents have never interacted with the larger KSE domain or with each other but still manages to achieve an acceptable control performance.

It is important to discuss several limitations of this extension. While it is the most resource efficient, the lack of direct interaction with the target system means that the transferred policy will not know of possible exploitations that could have been found via direct training. In otherwords, because the policy is replicated, the agents have no "experience" working together and thus are likely to perform sub-optimally as each only focuses on ac-

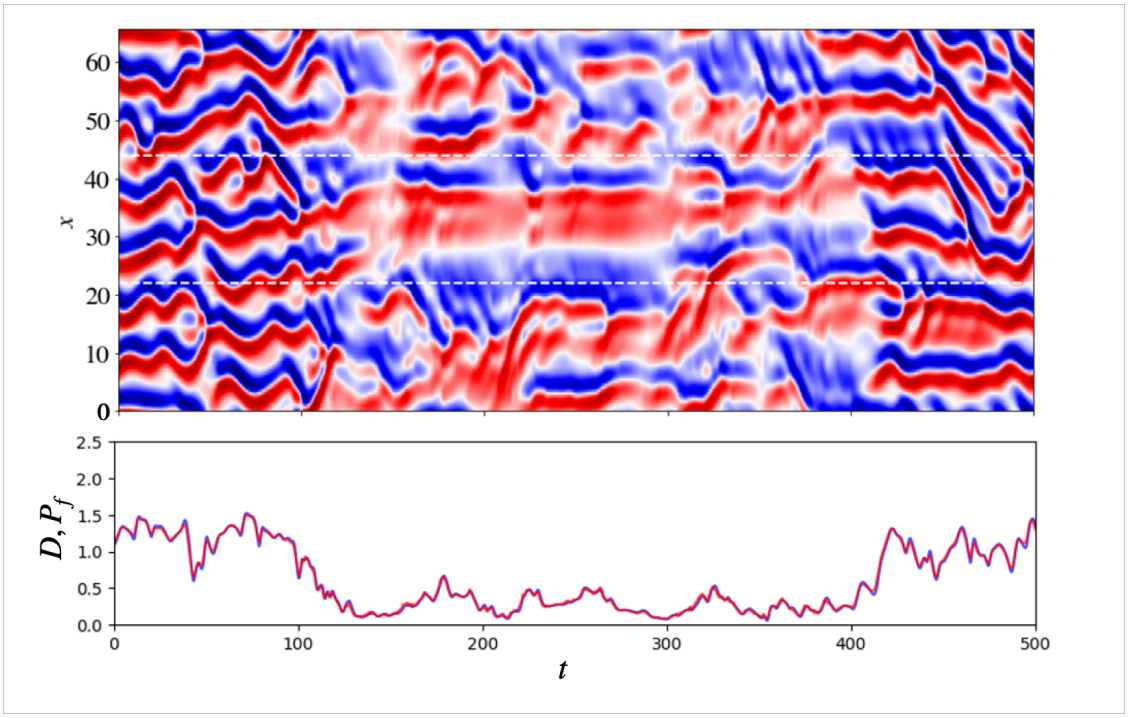

Figure 6.1: Symmetry-reduced KSE $L = 22$ agent applied to a KSE domain of $L = 66$ in three $L = 22$ subdomains marked by the dashed line. Control is implemented at $t = 100$ to $t = 400$. The $L = 66$ domain dissipation and power input cost is shown in blue and red, respectively.

complishing their task locally. This can be addressed by fine-tune training the collective together in the large domain. Finally, our symmetry groups in the small domain are subject to breakdown when extrapolating to patches in larger domains. Despite these challenges, the preliminary results are still promising!

6.2.2 Dynamically Regularized Latent Spaces

Thus far we have generally assumed that it is advantageous (or at least conservatively not disadvantageous) to perform downstream tasks with the reduced-order latent/manifold representation produced by our autoencoder networks. There is empirical evidence that this is advantageous compared to using the high-dimensionl ambient representation [123], but it is not obvious what properties, if any, our latent representation should possess to be conducive for specifically the downstream tasks of forecasting and control. Below is a discussion of some

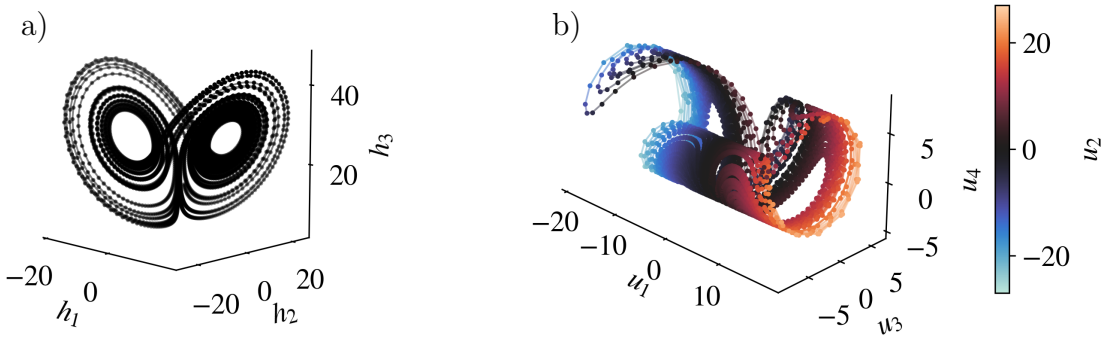

Figure 6.2: Reproduced from Chapter 5: a) Lorenz systems b) the $\mathbb{R}^4$ Archimedean Spiral Lorenz governed by the Lorenz system.

ideas and preliminary frameworks toward latent spaces for dynamical systems that may have some intuitive advantages. Although all of these preliminary results were produced from a standard regularized AE, they can all be extended to the AE framework we developed in this thesis.

Arc-length regularized latent space

Our systems of interest often exhibit a variety of local dynamics. There are regions that are slow and quiescent and others that are fast and explosive. For example, our system can be quiescent for many time units–shadowing a slow periodic orbit before making a quick leap to another region of state space. An example of this behavior can be observed in the Lorenz (butterfly) system, shown in Fig. 6.2a. The relative "speed" (i.e. absolute euclidean distance over time) the system is "moving" in the ambient representation is different depending on if it is in a slow or fast region. For numerical solvers, this problem appears in stiff problems, where we need to be cautious of our spatial and temporal resolution in order to accurately resolve fast dynamics. Analogously, for data-driven state-space/reduced-order modeling, these same issues can equally affect downstream forecasting models.

Generally speaking, the majority of encoding functions are agnostic to disparities in the magnitude of the vector fields that arise for dynamics in the latent space. Consequently, a

latent space that is learned only for the purpose of being information rich has no guarantee its manifold is suitable for data-driven forecasting.

One intuitive property that may be advantageous for the latent space is to push the underlying representation to be shaped in such a way that the magnitude of the vector field of the latent dynamics is approximately uniform. This can be accomplished simply with an additional regularization loss, generically described for an autoencoder as:

$$\mathcal{L}(u;\theta_E,\theta_D) = \|u - \mathcal{D}(\mathcal{E}(u;\theta_E);\theta_D)\|^2 + \mathcal{L}_{arc} \tag{6.1}$$

Practically, we can accomplish this by taking advantage of several properties of having deterministic time-series data. Because our systems are deterministic, our consideration for neighbors/graphs are simple as they only lie on a line. We only need to consider the data points preceding and following our data point in a time-series to approximate its speed. For data evenly spaced/sampled in time, we can reduce the problem down to simply regularizing the latent *arc-length* between consecutive time-series points to be uniform. For example, the arc-length of the dynamics at each encoded latent representation, z_t, can be estimated using just two neighbors in *time* and computing the arc-length of the circumcircle the three points lie on in $\mathbb{R}^{d_z}$. To compute the arc-length, $L_{arc} = \phi_{arc} R_{arc}$, we need to compute ϕ_{arc} and R_{arc}, the central angle and radius of the circumcircle, respectively,

$$R_{arc} = \frac{\|z_{t+1} - z_{t-1}\|}{2\sin(\phi_z)}, \quad \phi_z = \arccos\left(\frac{(z_{t+1} - z_t)\cdot(z_t - z_{t-1})}{\|z_{t+1} - z_t\|\|z_t - z_{t-1}\|}\right). \tag{6.2}$$

Here we make use of the inverse of the mengar curvature relationship to compute R_{arc}. Furthermore, we note that the relationship between the central angle and an inscribed angle is related by a factor of $1/2$, allowing us to compute $\phi_{arc} = 2\pi - 2\phi_z$. An overview of this idea is presented in Fig. 6.3.

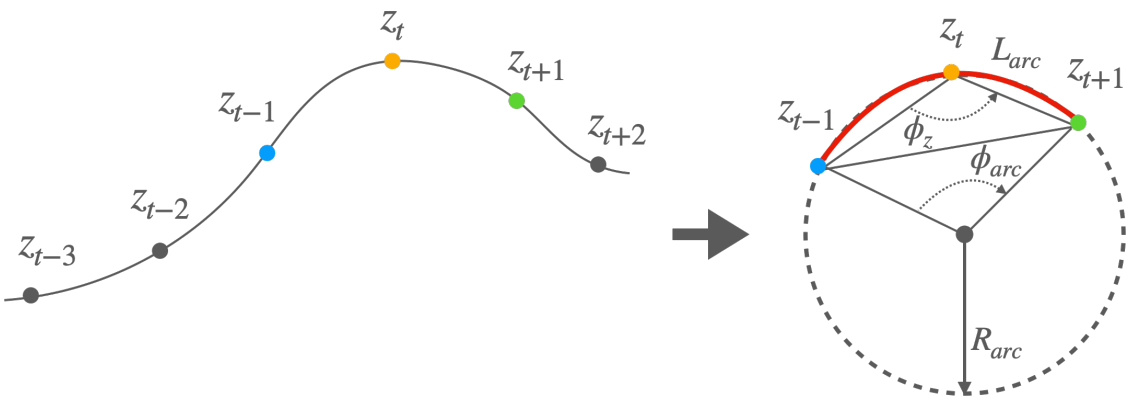

Figure 6.3: Schematic for estimating the distance traversed by a trajectory in $\mathbb{R}^{d_z}$ using an arc-length approximation.

Thus, we can devise the following arc-length penalty,

$$\mathcal{L}(x_{t-1}, x_t, x_{t+1}; \theta) = \|x_t - \mathcal{D}(\mathcal{E}(x_t; \theta_E); \theta_D)\|^2 + \beta_{arc} \left[L_0 - L_{arc}(z_{t-1}, z_t, z_{t+1})\right]^2, \tag{6.3}$$

where L_0 is a target arc-length and β_{arc} is the regularization scalar. This penalty holds for data that comes from a trajectory *sampled evenly in time*.

We demonstrate this regularization on the 4-D Archimedean Spiral Lorenz systems, shown in Fig. 6.2b. As a reminder, this system's underlying manifold is the 3D Lorenz system, which lives in $\mathbb{R}^3$. Shown in Fig. 6.4 and Fig. 6.5 are example standard and arc-length regularized autoencoders trained over this dataset, respectively. Notably, the standard autoencoder learns a manifold that twists and contorts the "butterfly", with many regions on the manifold having much larger or smaller arc-lengths (i.e. vector field of different magnitudes). The arc-length regulated autoencoder learns a manifold that dilates the slow centers of the wings and contracts the fast moving edges into a "heart"-like shape that yields approximately uniform arc-lengths–i.e. the underlying vector field is approximately uniform in value.

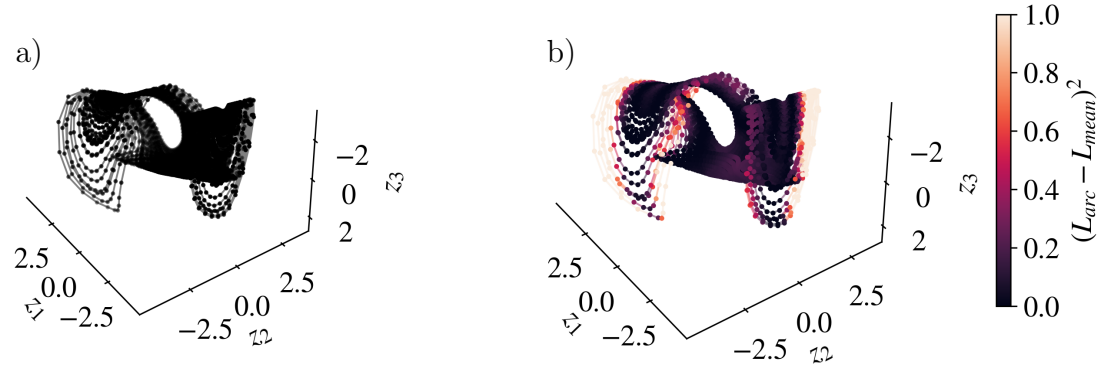

Figure 6.4: Standard latent space learned for the Archimedean Lorenz: a) trajectory b) trajectory colorized by curvature.

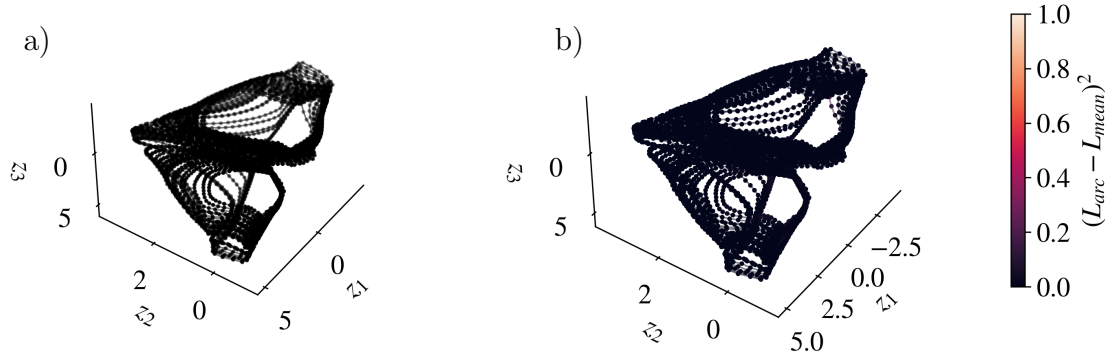

Figure 6.5: Arc-length regularized latent space ($\beta_{arc} = 0.1, L_0 = 0.5$) learned for the Archimedean Lorenz: a) trajectory b) trajectory colorized by curvature.

Curvature regularized latent space

In a similar vein to the above consideration for arc-length, another property to consider is *curvature*. While forcing the latent dynamics to have uniform speed may be beneficial for time-steppers, it may also be good to have dynamics on a manifold that does not have high curvatures or cusps. We can target curvature by regulating the angle between consecutive points in the latent space. This is just a simple modification of the above formulation,

$$\mathcal{L}(x_{t-1}, x_t, x_{t+1}; \theta) = \|x_t - \mathcal{D}(\mathcal{E}(x_t; \theta_E); \theta_D)\|^2 + \beta_{angle} \left[\phi_0 - \phi_z(z_{t-1}, z_t, z_{t+1})\right]^2, \qquad (6.4)$$

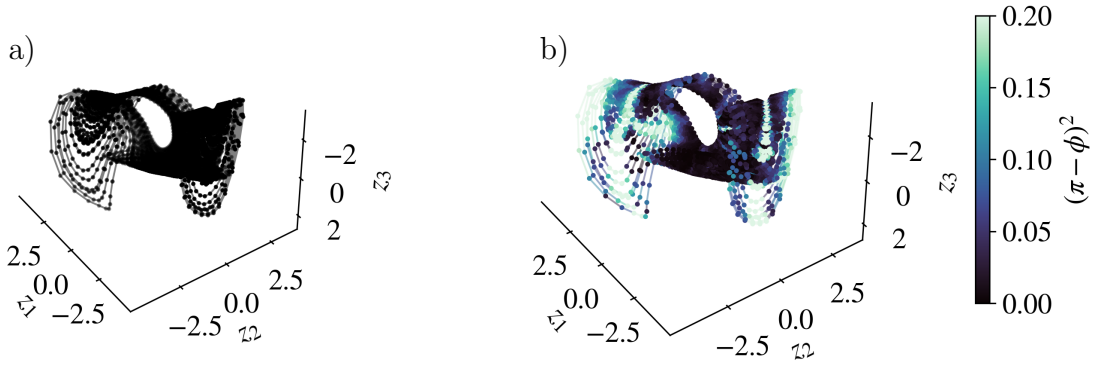

Figure 6.6: Standard latent space learned for the Archimedean Lorenz: a) trajectory b) trajectory colorized by curvature.

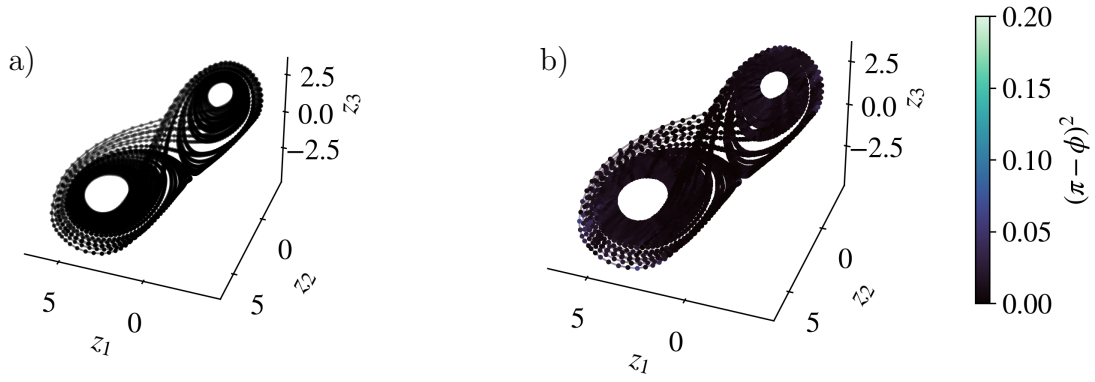

Figure 6.7: Curvature regularized latent space ($\beta_{angle} = 1.0, \phi_0 = \pi$) learned for the Archimedean Lorenz: a) trajectory b) trajectory colorized by curvature.

where ϕ_0 is a hyperparameter that sets the target angle between points in the manifold. As ϕ_0 approaches π, the loss targets a manifold with no curvature. Shown in Fig. 6.6 and Fig. 6.7 are example manifolds learned by standard and curvature regularized autoencoders trained on the same 4D Archimedean Lorenz data, respectively. Notice that the curvature-regularized autoencoder learn manifolds that appear unraveled without sharp twists or cusps compared to the standard autoencoder, which learns a contorted manifold.

Contrastive manifold coordinates

Finally, we note here the possibility of utilizing contrastive learning to develop manifold representations that are well separated. Contrastive learning can be easily extended to an autoencoder framework by simply adding a decoder to reconstruct the latent representation and an additional MSE loss. As contrastive methods require data augmentation, the natural augmentations to consider are naturally occurring symmetry groups–translations, reflections, etc. Other augmentations can include, noise or data corruption to promote learning of a robust latent space. Finally, a natural application is learning a latent space that captures multiple Reynolds numbers. The Reynolds numbers can serve to distinguish positive and negative sample pairs and allow us to learn a latent space that better separates multiple sets of dynamics.

References

[1] R. Agarwal, D. Schuurmans, and M. Norouzi. An optimistic perspective on offline reinforcement learning. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 104–114. PMLR, 13–18 Jul 2020.

[2] S. Arora, N. Cohen, W. Hu, and Y. Luo. Implicit regularization in deep matrix factorization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. de Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[3] D. W. Bechert and M. Bartenwerfer. The viscous flow on surfaces with longitudinal ribs. *Journal of Fluid Mechanics*, 206:105óÀÊ129, 1989.

[4] G. Beintema, A. Corbetta, L. Biferale, and F. Toschi. Controlling Rayleigh–Bénard convection via reinforcement learning. *Journal of Turbulence*, 21(9-10):585–605, 2020.

[5] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[6] T. R. Bewley. Flow control: new challenges for a new Renaissance. *Progression in Aerospace Sciences*, 37(1):21–58, 2001.

[7] C. Bishop. Bayesian PCA. In M. Kearns, S. Solla, and D. Cohn, editors, *Advances in Neural Information Processing Systems*, volume 11. MIT Press, 1998.

[8] S. Boccaletti, C. Grebogi, Y.-C. Lai, H. Mancini, and D. Maza. The control of chaos: theory and applications. *Physics Reports*, 329(3):103–197, 2000.

[9] J. D. Boskovic, L. Chen, and R. K. Mehra. Adaptive Control Design for Nonaffine Models Arising in Flight Control. *Journal of Guidance, Control, and Dynamics*, 27 (2):209–217, 2004.

[10] J. J. Bramburger, J. N. Kutz, and S. L. Brunton. Data-Driven Stabilization of Periodic Orbits. *IEEE Access*, 9:43504–43521.

[11] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[12] M. A. Bucci, O. Semeraro, A. Allauzen, G. Wisniewski, L. Cordier, and L. Mathelin. Control of chaotic systems by deep reinforcement learning. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 475(2231):20190351, 2019.

[13] N. B. Budanur and P. Cvitanović. Unstable Manifolds of Relative Periodic Orbits in the Symmetry-Reduced State Space of the KuramotoóÀÊSivashinsky System. *Journal of Statistical Physics*, 167(3-4):636–655, 2017.

[14] M. Budišić, R. Mohr, and I. Mezić. Applied Koopmanism. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 22(4):047510, 2012.

[15] F. Camastra and A. Staiano. Intrinsic dimension estimation: Advances and open problems. *Information Sciences*, 328:26–41, 2016.

[16] B. Chen, K. Huang, S. Raghupathi, I. Chandratreya, Q. Du, and H. Lipson. Automated discovery of fundamental variables hidden in experimental data. *Nature Computational Science*, 2(7):433–442, 2022.

[17] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural ordinary differential equations. *arXiv preprint arXiv:1806.07366*, 2019.

[18] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In H. D. III and A. Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR, 13–18 Jul 2020.

[19] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. Duvenaud. Neural Ordinary Differential Equations. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

[20] X. Cheng, Z. Qiao, X. Zhang, M. Quadrio, and Y. Zhou. Skin-friction reduction using periodic blowing through streamwise slits. *Journal of Fluid Mechanics*, 920:A50, 2021.

[21] H. Choi, P. Moin, and J. Kim. Direct numerical simulation of turbulent flow over riblets. *Journal of Fluid Mechanics*, 255:503óÀÊ539, 1993.

[22] H. Choi, P. Moin, and J. Kim. Active turbulence control for drag reduction in wall-bounded flows. *Journal of Fluid Mechanics*, 262:75–110, 1994.

[23] H. Choi, P. Moin, and J. Kim. Active turbulence control for drag reduction in wall-bounded flows. *Journal of Fluid Mechanics*, 262:75óÀÊ110, 1994.

[24] H. Choi, W.-P. Jeon, and J. Kim. Control of flow over a bluff body. *Annual Review of Fluid Mechanics*, 40(1):113–139, 2008.

[25] K.-S. Choi, T. Jukes, and R. Whalley. Turbulent boundary-layer control with plasma actuators. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 369(1940):1443–1458, 2011.

[26] K. Chua, R. Calandra, R. McAllister, and S. Levine. Deep reinforcement learning in a handful of trials using probabilistic dynamics models. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[27] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[28] Y. M. Chung and T. Talha. Effectiveness of active flow control for turbulent skin friction drag reduction. *Physics of Fluids*, 23(2):025102, 2011.

[29] S. Colabrese, K. Gustavsson, A. Celani, and L. Biferale. Flow Navigation by Smart Microswimmers via Reinforcement Learning. *Physical Review Letters*, 118(158004): 1–5, 2017.

[30] L. Condat. Fast projection onto the simplex and the $\boldsymbol{l}_1$ ball. *Mathematical Programming*, 158(1):575–585, 2016.

[31] H. S. M. Coxeter. *Regular polytopes*. Methuen, London, 1948.

[32] P. Cvitanović, R. L. Davidchack, and E. Siminos. On the State Space Geometry of the KuramotoóÀÊSivashinsky Flow in a Periodic Domain. *SIAM Journal on Applied Dynamical Systems*, 9(1):1–33, 2010.

[33] G. Cybenko. Approximation by Superpositions of a Sigmoidal Function. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

[34] S. Dieleman, K. W. Willett, and J. Dambre. Rotation-invariant convolutional neural networks for galaxy morphology prediction. *Monthly Notices of the Royal Astronomical Society*, 450(2):1441óÀÊ1459, 2015.

[35] S. Dieleman, J. De Fauw, and K. Kavukcuoglu. Exploiting Cyclic Symmetry in Convolutional Neural Networks. *arXiv preprint arXiv:1602.02660v2*, 2016.

[36] X. Ding, H. Chaté, P. Cvitanović, E. Siminos, and K. A. Takeuchi. Estimating the Dimension of an Inertial Manifold from Unstable Periodic Orbits. *Physical Review Letters*, 117(2):1–5, 2016.

[37] C. R. Doering, J. D. Gibbon, D. D. Holm, and B. Nicolaenko. Low-dimensional behaviour in the complex Ginzburg-Landau equation. *Nonlinearity*, 1(2):279–309, 1988.

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

[39] S. S. Du, S. M. Kakade, R. Wang, and L. F. Yang. Is a Good Representation Sufficient for Sample Efficient Reinforcement Learning? In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[40] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. Challenges of real-world reinforcement learning: definitions, benchmarks and analysis. *Machine Learning*, 110(9):2419–2468, 2021.

[41] T. Duriez, S. L. Brunton, and B. R. Noack. *Machine Learning Control óÀÃ Taming Nonlinear Dynamics and Turbulence*. Springer International Publishing, 2016.

[42] H. Eivazi, S. Le Clainche, S. Hoyas, and R. Vinuesa. Towards extraction of orthogonal and parsimonious non-linear modes from turbulent flows. *Expert Systems with Applications*, 202:117038, 2022.

[43] B. Esmaeili, H. Wu, S. Jain, A. Bozkurt, N. Siddharth, B. Paige, D. H. Brooks, J. Dy, and J.-W. van de Meent. Structured disentangled representations, 2018.

[44] D. Fan, L. Yang, Z. Wang, M. S. Triantafyllou, and G. E. Karniadakis. Reinforcement learning for bluff body active flow control in experiments and simulations. *Proceedings of the National Academy of Sciences*, 117(42):26091–26098, 2020.

[45] V. Feinberg, A. Wan, I. Stoica, M. I. Jordan, J. E. Gonzalez, and S. Levine. Model-Based Value Estimation for Efficient Model-Free Reinforcement Learning. *arXiv*, abs/1803.00101, 2018.

[46] D. Floryan and M. D. Graham. Charts and atlases for nonlinear data-driven models of dynamics on manifolds. *CoRR*, abs/2108.05928, 2021.

[47] D. Floryan and M. D. Graham. Charts and atlases for nonlinear data-driven models of dynamics on manifolds. *arXiv preprint arXiv:2108.05928*, 2021.

[48] C. Foias, O. Manley, and R. Temam. Modelling of the interaction of small and large eddies in two dimensional turbulent flows. *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 22(1):93–118, 1988.

[49] C. Foias, B. Nicolaenko, G. R. Sell, and R. Temam. Inertial manifold for the Kuramoto-Sivashinsky equation and an estimate of their lowest dimension. *J. Math. Pure Appl.*, 67:197–226, 1988.

[50] A. J. Fox and M. D. Graham. Predicting extreme events in a data-driven model of turbulent shear flow using an atlas of charts. *arXiv preprint arXiv:*, 2023.

[51] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018.

[52] B. GarcÕǎa-Archilla, J. Novo, and E. S. Titi. Postprocessing the galerkin method: a novel approach to approximate inertial manifolds. *SIAM Journal on Numerical Analysis*, 35(3):941–972, 1998.

[53] M. Gazzola, A. A. Tchieu, D. Alexeev, and A. D. Brauer. Learning to school in the presence of hydrodynamic interactions. *Journal of Fluid Mechanics*, 789:726–749, 2016.

[54] B. Geranmehr and S. R. Nekoo. Nonlinear suboptimal control of fully coupled non-affine six-DOF autonomous underwater vehicle using the state-dependent Riccati equation. *Ocean Engineering*, 96:248–257, 2015.

[55] J. F. Gibson. *Dynamical systems models of wall-bounded, shear-flow turbulence*. PhD thesis, Cornell University, New York, Jan. 2002.

[56] J. F. Gibson. Channelflow: a spectral Navier-Stokes simulator in C++. *University of New Hampshire*, (July):1–41, 2012.

[57] J. F. Gibson, J. Halcrow, and P. Cvitanović. Visualizing the geometry of state space in plane Couette flow. *Journal of Fluid Mechanics*, 611(1987):107–130, 2008.

[58] J. F. Gibson, S. A. F. Reetz, A. Ferraro, T. Kreilos, H. Schrobsdorff, M. Farano, A. F. Yesil, S. S. Schütz, M. Culpo, and T. M. Schneider. Channelflow 2.0. 2021.

[59] W. Gilpin. Deep reconstruction of strange attractors from time series. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[60] A. S. Glassner. *Graphics gems*. Academic Press, Boston, 1990.

[61] A. Glezer and M. Amitay. Synthetic jets. *Annual Review of Fluid Mechanics*, 34(1): 503–529, 2002.

[62] D. Goldstein, R. Handler, and L. Sirovich. Direct numerical simulation of turbulent flow over a modeled riblet covered surface. *Journal of Fluid Mechanics*, 302:333ó ÀÊ376, 1995.

[63] R. Gonzalez-Garcia, R. Rico-Martinez, and I. G. Kevrekidis. Identification of distributed parameter systems: A neural net based approach. *Computers & Chemical EngineeringComputers & Chemical Engineering*, 22:S965–S968, 1998.

[64] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[65] M. D. Graham. Drag reduction and the dynamics of turbulence in simple and complex fluids. *Physics of Fluids*, 26(10):101301, 2014.

[66] M. D. Graham and D. Floryan. Exact Coherent States and the Nonlinear Dynamics of Wall-Bounded Turbulent Flows. *Annual Review of Fluid Mechanics*, 53(1):227–253, 2021.

[67] M. D. Graham and D. Floryan. Exact Coherent States and the Nonlinear Dynamics of Wall-Bounded Turbulent Flows. *Annu. Rev. Fluid Mech.*, 53:227–253, 2021.

[68] M. D. Graham, P. H. Steen, and E. S. Titi. Computational efficiency and approximate inertial manifolds for a Bénard convection system. *Journal of Nonlinear Science*, 3(1): 153–167, 1993.

[69] R. O. Grigoriev. Symmetry and control : spatially extended chaotic systems. *Physica D*, 140(May 1999):171–192, 2000.

[70] R. O. Grigoriev and M. C. Cross. Controlling physical systems with symmetries. *Phys. Rev. E*, 57(2):1550–1554, 1998.

[71] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, B. Piot, k. kavukcuoglu, R. Munos, and M. Valko. Bootstrap your own latent - a new approach to self-supervised learning. In H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21271–21284. Curran Associates, Inc., 2020.

[72] L. Guastoni, J. Rabault, P. Schlatter, H. Azizpour, and R. Vinuesa. Deep reinforcement learning for turbulent drag reduction in channel flows, 2023.

[73] F. Gueniat, L. Mathelin, and Y. Hussaini. A statistical learning strategy for closed-loop control. *Theoretical Computational Fluid Dynamics*, 30:497–510, 2016.

[74] S. Gunasekar, B. E. Woodworth, S. Bhojanapalli, B. Neyshabur, and N. Srebro. Implicit regularization in matrix factorization. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[75] D. Ha and J. Schmidhuber. Recurrent World Models Facilitate Policy Evolution. In S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 2455–2467, 2018.

[76] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[77] J. Hamilton, J. Kim, and F. Waleffe. Regeneration mechanisms of near-wall turbulence structures. *Journal of Fluid Mechanics*, 287:317–348, 1995.

[78] E. P. Hammond, T. R. Bewley, and P. Moin. Observed mechanisms for turbulence attenuation and enhancement in opposition-controlled wall-bounded flows. *Physics of Fluids*, 10(9):2421–2423, 1998.

[79] J. P. Hespanha. *Linear Systems Theory*. Princeton University Press, 2009.

[80] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*, 2017.

[81] E. J. Hinch. *Perturbation Methods*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 1991.

[82] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[83] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Turbulence, Coherent Structures, Dynamical Systems and Symmetry*, volume 36. 1998.

[84] P. Holmes, J. L. Lumley, G. Berkooz, and C. W. Rowley. *Galerkin projection*, pages 106–129. Cambridge Monographs on Mechanics. Cambridge University Press, 2 edition, 2012.

[85] E. Hopf. A mathematical example displaying features of turbulence. *Communications on Pure and Applied Mathematics*, 1(4):303 – 322, 1948-12.

[86] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences*, 79(8):2554–2558, 1982.

[87] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

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

[89] J. I. Ibrahim, Q. Yang, P. Doohan, and Y. Hwang. Phase-space dynamics of opposition control in wall-bounded turbulent flows. *Journal of Fluid Mechanics*, 861:296ÀẼ54, 2019.

[90] M. Inubushi, S.-i. Takehiro, and M. Yamada. Regeneration cycle and the covariant lyapunov vectors in a minimal wall turbulence. *Phys. Rev. E*, 92:023022, Aug 2015.

[91] M. Jaderberg, K. Simonyan, A. Zisserman, K. Kavukcuoglu, and G. Deepmind. Spatial Transformer Networks. In *Advances in Neural Information Processing Systems*, pages 1–9, 2015.

[92] H. Jaeger. The "echo state" approach to analysing and training recurrent neural networks. GMD Report 148, GMD - German National Research Institute for Computer Science, 2001.

[93] F. Jauberteau, C. Rosier, and R. Temam. The nonlinear Galerkin method in computational fluid dynamics. *Applied Numerical Mathematics*, 6(5):361–370, 1990.

[94] C. E. P. D. JesÕẽs and M. D. Graham. Data-driven low-dimensional dynamic model of Kolmogorov flow. *Physical Review Fluids*, 8(4):044402, 2023.

[95] J. Jiménez and A. Lozano-Durán. Coherent structures in wall-bounded turbulence. *Journal of Fluid Mechanics*, 842, 2018.

[96] J. Jiménez and P. Moin. The minimal flow unit in near-wall turbulence. *Journal of Fluid Mechanics*, 225:213óÀẼ240, 1991.

[97] L. Jing, J. Zbontar, and Y. LeCun. Implicit rank-minimizing autoencoder. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 14736–14746. Curran Associates, Inc., 2020.

[98] I. Jolliffe. *Principal Component Analysis*. Springer Verlag, 1986.

[99] J. Karhunen and J. Joutsensalo. Representation and separation of signals using nonlinear PCA type learning. *Neural Networks*, 7(1):113–127, 1994.

[100] M. B. Kennel, R. Brown, and H. D. I. Abarbanel. Determining embedding dimension for phase-space reconstruction using a geometrical construction. *Phys. Rev. A*, 45: 3403–3411, Mar 1992.

[101] N. Keriven and G. Peyré. Universal Invariant and Equivariant Graph Neural Networks. In *Conference on Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[102] D. P. Kingma and J. L. Ba. Adam: A method for stochastic optimization. *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pages 1–15, 2015.

[103] D. P. Kingma and M. Welling. Auto-encoding variational bayes, 2022.

[104] T. N. Kipf and M. Welling. Variational graph auto-encoders, 2016.

[105] L. Kleiser and U. Schumann. *Treatment of Incompressibility and Boundary Conditions in 3-D Numerical Spectral Simulations of Plane Channel Flows*, pages 165–173. Vieweg+Teubner Verlag, Wiesbaden, 1980.

[106] A. Krizhevsky and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems*, pages 1–9, 2012.

[107] J. N. Kutz, S. L. Brunton, B. W. Brunton, and J. L. Proctor. *Dynamic Mode Decomposition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2016.

[108] A. Lasota and M. C. Mackey. *Chaos, Fractals and Noise: stochastic aspects of dynamics*. Springer. Springer, New York, second edition, 1994.

[109] C. Lee, J. Kim, D. Babcock, and R. Goodman. Application of neural networks to turbulence control for drag reduction. *Physics of Fluids*, 9:1740–1747, 1997.

[110] J. Lee. *Introduction to Smooth Manifolds*. Graduate Texts in Mathematics. Springer, 2003.

[111] J. M. Lee. *Introduction to Smooth Manifolds*, volume 218 of *Springer New York*. Springer New York, 2012.

[112] R. Legenstein and W. Maass. Edge of chaos and prediction of computational performance for neural circuit models. *Neural Networks*, 20(3):323–334, 2007. Echo State Networks and Liquid State Machines.

[113] K. Lenc and A. Vedaldi. Understanding Image Representations by Measuring Their Equivariance and Equivalence. *International Journal of Computer Vision*, 127(5):456–476, 2018.

[114] E. Levina and P. Bickel. Maximum likelihood estimation of intrinsic dimension. In L. Saul, Y. Weiss, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, volume 17. MIT Press, 2004.

[115] S. Levine, A. Kumar, G. Tucker, and J. Fu. Offline reinforcement learning: Tutorial, review, and perspectives on open problems. *CoRR*, abs/2005.01643, 2020.

[116] J. Li and M. Zhang. Reinforcement-learning-based control of confined cylinder wakes with stability analyses. *Journal of Fluid Mechanics*, 932:A44, 2022.

[117] J. Li and M. Zhang. Reinforcement-learning-based control of confined cylinder wakes with stability analyses. *Journal of Fluid Mechanics*, 932:A44, 2022.

[118] Y. Li. Deep reinforcement learning: An overview. *CoRR*, abs/1701.07274, 2017.

[119] T. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous learning control with deep reinforcement. In *International Conference on Learning Representations*, pages 1–26, 2016.

[120] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning, 2019.

[121] J. Ling, R. Jones, and J. Templeton. Machine learning strategies for systems with invariance properties. *Journal of Computational Physics*, 318:22–35, 2016.

[122] A. J. Linot and M. D. Graham. Deep learning to discover and predict dynamics on an inertial manifold. *Phys. Rev. E*, 101:062209, 2020.

[123] A. J. Linot and M. D. Graham. Data-driven reduced-order modeling of spatiotemporal chaos with neural ordinary differential equations. *arXiv preprint arXiv:2109.00060*, 2021.

[124] A. J. Linot and M. D. Graham. Dynamics of a data-driven low-dimensional model of turbulent minimal Couette flow. *arXiv*, 2023.

[125] A. J. Linot and M. D. Graham. Dynamics of a data-driven low-dimensional model of turbulent minimal couette flow. *arXiv preprint arXiv:2301.04638*, 2023.

[126] A. J. Linot, J. W. Burby, Q. Tang, P. Balaprakash, M. D. Graham, and R. Maulik. Stabilized neural ordinary differential equations for long-time forecasting of dynamical systems. *arXiv preprint arXiv:2203.15706*, 2022.

[127] A. J. Linot, K. Zeng, and M. D. Graham. PyChannel. `https://github.com/alinot5/PyChannel`, 2023.

[128] A. V. Little, Y.-M. Jung, and M. Maggioni. Multiscale estimation of intrinsic dimensionality of data sets. volume FS-09-04, page 26 óÀẼ 33, 2009.

[129] X.-Y. Liu and J.-X. Wang. Physics-informed Dyna-style model-based deep reinforcement learning for dynamic control. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 477(2255):20210618, 2021.

[130] E. N. Lorenz. Deterministic Nonperiodic Flow. *Journal of the Atmospheric Sciences*, 20(2):130–141, 1963.

[131] I. Loshchilov and F. Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[132] M. Lukosevicius and H. Jaeger. Reservoir Computing Approaches to Recurrent Neural Network Training. 2009.

[133] M. Marion and R. Temam. Nonlinear Galerkin methods: The finite elements case. *Numerische Mathematik*, 57(1):205–226, 1990.

[134] L. McInnes, J. Healy, and J. Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020.

[135] S. Meyn. *Control Systems and Reinforcement Learning*. Cambridge University Press, 2022.

[136] V. Mnih. Human-level control through deep reinforcement learning. *Nature*, 518: 529–533, 2015.

[137] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR.

[138] J. Moehlis, H. Faisst, and B. Eckhardt. A low-dimensional model for turbulent shear flows. *New Journal of Physics*, 6(1):56, may 2004.

[139] A. Mousavi-Hosseini, S. Park, M. Girotti, I. Mitliagkas, and M. A. Erdogdu. Neural networks efficiently learn low-dimensional representations with SGD. In *The Eleventh International Conference on Learning Representations*, 2023.

[140] O. Nachum, S. S. Gu, H. Lee, and S. Levine. Data-efficient hierarchical reinforcement learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[141] A. Nagabandi, G. Kahn, R. S. Fearing, and S. Levine. Neural Network Dynamics for Model-Based Deep Reinforcement Learning with Model-Free Fine-Tuning. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 7559–7566. IEEE, 2018.

[142] A. Ng et al. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

[143] D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. *Cogn. Process.*, 12(4):319–340, 2011.

[144] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. Pondé, D. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang. Dota 2 with Large Scale Deep Reinforcement Learning. *ArXiv Preprint*, 2019.

[145] E. Ott, C. Grebogi, and J. A. Yorke. Controlling Chaos. *Physical Review Letters*, 64 (11):1196–1199, 1990.

[146] S. E. Otto and C. W. Rowley. Koopman operators for estimation and control of dynamical systems. *Annual Review of Control, Robotics, and Autonomous Systems*, 4 (1):59–87, 2021.

[147] R. Paris, S. Beneddine, and J. Dandois. Robust flow control and optimal sensor placement using deep reinforcement learning. *Journal of Fluid Mechanics*, 913:A25, 2021.

[148] J. Park and H. Choi. Machine-learning-based feedback control for drag reduction in a turbulent channel flow. *Journal of Fluid Mechanics*, 904:A24, 2020.

[149] R. Peyret. *Spectral Methods for Incompressible Viscous Flow*. Applied Mathematical Sciences. Springer New York, 2002.

[150] J. L. Proctor, S. L. Brunton, and J. N. Kutz. Dynamic Mode Decomposition with Control. *SIAM Journal on Applied Dynamical Systems*, 15(1):142–161, 2016.

[151] K. Pyragas. Continuous control of chaos by self-controlling feedback. *Physics Letters A*, 170(6):421–428, 1992.

[152] S. Qin, S. Wang, J. Rabault, and G. Sun. An application of data driven reward of deep reinforcement learning by dynamic mode decomposition in active flow control. *ArXiv Preprint*, 2021.

[153] M. Quadrio and A. Baron. Turbulent Drag Reduction by Spanwise Wall Oscillations. *Applied Scientific Research*, 55:311–326, 1996.

[154] M. Quadrio and P. Ricco. Critical assessment of turbulent drag reduction through spanwise wall oscillations. *Journal of Fluid Mechanics*, 521:251–271, 2004.

[155] M. Quadrio and P. Ricco. Critical assessment of turbulent drag reduction through spanwise wall oscillations. *Journal of Fluid Mechanics*, 521:251óÀẼ271, 2004.

[156] J. Rabault, M. Kuchta, A. Jensen, U. Réglade, and N. Cerardi. Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control. *Journal of Fluid Mechanics*, 865:281–302, 2019.

[157] S. Racanière, T. Weber, D. Reichert, L. Buesing, A. Guez, D. Jimenez Rezende, A. Puigdomènech Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra. Imagination-augmented agents for deep reinforcement learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

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

[159] M. Ramesh and S. Narayanan. Chaos control by nonfeedback methods in the presence of noise. *Chaos, Solitons and Fractals*, 10(9):1473–1489, 1999.

[160] S. Ravanbakhsh and J. Schneider. In *International Conference on Machine Learning*. PMLR.

[161] N. Razin and N. Cohen. Implicit regularization in deep learning may not be explainable by norms. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 21174–21187. Curran Associates, Inc., 2020.

[162] H. Rebbeck and K.-S. Choi. A wind-tunnel experiment on real-time opposition control of turbulence. *Physics of Fluids*, 18(3):035103, 2006.

[163] G. Reddy, J. Wong-ng, A. Celani, T. J. Sejnowski, and M. Vergassola. Glider soaring via reinforcement learning in the field. *Nature*, 562:236–238, 2018.

[164] F. Ren, J. Rabault, and H. Tang. Applying deep reinforcement learning to active flow control in weakly turbulent conditions. *Physics of Fluids*, 33(3):037121, 2021.

[165] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ICML'11, page 833óÀẼ840, Madison, WI, USA, 2011. Omnipress.

[166] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[167] Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. *International journal of computer vision*, 40(2):99, 2000.

[168] A. Sannai, Y. Takai, and M. Cordonnier. Universal approximations of permutation invariant / equivariant functions by deep neural networks. *arXiv preprint arXiv:1903.01939v3*, (1989):1–17, 2019.

[169] T. Sauer, J. A. Yorke, and M. Casdagli. Embedology. *Journal of Statistical Physics*, 65(3):579–616, 1991.

[170] P. J. Schmid. Dynamic mode decomposition of numerical and experimental data. *Journal of Fluid Mechanics*, 656:5–28, 2010.

[171] W. Schoppa and F. Hussain. A large-scale control strategy for drag reduction in turbulent boundary layers. *Physics of Fluids*, 10(1049):1–4.

[172] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In F. Bach and D. Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR.

[173] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017.

[174] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[175] S. Scott Collis, R. D. Joslin, A. Seifert, and V. Theofilis. Issues in active flow control: theory, control, simulation, and experiment. *Progress in Aerospace Sciences*, 40(4): 237–289, 2004.

[176] J. Shawe-taylor. Symmetries and Discriminability in Feedforward Network Architectures. *IEEE Transactions of Neural Networks*, 4(5):816–826, 1993.

[177] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. V. D. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, and K. Kavukcuoglu. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529 (7585):484–489, 2016.

[178] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis. Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[179] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*, 362(6419):1140–1144, 2018.

[180] C. R. Smith and S. P. Metzler. The characteristics of low-speed streaks in the near-wall region of a turbulent boundary layer. *Journal of Fluid Mechanics*, 129:27–54, 1983.

[181] T. R. Smith, J. Moehlis, and P. Holmes. Low-dimensional modelling of turbulence using the proper orthogonal decomposition: A tutorial. *Nonlinear Dynamics*, 41(1): 275–307, 2005.

[182] K. so Choi. Near-wall structure of a turbulent boundary layer with riblets. *Journal of Fluid Mechanics*, 208(April 2006):417–458, 1989.

[183] T. Sonoda, Z. Liu, T. Itoh, and Y. Hasegawa. Reinforcement learning of control strategies for reducing skin friction drag in a fully developed channel flow, 2022.

[184] P. R. Spalart, R. D. Moser, and M. M. Rogers. Spectral methods for the navier-stokes equations with one infinite and two periodic directions. *Journal of Computational Physics*, 96(2):297–324, 1991.

[185] S. H. Strogatz. *Nonlinear Dynamics and Chaos*. 1994.

[186] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction, Second Edition*. MIT Press, 2018.

[187] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[188] K. A. Takeuchi, H.-l. Yang, F. Ginelli, G. Radons, and H. Chaté. Hyperbolic decoupling of tangent space and effective dimension of dissipative systems. *Phys. Rev. E*, 84: 046214, Oct 2011.

[189] R. Temam. *Infinite-Dimensional Dynamical Systems in Mechanics and Physics*. Springer, New York, NY, 1977.

[190] R. Temam. Do inertial manifolds apply to turbulence? *Physica D: Nonlinear Phenomena*, 37(1-3):146–152, 1989.

[191] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[192] I. Tolstikhin, O. Bousquet, S. Gelly, and B. Schoelkopf. Wasserstein auto-encoders. In *International Conference on Learning Representations*, 2018.

[193] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[194] P. Varela, P. SuÕárez, F. AlcÕántara-ÕÁvila, A. MirÕẽ, J. Rabault, B. Font, L. M. GarcÕắa-Cuevas, O. Lehmkuhl, and R. Vinuesa. Deep reinforcement learning for flow control exploits different physics for increasing reynolds number regimes. *Actuators*, 11(12), 2022.

[195] S. Vashishtha and S. Verma. Restoring chaos using deep reinforcement learning. *Chaos*, 30(3), 2020.

[196] S. Verma, G. Novati, and P. Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proc. Nat. Acad. Sci.*, 115(23): 5849–5854, 2018.

[197] C. Vignon, J. Rabault, and R. Vinuesa. Recent advances in applying deep reinforcement learning for flow control: perspectives and future directions. *Physics of Fluids*, 0(ja):null, 2023.

[198] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575 (7782):350–354, 2019.

[199] J. Viquerat, P. Meliga, and E. Hachem. A review on deep reinforcement learning for fluid mechanics: an update. *arXiv preprint arXiv:2107.12206*, 2023.

[200] P. S. Virk. Drag reduction fundamentals. *AIChE Journal*, 21(4):625–656, 1975.

[201] D. Viswanath. Recurrent motions within plane couette turbulence. *Journal of Fluid Mechanics*, 580:339óÀẼ358, 2007.

[202] P. Vlachas, J. Pathak, B. Hunt, T. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. Backpropagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics. *Neural Networks*, 126:191–217, 2020.

[203] P. R. Vlachas, G. Arampatzis, C. Uhler, and P. Koumoutsakos. Multiscale simulations of complex systems by learning their effective dynamics. *Nature Machine Intelligence*, 4(4):359–366, 2022.

[204] N. Wahlström, T. B. Schön, and M. P. Deisenroth. From Pixels to Torques: Policy Learning with Deep Dynamical Models. *arXiv*, abs/1502.02251, 2015.

[205] F. Waleffe. On a self-sustaining process in shear flows. *Physics of Fluids*, 883(1997), 1997.

[206] Z. Y. Wan, P. Vlachas, P. Koumoutsakos, and T. Sapsis. Data-assisted reduced-order modeling of extreme events in complex dynamical systems. *PLoS ONE*, 13(5):1–22, 2018.

[207] Y.-Z. Wang, Y.-F. Mei, N. Aubry, Z. Chen, P. Wu, and W.-T. Wu. Deep reinforcement learning based synthetic jet control on disturbed flow over airfoil. *Physics of Fluids*, 34(3):033606, 2022.

[208] Z. Wang and C. Guet. Reconstructing a dynamical system and forecasting time series by self-consistent deep learning. *CoRR*, abs/2108.01862, 2021.

[209] Z. Wang and C. Guet. Self-consistent learning of neural dynamical systems from noisy time series. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 6 (5):1103–1112, 2022.

[210] M. Watter, J. T. Springenberg, J. Boedecker, and M. A. Riedmiller. Embed to Control: A Locally Linear Latent Dynamics Model for Control from Raw Images. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 2746–2754, 2015.

[211] M. Welling and T. S. Cohen. In *International Conference on Machine Learning*, pages 2990–2999.

[212] H. Whitney. The self-intersections of a smooth n-manifold in 2n-space. *Annals of Mathematics*, 45(2):220–246, 1944.

[213] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3):229–256, 1992.

[214] H.-l. Yang and G. Radons. Geometry of inertial manifolds probed via a Lyapunov projection method. *Phys. Rev. Lett.*, 108:154101, Apr 2012.

[215] H. L. Yang, K. A. Takeuchi, F. Ginelli, H. Chaté, and G. Radons. Hyperbolicity and the effective dimension of spatially extended dissipative systems. *Physical Review Letters*, 102(7):1–4, 2009.

[216] J. Zbontar, L. Jing, I. Misra, Y. LeCun, and S. Deny. Barlow twins: Self-supervised learning via redundancy reduction. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 12310–12320. PMLR, 18–24 Jul 2021.

[217] S. Zelik. Inertial manifolds and finite-dimensional reduction for dissipative PDEs. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, 144(6):1245–1327, 2013.

[218] S. Zelik. Attractors. Then and now. *arXiv*, 2022.

[219] K. Zeng and M. D. Graham. Symmetry reduction for deep reinforcement learning active control of chaotic spatiotemporal dynamics. *Phys. Rev. E*, 104:014210, Jul 2021.

[220] K. Zeng, A. Linot, and M. D. Graham. Learning turbulence control strategies with data-driven reduced-order models and deep reinforcement learning. In H. J. Sung, A. Johansson, B. McKeon, M. Oberlack, S. Tavoularis, and K. Suga, editors, *12th International Symposium on Turbulence and Shear Flow Phenomena (TSFP12) Osaka, Japan (Online), July 19-22, 2022*, 2022.

[221] K. Zeng, A. J. Linot, and M. D. Graham. Data-driven control of spatiotemporal chaos with reduced-order neural ODE-based models and reinforcement learning. *Proceedings of the Royal Society A*, 478(2267):20220297, 2022.

[222] S. Zhao, J. Song, and S. Ermon. Infovae: Information maximizing variational autoencoders, 2018.

[223] H. Zou, T. Hastie, and R. Tibshirani. Sparse principal component analysis. *Journal of Computational and Graphical Statistics*, 15(2):262 óÀÊ 286, 2004.