Model-based Investigation of Cascade Dynamics on Multi-layer Networks

Yaofeng Zhong

A Dissertation

Presented to the Faculty

of Princeton University

in Candidacy for the Degree

of Doctor of Philosophy

Recommended for Acceptance

by the Department of

Mechanical and Aerospace Engineering

Adviser: Naomi Ehrich Leonard

November 2020

Abstract

This dissertation examines the spread of an activity among a network of heterogeneous agents with multiple communication modalities, the spread of a continuous activity level and the learning of rigid body dynamics from data.

Motivated by the remarkable ability of animal groups in responding to a real threat while not responding to a spurious one, we investigate how information propagates in a network of agents using the linear threshold model (LTM). We found that the key to the group's sensitivity to a threat and robustness to noise is the existence of multiple communication modalities. To distinguish different communication modalities, we extend the LTM to multiplex networks. We propose protocols for an agent to synthesize information from different communication modalities and study groups with heterogeneous protocols. We propose a provably accurate algorithm and an efficient approximate algorithm to compute the size of spread given a set of early adopters.

We generalize the discrete LTM into a continuous threshold model (CTM) and analyze cascade dynamics. We rigorously show the existence of a pitchfork bifurcation in the dynamics. Abrupt change of all agents' activity levels happens when the pitchfork is subcritical. We show how high disparity in the thresholds and network structure lead to a cascade.

Both the LTM and CTM assume the state of agents can be directly observed, which is often the case in biological systems. To incorporate these insights into designing coordinated engineering systems, e.g., a robot team, an agent need to infer the activity of neighboring agents from sensor data. We study the scenario where the activity is indicated by the configuration or the dynamics of a rigid body system. We propose a neural network model to learn rigid body dynamics assuming the sensor data are raw images. We design a coordinate-aware variational auto-encoder (VAE) to infer coordinates from image data and learn Lagrangian/Hamiltonian dynamics on the inferred coordinates. We show that the prior of Lagrangian/Hamiltonian dynamics improves accuracy and generalization. We show the coordinate-aware VAE is crucial in learning interpretable coordinates. This interpretability benefits long term prediction and allows for synthesis of energy-based controllers.

Acknowledgements

I am grateful for the opportunities available to me as a graduate student at Princeton and the people I met along the journey. I would like to take this opportunity to express my gratitude to those who supported me and shaped me into who I am.

I am indebted to my advisor Naomi Leonard, who provided help and support in almost all aspects in my PhD journey. I am grateful for Naomi taking me as her student even though I have no background in control except math. Naomi provided a perfect environment for me to conduct research. I never have to worry about funding for research and attending conferences, never have to work on a problem that I am not willing to work on and never have to present results that I do not feel ready to present. Naomi's advising style is both hands-off and hands-on. We don't have regular individual meetings but Naomi would always schedule a time to meet when I requested. This gives me great freedom to progress at my own pace. Many times during our meeting, I am impressed by Naomi's passion for research and her creativity by thinking out of the box, which has a profound impact on my development as a researcher. When it comes to presenting ideas and writing manuscripts, Naomi would be meticulous in every detail and help the presentation or manuscript to be a better one. I learned tremendously from how to formalize innovative research ideas to how to make these ideas accessible to audiences and readers.

I would like to thank my PhD committee members, Clancy Rowley and Amir Ali Ahmadi. Both of them provided guidance and feedback starting from my general exam and I appreciate their support throughout the course of my PhD. Clancy also plays a role as a reader of my dissertation and Amirali as my dissertation examiner. Vaibhav Srivastava has already helped me start my research from scratch and provided me with concrete instructions while he was a postdoc in our group, yet he continued his support for my research after joining Michigan State University as a professor and graciously agreed to be my dissertation reader. Thank you Ani Majumdar for providing helpful feedback on my research and being my dissertation examiner.

I am thankful to former and current members of the Leonard group. Thank you Katie Fitch and Liz Davison for introducing your research to me even before I join the group; Will Scott for our late night conversations in the office when you took a break from dissertation writing; Peter Landgren for organizing robot camp (twice!) and provided me with guidance to operate the tank lab; Bec Gray for being creative and suggestive; Renato Pagliara for being supportive when I got stuck with my research; Anthony Savas for our time together working on APC 503 assignments; Anastasia Bizyaeva for hosting lab coffee hours during WFH period and maintaining our group

slack channel; Justice Mason for taking on part of my research and probably would apply and extend it to applications I could not imagine; Udari Madhushani, Mari Kawakatsu, Charlotte Cathcart and Yunxiu (Joey) Zhou for being both critical and supportive, and providing valuable feedback on my research. In addition, I would like to thank our former postdocs, Vaibhav Srivastava, Biswadip Dey, Karla Kvaternik, Kayhan Ozcimder and Zahra Aminzare, all of whom I viewed as role models. Vaibhav and Biswa are also my wonderful collaborators. Whenever I encountered challenges in my research and asked them for help, they would direct me to some references immediately addressing my concerns. I would also like to thank our current postdocs Christine Allen-Blanchette for hosting weekly reading groups, and Shinkyu Park for sharing his experience as an international student with me.

The MAE department provides a friendly environment which I enjoyed in the last five years. In fact, I felt the vibrant and welcoming environment even before I started my graduate life, during the MAE Open House, which is my first trip to the U.S. Special thanks to our graduate administrator Jill Ray, who made my trip to the Open House extremely smooth, despite that I require an extra day of accommodation due to availability of flight tickets. Jill, together with Theresa Russo, helped me navigate the graduate program. I would like to thank my MAE cohorts, Mengya (Mia) Hu, Fan Yang, David Feng, Katherine Kokmanian, Tianhan Zhang, Adam Fisher, Yingxian (Estella) Yu, Alex Novoselov, Fan Yang, Leonid Pogorelyuk, Kristofer Meehan, Anthony Savas, Tasman Powis, Vivian Steyert, Thomas and Tara Hudson, for creating a welcoming and splendid environment. I would like to thank former MAE students He Sun and Hao Zhang for their advises on almost every aspects of graduate life and our conversations on academic problems and challenges. Thank you Yibin Zhang, Thomas Hudson, Kerry Klemmer and Daniel Dudt for organizing various kinds of MAE social events.

I would like to thank my English tutor, Jan Papas Gramer, and my host family, Mary Ann Cavallaro, connected through Davis International Center. Jan accompanied me through my hard time in my first year at Princeton. Mary Ann helped me better fit in the culture and life at Princeton. I appreciate their love and care for me.

My graduate life at Princeton would not have started without the support of my undergraduate mentor and supervisor Hongzhi Zhong, who help me get into the world of scientific research. Thank you Mengwu Guo for being my role model of a PhD student and all the help you provided to me. Thank you my undergraduates classmates and I enjoyed our friendship, trips and four years' time spent together. Special thanks to friends I met during undergrad in Tsinghua University: Shumiao Ouyang, Xinlei Sheng, Kangning Liu, Tiancheng Yu, Zehao Pan and Yiyun Cao, all of whom

remarkably ended up in the east coast and we had an unforgettable get-together near Princeton.

I would like to thank Liqun (Zoe) Peng, for her love and support. She is thoughtful and encouraging. She always cares for me, lightens me up when I am down, opens me up when I am quiet. We shared joy and happiness, and witnessed each other's progress over the years. Thank you for proposing our "relax time" from 11PM to 12AM, when we detach ourselves from a whole day's work and enjoy our favorite books or episodes of our favorite shows together. This is one of my favorite moments in grad school and it miraculously fixed my issue of having trouble to sleep at midnight. (Well, I apologize for taking one of these precious moments to write down these words!)

Above all, I would like to devote my deepest gratitude to my parents for their unconditional love through my life. It is their endless support and encouragement that has enabled me to pursue my own passion. This dissertation marks a milestone of my educational journey of more than 20 years, where my parents played a major role. Their continuous support and respect of my choice would propel me forward to pursue future endeavors.

This dissertation has been sponsored by Office of Naval Research grants N00014-14-1-0635, N00014-19-1-2556 and N00014-18-1-2873, and Army Research Office grant W911NF-18-1-0325.

This dissertation carries T#3405 in the records of the
Department of Mechanical and Aerospace Engineering.

To my parents

献给我的父母

Contents

Part I

Cascade Dynamics and the Learning of Dynamics

Chapter 1

Introduction

1.1 Overview

Understanding the spread of an innovation or activity within a group of agents is important for designing the spread in engineering systems and controlling the spread in biological systems. In a coordinated robot team, a robot that detects an external signal and takes action can spread the behavior to the team. In animal groups such as fish schools and bird flocks, an animal that senses a predator or threat would respond to it and the response can spread to other animals in the group. A novel high-quality service or product would be adopted by a large number of people through word of mouth recommendations from early users.

The systems we study involve multiple agents and the activity spread from one agent to another through the sensing or communication channels among the agents. Traditionally, multi-agent systems are modelled by graphs, where each node in the graph represents an agent and each edge represents communication between two agents. However, the graph structure cannot distinguish different communication channels or sensing modalities in the group. Robots can be equipped with sensors of different types and ranges. Fish can sense others by their eyes and lateral lines. We as human beings can communicate face-to-face and through social media. Distinguishing these different communication modalities is essential to understanding the complex cascading behavior in biological systems. In order to model multiple communication modalities, we consider a collection of graphs, each of which models a communication channel. This framework is referred to as the multiplex network and graphs in the network are referred to as layers.

We study the spread of an activity using the linear threshold model (LTM). The idea behind the

LTM is that each agent has a threshold and if the fraction of its active neighbors is greater than its threshold, it would turn from inactive into active. We extend the LTM to multiplex networks. When multiple communication channels are taken into account, each agent needs a protocol to combine signals from different layers. We design a general protocol and then investigate the two key limiting protocols: Protocol OR models agents that are readily activated, and Protocol AND models agents that are conservatively activated.

The size of the spread given a few early adopters is an important metric to measure the influence of agents in terms of spreading the activity. When there is only one early adopter, the size of the spread at steady state is called the cascade centrality of that agent. We propose an algorithm to compute the influence spread of a set of agents and the cascade centrality of single agents based on network structure. Since influence spread in large networks is computationally intractable, we propose an approximate algorithm by leveraging probabilistic inference in Bayesian networks. Using cascade centrality, we study the spread of activity, in particular, the role of distinguishing multiple communication modalities and the role of agent heterogeneity in protocols. We show how Protocol OR enhances a cascade and Protocol AND diminishes a cascade as compared to a traditional network not distinguishing different communication modalities. We also show how heterogeneity in protocols plays a role in the tradeoff between the sensitivity of a cascade to a real external signal and the robustness of a cascade to a spurious external signal.

The LTM can be viewed as a discrete-time, discrete-state dynamical system. Motivated by the idea behind the LTM, we propose a continuous threshold model (CTM) to study the spread of an activity so that we can leverage the rich tools in the world of continuous dynamical systems to analyze cascade. We apply the model to a class of networks - a chain of three clusters - from which we can draw fundamental insights. We rigorously prove that there exists a pitchfork bifurcation in the dynamics and that a subcritical pitchfork leads to a cascade response while a supercritical pitchfork leads to a contained response. We show the surprising result that a high disparity between the two clusters at the ends of the chain leads to a cascade.

Both the LTM and the CTM models assume agents possess a single state variable, which can be observed by the others. In applications, the state of “active" or “inactive" might need to be inferred instead of being directly observed, especially if each agent has its own dynamics. For example, a robot in a coordinated team equipped with camera sensors needs to infer the activity of others from image data. To infer the activity from image pixels directly is challenging since instead of pixel values, activity is usually associated with high-level concepts such as the configuration or dynamics of a system. A robot lifting its arm or a wheeled robot turning quickly indicate high activity levels

while a legged robot in a sit-down position or a static humanoid suggests a low activity level. To tackle this challenge, we start by investigating how to learn dynamics of a mechanical system from image data.

We propose a neural network approach to learning dynamics from images since neural networks have achieved unprecedented success in computer vision. Our proposed approach simultaneously learns the embedding from the high dimensional image space to a low dimensional configuration space, and the dynamics on the configuration space, which describe the time evolution of the system. We demonstrate the approach on three mechanical systems and show how the learned models can successfully predict the time evolution of the dynamics based on image inputs and control those dynamics to a target configuration based on an image of that target configuration. Moreover, unlike most of black-box neural network models, our model learns interpretable coordinates since we incorporate the geometry of the system into the model. Our model also learns the system energy that is consistent with the physical configuration of the system. For example, a pendulum in a more upright position has a higher learned potential energy. This interpretability allows us to synthesize energy-based controllers to control the system to a target configuration. This methodology can be leveraged by agents to estimate the activity level of their neighbors, and can be extended to learn cascade dynamics from data.

1.2 Related Works

Cascade dynamics and multiplex networks There are two types of models typically used to study a spread of an activity or a cascade in a group: compartmental models and agent-based models. Most of the epidemiological models belong to the first type by assuming the population is well mixed and each individual falls into one of the categories called compartments (e.g., susceptible, infected, recovered) [2, 39]. Agent-based models explicitly model each individual as an agent and the connections among agents. The agents and their connections are usually modelled as a graph or a network. There are some hybrid models; for example, Pagliara and Leonard [67] studied epidemiological models on networks.

A popular class of models studies the spread of an innovation or activity so that each agent has a binary state indicating if it is active or not. The spread of the activity is based on a set of rules. Two most studied models of this kind are the linear threshold models (LTM) and the independent cascade model (ICM) Kempe et al. [45]. Here we briefly survey the LTM, first proposed by Granovetter [31] and Schelling [80]. It describes the spread of an activity, innovation or strategy which an agent

adopts or rejects by comparing the fraction of its neighbors that have adopted the activity with its individual threshold. Kempe et al. [45] studied the LTM with random thresholds and mapped it to a live-edge model (LEM), which does not require temporal iteration, to compute the size of cascade given a group of early adopters. Lim et al. [55] specialized this method to a single early adopter and defined the size of cascade at steady state as the cascade centrality, a metric that measures the ability of the early adopter's ability in terms of spreading information. Acemoglu et al. [1] analyzed the LTM for deterministic thresholds. LTM has been well studied in single-layer networks, see for example in [70, 25, 24, 72]. Yağan and Gligor [94] studied LTM with multiplex networks with a weighted average to synthesize information from different layers. Salehi et al. [75] reviewed various spreading dynamics on multilayer networks. We refer readers to [12] for a review of multiplayer networks. Among the spreading models on multilayer networks, few of them focus on heterogeneous agents.

Networks of robotic systems with Lagrangian/Hamiltonian dynamics Nair et al. [62] studied coordinated control of networked mechanical systems using the method of controlled Lagrangians. Each agent in the network is a mechanical system where the time evolution is governed by Lagrangian dynamics. Nair and Leonard [59] further applied the method to a network of rotating rigid bodies. Stable synchronization of networked robotic systems is studied in [83, 60, 61, 79, 36]. This line of works assume the coordinates of a neighboring system in the configuration space is directly observed. The more realistic scenarios where the coordinates need to be inferred from high-dimensional sensor data have not been studied.

Physics-informed neural network models Physics-informed neural networks incorporate physics priors into deep learning to enhance transparency of the model and improve generalization. Lagrangian and Hamiltonian dynamics are able to model a broad class of physical systems so previous research has explored incorporating Lagrangian or Hamiltonian priors into deep learning. Lutter et al. [56] proposed Deep Lagrangian Network to learn Lagrangian dynamics for rigid-body systems from position, velocity and acceleration data. The model allows online learning and control. Cranmer et al. [20] proposed Lagrangian Neural Network to learn an arbitrary Lagrangian beyond rigid-body systems. Greydanus et al. [33] proposed Hamiltonian Neural Network to learn Hamiltonian dynamics from position, momentum data and their derivatives. Sanchez-Gonzalez et al. [78] proposed Hamiltonian Graph Networks with ODE integrators to learn Hamiltonian dynamics from only position and velocity data. Chen et al. [18] learns symplectic dynamics using symplectic inte-

grators. This line of works require direct observation of low dimensional data in the configuration space or phase space. It is not clear how to infer coordinates in the configuration space from high dimensional data so that the learned coordinates can be used by physics-informed neural networks.

Learning dynamics without supervision Various works explore how to infer a low dimensional latent space from a high dimension image space but the latent space is usually not interpretable and cannot be used as the configuration space required by physics priors. Among these works, Belbute-Peres et al. [8] found that their model is not able to learn meaningful dynamics when no position and velocity data are provided. However, with a little supervision data, the model is able to infer meaningful dynamics. This suggests that useful inductive biases are needed to learn dynamics in a unsupervised way. Various kinds of inductive biases are explored in the literature. Watter et al. [89] and Levine et al. [53] learned locally linear dynamics. Jaques et al. [42] learned unknown parameters of dynamics within a given class. Kossen et al. [48] extracted position and velocity of each object from videos directly. Watters et al. [91] used an object-oriented design to improve data efficiency and robustness in unsupervised learning. Battaglia et al. [6], Sanchez-Gonzalez et al. [76] and Watters et al. [90] incorporated objects and their relations into supervised learning. These object-oriented designs improve learning but they focus little on rotations of objects. Saemundsson et al. [74] proposed Variational Integrator Network which work with rotational coordinates but it cannot model systems with more than one rotational coordinates. As rotational coordinates are common with Lagrangian or Hamiltonian dynamics, previous unsupervised learning approaches do not work well with Lagrangian or Hamiltonian priors. Toth et al. [86] incorporate the Hamiltonian prior but the latent coordinates are not interpretable.

1.3 Contributions

This dissertation contains several separate studies at the intersection of multi-agent systems, dynamical systems, control and machine learning. The contributions of this dissertation are listed as follows:

- We study the spread of an activity by extending the linear threshold model and its correspondent live-edge model from traditional networks to multiplex networks where different communication modalities are distinguished.
- We propose different protocols for an agent to synthesize information from different layers in the network. We study heterogeneous groups where different agents use different protocols.

We show the role of heterogeneity in the trade off between the sensitivity of spreading a real signal and the robustness of spreading a spurious signal.

- We propose two algorithms to compute the size of cascade (influence spread) in the heterogeneous multiplex LTM. The first algorithm accurately computes the influence spread based on heterogeneous protocols and network structure. The second algorithm computes the influence spread approximately by solving a probabilistic inference problem in Bayesian networks.
- We introduce a physics-informed neural network model which learns Hamiltonian dynamics with control from generalized coordinate data and their first-order derivatives. The model narrows the gap between model-based methods and the data-driven methods by the angle-aware design.
- Our proposed physics-informed neural network model learns interpretable energy directly, which allows synthesis of energy-based controllers.
- We design a coordinate-aware VAE to infer generalized coordinates from images, which works with our physics-informed neural network model to learn Lagrangian/Hamiltonian dynamics from images.

1.4 Outline

This dissertation is organized into two parts, where Part II (Chapters 8 to 13) contains four published peer-reviewed papers and two papers that have been submitted for publication.

Part I is organized into six chapters. In Chapter 1, we introduce the main theme of this dissertation and survey related works in the literature. In Chapter 2, we present and define the multiplex LTM, protocols, influence spread and cascade centrality, based on Chapter 8 [100] and Chapter 9 [103]. We prove the equivalence of multiplex LTM to multiplex live-edge model. We map the problem of computing cascade centrality into a probabilistic inference problem in a Bayesian network. We show the problem of computing influence spread in multiplex networks are computationally complex and an approximate algorithm is the best we can achieve for large networks. In Chapter 3, we generalize the LTM to the continuous threshold model (CTM), proposed in Chapter 10 [98]. We study the CTM on a family of networks with three clusters. With bifurcation analysis, we show that high disparity in the network leads to a cascade. In Chapter 4, we study the problem of learning dynamics from trajectory data and summarize results from Chapter 11 [101] and Chapter 12

[102]. We show that the physics-informed neural networks outperform baseline models in terms of prediction accuracy and generalization. In Chapter 5, we assume trajectory data are not given and we need to infer those from images. We present the coordinate-aware VAE proposed in Chapter 13 [99] and demonstration of the framework on three mechanical systems. In Chapter 6, we conclude the dissertation and discuss future directions.

Chapter 2

Cascade Dynamics on Multiplex Networks

This chapter considers the spread of an activity among a group of agents where multiple communication modalities among the agents exist, presented in Part II: Chapters 8 and 9, which appear as Zhong et al. [100] and Zhong et al. [103]. The spread of an activity is referred to as the cascade dynamics, which widely exist from biological systems to engineering systems. This chapter is motivated by the spread of starling behavior in the fish school. The starling behavior is a quick twitch of a fish to flee away from a predator. If a predator is near a fish school, it would startle a few fish in the school and the starling behavior would spread across the group so that the fish that does not directly sense the predator could swim away from the potential threat. Moreover, disturbance and noise exist in the environment where the fish school stays. The fish school seems good at distinguishing a real threat from environmental disturbances. It is known that a fish can sense other fish by its eye and lateral line. While a fish might see another fish in the front, it can also feel others behind with its lateral line. This multiple sensing modalities exist in various kinds of groups. In this chapter, we investigate how multiple communication modalities affect the spread of an activity. We also show how individual preferences along with multiple communication modalities help a group respond to a real threat while not responding to disturbances. We first introduce multiplex network - a framework that allows us to explicitly model different communication modalities.

2.1 Traditional Networks and Multiplex Networks[1]

A network or a graph is usually used to model a multi-agent system. A graph $G = (V, E)$ is a collection of nodes V and edges $E \subseteq V \times V$. Traditionally, we model agents as nodes in the graph and the communications among agents as edges. When there are more than one communication modalities among agents, modelling all communications in a traditional network fails to distinguish different modalities. Multiplex networks are a framework that allows us to distinguish those modalities. A multiplex network $\mathcal{G}$ is a collection of $m \in \mathbb{N}$ directed weighted graphs $G_1, G_2, ..., G_m$. Each graph $G_k = (V, E^k)$, $k = 1, \ldots, m$ is referred to as a layer in the multiplex network. The node set $V = \{1, 2, 3, ..., n\}$ is the same across all layers, representing the same group of agents. The edge set of layer k is $E^k \subseteq V \times V$ and can be different in different layers. Each directed edge $e^k_{i,j} \in E^k$, from i to j in layer k, is associated with a weight $w^k_{i,j} \in \mathbb{R}^+$. Here we adopt the "sensing" convention for edges: edge $e^k_{i,j}$ exists if agent i can sense agent j in layer k and we say that agent j is an out-neighbor of agent i in layer k. We say that the weight of i's out-neighbor j in layer k is the weight $w^k_{i,j}$. We assume the weights of all out-neighbors for an agent are normalized, i.e., $\sum_{j \in N^k_i} w^k_{i,j} = 1$ for every agent i. A traditional network is a multiplex network with $m = 1$, i.e., with only a single layer and we refer to it as a monoplex network.

For undirected graphs, every edge is modeled with two opposing directed edges. For unweighted graphs, every edge $e^k_{i,j}$ is assigned a weight $w^k_{i,j} = 1/d^k_i$, where d^k_i is the out-degree of node i in layer k and equals to the number of out-neighbors of node i in layer k. A *projection network* of $\mathcal{G}$ is the graph $\text{proj}(\mathcal{G}) = (V, E)$ where $E = \cup_{k=1}^m E^k$.

Fig. 2.1 shows an illustration of an example multiplex network with two layers of five agents. Each layer is a unweighted graph. Next, we present the heterogeneous multiplex linear threshold model - the discrete cascade dynamics on multiplex networks.

2.2 The Heterogeneous Multiplex Linear Threshold Model

The linear threshold model (LTM) studies the spread of an activity among a network of agents. Each agent i has a binary state $x_i(t) \in \{0, 1\}$ at a discrete time step t, representing the activity of the agent. Let S_t be the set of agents that are active by the end of iteration t, where S_0 is referred to as the seeds. In a traditional network where all the communication modalities are modelled in a graph, the LTM determines the spread of the activity as follows [45]. Each agent randomly and

[1]Adapted from Chapter 9 [103].

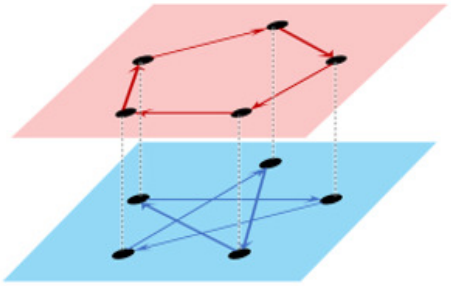

Figure 2.1: An example multiplex network with two layers (duplex network) of five agents. The edges in the red layer and the blue layer are different but the nodes in both layers represent a group of five agents.

independently chooses a threshold $\mu_i \in [0,1]$ from $U(0,1)$. At $t = 0$, all the agents are inactive except seeds, i.e., $(x_i(0) = 0, \forall i \notin S_0)$. At each times step, each agent compare its threshold with the sum of weights of its active out-neighbors. It becomes active if the sum is greater than its threshold. Once the agent is active, it remains active so that $S_{t-1} \subseteq S_t$. For a network of n agents, the steady state is obtained in $t \leq n$.

We extend the LTM to multiplex networks to distinguish different communication modalities. We propose protocols for an agent to synthesize information from different communication modalities. For a multiplex network with m layers, each agent randomly and independently chooses a threshold μ_i^k in each layer from $U(0,1)$. We say the agent i receives a positive input $y_i^k(t) = 1$ from layer k at t if the sum of weights of active out-neighbors of agent i in layer k at $t-1$ exceeds μ_i^k, i.e., $\mu_i^k < \sum_{j \in N_i^k \cap S_{t-1}} w_{i,j}^k$. Otherwise, agent i receives a neutral input. As the neighbors in different layers might be different, the agents need a protocol to synthesize the inputs from different layer and decide whether to become active or not. Let the average input be $y_i(t) = \sum_{k=1}^{m} y_i^k(t)/m$.

Definition 1 (Multiplex LTM Protocol [103])**.** *Given multiplex network $\mathcal{G}$ with seed set S_0, the* multiplex LTM protocol for agent i *is parametrized by $\delta_i \in [1/m, 1]$ as follows:*

$$x_i(0) = 1, \quad \forall i \in S_0 \tag{2.1}$$

$$x_i(0) = 0, \quad \forall i \notin S_0 \tag{2.2}$$

$$x_i(t) = \begin{cases} 1, & \text{if } y_i(t) \geq \delta_i \text{ or } x_i(t-1) = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{2.3}$$

We identify two protocols for the limiting values of δ_i:

Protocol OR*: $\delta_i = 1/m$. Inactive agent i at iteration $t-1$ becomes active at iteration t if it receives a positive*

input from any *layer at* t*;*

Protocol AND*:* $\delta_i = 1$*. Inactive agent* i *at iteration* $t-1$ *becomes active at iteration* t *if it receives positive inputs from* all *layers at* t*.* □

Protocol OR models agents that are readily active since positive input in one layer is sufficient for the agent to become active. Protocol AND models agents that are conservatively activated, since positive inputs from all layers are required for the agent to become active. We examine heterogeneous agents where different agents could choose different limiting protocols.

Definition 2 (Sequence of Protocols [103])**.** *Let* $u_i \in \{OR, AND\}$ *be the protocol used by agent* i*. We define the* sequence of protocols $\mathcal{U} = (u_1, u_2, ..., u_n)$ *to be the protocols used by the* n *agents ordered from agent 1 to agent* n*.*

Similar to the LTM in the traditional network, the steady state is reached by at most n time steps.

2.3 The Heterogeneous Multiplex Live-edge Model

The LTM in traditional networks has been studied by mapping it into the live-edge model (LEM) [45], so that we can study the LTM without temporal iteration and performing simulation. We present the LEM in traditional networks (quoted from [103]):

> The LEM for a monoplex network is defined as follows [45]. Let S_0 be the set of seeds. Each unseeded agent randomly selects one of its outgoing edges with probability given by the edge weight. The selected edge is labeled as "live", while the unselected edges are labeled as "blocked". The seeds block all of their outgoing edges. Every directed edge will thus be either live or blocked. The choice of edges that are live is called a *selection of live edges*.
>
> Let L be the set of all possible selections of live edges. The probability q_l of selection $l \in L$ is the product of the weights of the live edges in selection l. Because the selection of live edges can be done at the same time for every node, the LEM can be viewed as a static model. The LEM can alternatively be viewed as an iterative process in the case the live edges are selected sequentially.
>
> A *live-edge path* [45] is a directed path that consists only of live edges. Let $\mathcal{L}_{ij}$ be the set of all possible distinct live-edge paths from agent $i \notin S_0$ to $j \in S_0$. The probability r_α of live-edge path $\alpha \in \mathcal{L}_{ij}$ is the product of the edge weights along the path. We say $i \notin S_0$

is *reachable from* $j \in S_0$ *by live-edge path* α *with probability* r_α, and $i \notin S_0$ is *reachable from* $j \in S_0$ *with probability* r_{ij}, where $r_{ij} = \sum_{\alpha \in \mathcal{L}_{ij}} r_\alpha$.

Alternatively, we can compute r_{ij} in terms of selections of live-edges. Let $L_{ij} \subseteq L$ be the set of all selections of live edges that contain a live-edge path from $i \notin S_0$ to $j \in S_0$. Then, $r_{ij} = \sum_{l \in L_{ij}} q_l$. Likewise, let $L_{iS_0} \subseteq L$ be the set of all selections of live edges that contain a live-edge path from $i \notin S_0$ to at least one node $j \in S_0$. Then, $i \notin S_0$ is *reachable from* S_0 *with probability* r_{iS_0}, where $r_{iS_0} = \sum_{l \in L_{iS_0}} q_l$.

We extend the LEM into multiplex networks so that we can analyze multiplex LTM without temporal iteration. We can also compute the size of cascade by leveraging network structures.

Definition 3 (Multiplex LEM [103])**.** *Consider a multiplex network* $\mathcal{G}$ *with seed set* S_0*. In each layer* k*, each unseeded agent* i *randomly selects one of its outgoing edges* e^k_{i,j_k} *with probability* w^k_{i,j_k}*. The selected edges are labeled as "live", while the unselected edges are labeled as "blocked". The seeds block all of their outgoing edges in every layer. The choice of edges that are live is a* multiplex selection of live edges*. Let* L *be the set of all possible multiplex selections of live edges. The probability* q_l *of selection* $l \in L$ *is the product of the weights of all live edges in selection* l*.*

A key concept in LTM is reachability. In traditional networks, reachability is defined based on live-edge paths. Here we generalize live-edge paths to live-edge trees in multiplex networks.

Definition 4 (Live-edge Tree [103])**.** [2] *Given a set of seeds* S_0 *and a multiplex selection of live edges* $l \in L$*, the* live-edge tree T^l_i *associated with agent* $i \notin S_0$ *is constructed as follows with agent* i *as the root node. Let* e^k_{i,j_k} *be the live edge of agent* i *in layer* k*,* $k = 1, \ldots, m$*. Then the children of the root node are agents* $j_1, j_2, \ldots, j_m$*, and the root node is connected to each child with the live edge in the corresponding layer. The tree is constructed recursively in this way for each child that itself has at least one child. Any agent in the network may appear multiple times as a node in the tree.*

Fig. 2.2 shows an illustrative example of a multiplex network with three layers and five agents. In this simple example, there is only one possible live-edge selection, shown in Fig. 2.3. Fig. 2.4 shows the live-edge tree associated with agent 5. Now we are ready to formally define reachability.

Definition 5 ($\mathcal{U}$-Reachability [103])**.** *Consider multiplex network* $\mathcal{G}$ *with seed set* S_0 *and multiplex selection of live edges* $l \in L$*. Let* T^l_i *be the live-edge tree associated with agent* $i \notin S_0$*. Suppose there are*

[2] To highlight key differences between multiplex and monoplex networks, we assume each $i \notin S_0$ has at least one neighbor in each layer. If not, with a slight modification of Defs. 4- 5, the theory and computation are still valid.

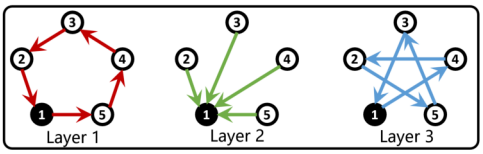

Figure 2.2: An example of a three-layer multiplex network with five agents. Agent 1 is the seed, which is denoted by the black circle. Repeated from Fig. 9.1 [103].

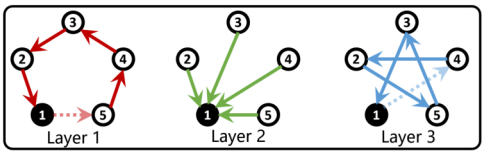

Figure 2.3: The unique multiplex selection of live edges for the network in Fig. 2.2. Repeated from Fig. 9.2 [103].

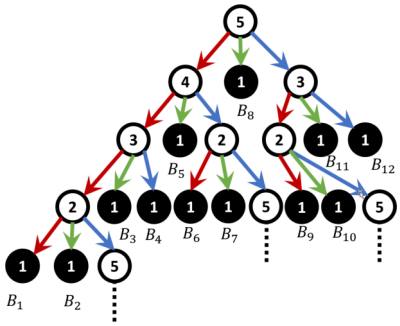

Figure 2.4: The live-edge tree associated with agent 5 for the example three-layer multiplex network of Fig. 2.2 and the unique selection of live edges of Fig. 2.3. $\mathcal{B}_5 = \{B_1, B_2, ..., B_{12}\}$ is the set of distinct branches that end with a seed. Repeated from Fig. 9.3 [103].

b distinct branches in T_i^l indexed by $\beta = 1, \ldots, b$ and of the form: $B_\beta = (i, e_{i,i_1}^{k_0}, i_1, e_{i_1,i_2}^{k_1}, i_2, ..., i_s)$, where $i_j \in V$, $j = 1, \ldots, s$, $i_s \in S_0$, and each agent in V appears at most once in B_β. We call each B_β a distinct branch that ends in a seed. *Denote the set of these branches as $\mathcal{B}_i^l = \{B_1, B_2, ..., B_b\}$. For any subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$, let the set of agents in $\hat{\mathcal{B}}$ be $\hat{V}$ and the set of edges in $\hat{\mathcal{B}}$ be $\hat{E}$.*

*Given a sequence of protocols $\mathcal{U}$, we say that branch subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$ is $\mathcal{U}$-*feasible for *i if $\hat{\mathcal{B}} \neq \emptyset$ and for every $\hat{i} \in \hat{V} \setminus S_0$ for which $u_{\hat{i}} =$* AND*, all of $\hat{i}$'s live edges belong to $\hat{E}$. Then, i is $\mathcal{U}$-*reachable from S_0 by the selection of live edges l with probability q_l *if there exists at least one $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$ that is $\mathcal{U}$-feasible for i. Let $L_{iS_0}^{\mathcal{U}} \subseteq L$ be the set of all selections of live edges by which i is $\mathcal{U}$-reachable from S_0. Then, i is $\mathcal{U}$-*reachable from S_0 with probability $r_{iS_0}^{\mathcal{U}}$, *where $r_{iS_0}^{\mathcal{U}} = \sum_{l \in L_{iS_0}^{\mathcal{U}}} q_l$.* □

We will connect the $\mathcal{U}$-reachability to the LTM in the next section, but now let us look at an example of $\mathcal{U}$-reachability (quoted from [103]):

> To illustrate $\mathcal{U}$-reachability, consider the live-edge tree associated with agent 5 in Fig. 2.4 for the unique selection of live edges in Fig. 2.3 for the multiplex network of Fig. 2.2 with seed set $S_0 = \{1\}$. Because the selection of live edges in Fig. 2.3 is unique, it is chosen with probability $q = 1$. Therefore, agent $i \notin S_0$ is $\mathcal{U}$-reachable from S_0 with probability 1 if there exists at least one $\hat{\mathcal{B}} \subseteq \mathcal{B}_i$ that is $\mathcal{U}$-feasible for i. For agent 5, there are 12 distinct branches that end in a seed, as shown in Fig. 2.4; thus, $\mathcal{B}_5 = \{B_1, B_2, ..., B_{12}\}$. For example, $B_8 = (5, e_{5,1}^2, 1)$.
>
> We compute $\mathcal{U}$-reachability from S_0 for agent 5 for each the following three sequences of protocols used by the five agents in the three-layer multiplex network:

$$\mathcal{U}_1 = (\text{OR}, \text{AND}, \text{AND}, \text{AND}, \text{OR}) \tag{2.4}$$

$$\mathcal{U}_2 = (\text{OR}, \text{OR}, \text{AND}, \text{AND}, \text{AND}) \tag{2.5}$$

$$\mathcal{U}_3 = (\text{OR}, \text{AND}, \text{AND}, \text{AND}, \text{AND}). \tag{2.6}$$

1. Let $\mathcal{U} = \mathcal{U}_1$. Consider $\hat{\mathcal{B}} = \{B_8\} \subset \mathcal{B}_5$. Then, $\hat{V} = \{1, 5\}$ and $\hat{E} = \{e_{5,1}^2\}$. Since 5 is the only unseeded node in $\hat{V}$ and $u_5 =$ OR, $\hat{\mathcal{B}}$ is $\mathcal{U}$-feasible for 5. Thus, agent 5 is $\mathcal{U}$-reachable from S_0 with probability 1.
2. Let $\mathcal{U} = \mathcal{U}_2$. Consider $\hat{\mathcal{B}} = \mathcal{B}_5$. Then, $\hat{V} = \{1, 2, 3, 4, 5\}$. The unseeded nodes $j \in \hat{V}$ for which $u_j =$ AND are $j = 3, 4, 5$. From Fig. 2.4, observe that all the live edges of nodes 3, 4, and 5, belong to $\hat{E}$, the edge set of $\hat{\mathcal{B}} = \mathcal{B}_5$. Thus, $\hat{\mathcal{B}}$ is $\mathcal{U}$-feasible for 5, and agent 5 is $\mathcal{U}$-reachable from S_0 with probability 1.

3. Let $\mathcal{U} = \mathcal{U}_3$. In this case there is no $\mathcal{U}$-feasible subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_5$, since u_2 = AND and agent 2 has a live edge $e^3_{2,5}$, which is not in the edge set of any branch in $\mathcal{B}_5$. Thus, agent 5 is not $\mathcal{U}$-reachable from S_0.

2.4 Equivalence

The benefit of LEM is that it is proved to be equivalent to the LTM in single-layer networks, so that it can be used to compute the size of cascade based on network structure. We present the equivalence using our notation.

Lemma 1 ([45]). *For a given monoplex network $\mathcal{G}$ with seed set S_0, the probabilities of the following two events for arbitrary agent $i \notin S_0$ are the same:*

1. *i is active at steady state for the LTM with random thresholds and initial active set S_0;*
2. *i is reachable from set S_0 under the random selection of live edges in the LEM.*

In this section, we generalize this equivalence from single-layer networks to multiplex networks, so that we can leverage the LEM to compute the size of cascade in the heterogeneous multiplex LTM. The key in proving the equivalence is to treat the LEM as an iterative model. The following lemma sheds light on how to do this.

Lemma 2 ([103]). *Given a multiplex network $\mathcal{G}$ with seed set S_0, multiplex selection of live edges $l \in L$ and sequence of protocols $\mathcal{U}$, consider agent $i \notin S_0$ and its associated live-edge tree T_i^l. Assume i's live edge in layer k connects to agent i_1^k, $k = 1, \ldots, m$. Then the $\mathcal{U}$-reachability of i from S_0 by selection l can be inferred from the reachability of its children i_1^k and its protocol u_i as follows:*

1. *Let u_i = OR. Then, i is $\mathcal{U}$-reachable from S_0 by selection of live edges l if and only if at least one child i_1^k is $\mathcal{U}$-reachable from S_0 by selection of live edges l.*
2. *Let u_i = AND. Then, i is $\mathcal{U}$-reachable from S_0 by selection of live edges l if and only if every child i_1^k is $\mathcal{U}$-reachable from S_0 by selection of live edges l.*

Using Lemma 2, we can reveal $\mathcal{U}$-reachability of agents iteratively. Starting from $S'_0 = S_0$, we determine the $\mathcal{U}$-reachability of one-hop neighbors of S'_0. Those one-hop neighbors that are determined to be $\mathcal{U}$-reachable are added to S'_0 to form S'_1. In this way, we get a series of reachable sets $S'_0, S'_1, S'_2, \ldots$. The iterations end at t if $S'_t = S'_{t-1}$. Leveraging this insight, we are able to prove the equivalence.

Theorem 1 (Equivalence of multiplex LTM and multiplex LEM [103]). *For a multiplex network $\mathcal{G}$ with seed set S_0, multiplex selection of live edges $l \in L$ and sequence of protocols $\mathcal{U}$, the probabilities of the following two events regarding an arbitrary agent $i \notin S_0$ are the same:*

1. *i is active at steady state for the multiplex LTM under $\mathcal{U}$ with random thresholds and initial active set S_0;*

2. *i is $\mathcal{U}$-reachable from the set S_0 under random selection of live edges in the multiplex LEM.*

By leveraging the equivalence, we are able to compute the an agent's probability of becoming active and the size of cascade in LTM using network structure instead of temporal iterations. In the next section, we formally define the size of cascade.

2.5 Multiplex Influence Spread

We study LTM with random thresholds. With different set of thresholds, the outcomes of the spread are different. The expected number of active agents at steady state given a single-layer network G and seed set S_0 is defined as the influence spread $\sigma^G_{S_0}$ [45]. When the seed set contains only one agent j, the influence is specialized to cascade centrality of j by Lim et al. [55], denoted as $C^G_j = \sigma^G_j$. Both influence spread and cascade centrality can be defined in multiplex networks as follows.

Definition 6 (Multiplex influence spread [103]). *The* multiplex influence spread of agents in S_0, *denoted $\sigma^{\mathcal{G},\mathcal{U}}_{S_0}$, is defined as the expected number of active agents at steady state for the multiplex LTM given the network $\mathcal{G}$, sequence of protocols $\mathcal{U}$, and initial active set S_0. Let $\mathbb{E}^{\mathcal{G},\mathcal{U}}_{S_0}$ and $\mathbb{P}^{\mathcal{G},\mathcal{U}}_{S_0}$ be expected value and probability, respectively, conditioned on $\mathcal{G}, \mathcal{U}, S_0$. Then*

$$\sigma^{\mathcal{G},\mathcal{U}}_{S_0} = \mathbb{E}^{\mathcal{G},\mathcal{U}}_{S_0}\Big(\sum_{i=1}^{n} \bar{x}_i\Big) = \sum_{i=1}^{n} \mathbb{P}^{\mathcal{G},\mathcal{U}}_{S_0}(\bar{x}_i = 1). \tag{2.7}$$

Definition 7 (Multiplex cascade centrality [103]). *The* multiplex cascade centrality of agent j, *denoted $C^{\mathcal{G},\mathcal{U}}_j$, is defined as*

$$C^{\mathcal{G},\mathcal{U}}_j = \sigma^{\mathcal{G},\mathcal{U}}_j. \tag{2.8}$$

From the equivalence of multiplex LTM and multiplex LEM (Theorem 1), we can compute the multiplex influence spread and cascade centrality by leveraging the LEM.

Corollary 1 ([103]). *Given multiplex network $\mathcal{G}$ and sequence of protocols $\mathcal{U}$, multiplex influence spread*

of agents in S_0 and multiplex cascade centrality of agent j can be determined as

$$\sigma_{S_0}^{\mathcal{U}} = \sum_{i=1}^{n} r_{iS_0}^{\mathcal{U}}, \quad C_j^{\mathcal{U}} = \sum_{i=1}^{n} r_{ij}^{\mathcal{U}}. \tag{2.9}$$

We formalize the procedure of accurately compute multiplex influence spread in the following algorithm.

Algorithm 1 (Compute multiplex influence spread $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$ [103])**.** *Given multiplex network $\mathcal{G}$ and sequence of protocols $\mathcal{U}$:*

1. *Find the set L of all possible selections of live edges for multiplex network $\mathcal{G}$ and initially active set S_0. Calculate the probability q_l of each $l \in L$.*
2. *For each agent i find $L_{iS_0}^{\mathcal{U}} \subseteq L$, the set of all $l \in L$ such that i is $\mathcal{U}$-reachable from S_0 by selection l.*
3. *Calculate $r_{iS_0}^{\mathcal{U}} = \sum_{l \in L_{iS_0}^{\mathcal{U}}} q_l$.*
4. *Calculate $\sigma_{S_0}^{\mathcal{G},\mathcal{U}} = \sum_{i=1}^{n} r_{iS_0}^{\mathcal{U}}$.*

2.6 Examples - Calculate Influence Spread Accurately

We present some examples of leveraging Algorithm 1 to compute cascade centrality. We show how distinguishing multiple communication modalities influence spread and how heterogeneity plays a role in distinguishing a real threat from disturbances.

2.6.1 Homogeneous Agents

We show analytical results of two families of homogeneous networks. The special structure of the networks enable us to write down analytical expressions of influence spread based on Algorithm 1. The first family is the duplex repeated path network $\mathcal{G}_R$, illustrated in Fig. 2.5.

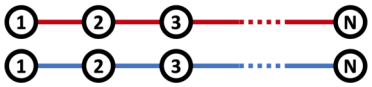

Figure 2.5: Duplex repeated path network $\mathcal{G}_R$ has path graph G_{Pa} as each layer. Repeated from Fig. 9.5 [103].

The projection network of it is a path graph $G_{Pa} = \text{proj}(\mathcal{G}_R)$. The cascade centrality of an agent in the network is stated in the following proposition.

Proposition 1 (Multiplex cascade centrality for $\mathcal{G}_R$ [103])**.** *Consider the monoplex path network G_{Pa} and duplex repeated path network $\mathcal{G}_R$ for N agents with $u_i = u \in \{OR, AND\}$. Then*

$$C_j^{G_{Pa}} = h_j(.5), \quad C_j^{\mathcal{G}_R,\text{OR}} = h_j(.75), \quad C_j^{\mathcal{G}_R,\text{AND}} = h_j(.25),$$

$$h_j(p_0) = \begin{cases} \sum_{l=0}^{N-2} p_0^l + p_0^{N-2}, & j \in \{1, N\} \\ 1 + \sum_{l=0}^{N-3} p_0^l + p_0^{N-3}, & j \in \{2, N-1\} \\ \sum_{l=0}^{j-1} p_0^l + p_0^{j-1} + \sum_{l=1}^{N-j-1} p_0^l + p_0^{N-j-1}, & \text{o.w.} \end{cases}$$

Moreover,

$$C_j^{\mathcal{G}_R,\text{OR}} > C_j^{G_{Pa}} > C_j^{\mathcal{G}_R,\text{AND}}.$$

The results reveal the role of distinguishing communication modalities. A homogeneous group with Protocol OR (AND), produce a larger (smaller) size of cascade as compared to projecting the communication modalities into a traditional graph. The results hold for arbitrary number of agents.

We then examine the duplex permutation networks $\mathcal{G}_P$, illustrated in Fig. 2.6.

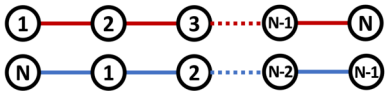

Figure 2.6: Duplex permutation network $\mathcal{G}_P$. Repeated from 9.6 [103].

Different from the duplex path networks, the projection network of the duplex permutation networks are cycle graphs G_C. As the analytical expression of cascade centrality is messy and not insightful in this case, we show the probability of an agent becoming active given another agent as the seed in a homogeneous network, i.e., $\mathbb{P}_j^{\mathcal{G}_P,u}(\bar{x}_i = 1)$, $u \in \{\text{OR}, \text{AND}\}$.

Proposition 2 (Probabilities for multiplex cascade centrality for $\mathcal{G}_P$ [103])**.** *Consider the duplex permutation network $\mathcal{G}_P$ for N agents with $u_i = u \in \{OR, AND\}$ and the cyclic network $G_C = proj(\mathcal{G}_P)$. Then*

$$\begin{aligned} \mathbb{P}_j^{\mathcal{G}_P,\text{OR}}(\bar{x}_i = 1) &= (.75)^{|i-j|} + .5(.75)^{N-|i-j|-3} - .5(.75)^{N-5} \\ \mathbb{P}_j^{\mathcal{G}_P,\text{AND}}(\bar{x}_i = 1) &= (.25)^{|i-j|} \\ \mathbb{P}_j^{G_C}(\bar{x}_i = 1) &= (.5)^{|i-j|} + (.5)^{N-|i-j|} \end{aligned} \tag{2.10}$$

where $3 \leq i \leq N-3$ *and* $j = 2, \ldots, i-2, i+2, \ldots, N-2$. *Moreover,*

$$C_j^{\mathcal{G}_P,\mathrm{OR}} > C_j^{G_C} > C_j^{\mathcal{G}_P,\mathrm{AND}}$$

$$\mathbb{P}_j^{\mathcal{G}_P,\mathrm{AND}}(\bar{x}_i = 1) = \mathbb{P}_j^{\mathcal{G}_R,\mathrm{AND}}(\bar{x}_i = 1) = (0.25)^{|i-j|}$$

$$\mathbb{P}_j^{\mathcal{G}_P,\mathrm{OR}}(\bar{x}_i = 1) > \mathbb{P}_j^{\mathcal{G}_R,\mathrm{OR}}(\bar{x}_i = 1).$$

As we can see from the results, a homogeneous duplex permutation network with Protocol AND behaves the same as a duplex path network. This is because when u_N = AND, u_N is always the last agent to be activated. Information can only be spread from j to i along the path on G_C that does not contain agent N. This effectively turns a homogeneous duplex permutation network with Protocol AND into a duplex path network. In a homogeneous duplex permutation network with Protocol OR, information can be spread through either path between j and i. This enhances the spread of the activity as compared to the duplex path networks.

2.6.2 Heterogeneous Agents

After knowing the roles of homogeneous Protocol OR and AND, we examine the role of heterogeneity in protocols in this section. We use the two-layer multiplex network illustrated in Fig. 2.7.

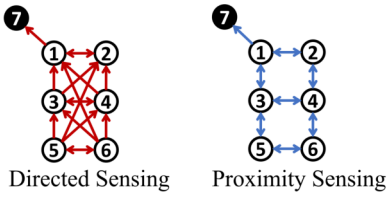

Figure 2.7: Duplex network with agents 1 to 6, layer 1 (red), and layer 2 (blue). Node 7, the external signal, is real since it appears in both layers. Repeated from Fig. 9.7 [103].

The network contains 6 agents (denoted as 1 to 6) and a seventh node that represents an external signal. We are particularly interested in the tension between sensitivity of a spread to a real external signal and the robustness of the cascade to disturbances. We model a real external signal by presenting node 7 in both layers and model a disturbance or noise by presenting node 7 in either layers but not both. We assume only one agent can sense the signal and the edge pointing to the signal has a weight of 1, since the first agent that sense the signal pays all its attention to detect if it is a real one or a spurious one. We also assume each of the 6 agents can sense the signal with equal

chance since in the real environment, since the threat can come from any direction. The graphs in layers represent different sensing modalities. The red layer presents directed sensing, where each agent can only sense others in the front or on the side. The blue layer presents proximity sensing, where each agent can only sense its closest neighbors. This network structure is motivated by a robot team equipped with different types of sensors and fish schools with visual sensing and lateral line sensing. Each agent can use either protocol, and there are 64 different sequences of protocols possible. We propose a utility function Q as a function of $\mathcal{U}$ and a design parameter c to evaluate the benefit of the group from responding to a real signal and not responding to a disturbance. The design parameter c captures the tradeoff between sensitivity to a real input and robustness to a disturbance. We propose the utility function as follows (quoted from [103]),

$$Q(\mathcal{U}, c) = \frac{1}{6}\sum_{l=1}^{6}\left(C_7^{\mathcal{G}^l_{\text{real}},\mathcal{U}} - c\frac{1}{2}(C_7^{\mathcal{G}^l_{\text{spur1}},\mathcal{U}} + C_7^{\mathcal{G}^l_{\text{spur2}},\mathcal{U}})\right). \tag{2.11}$$

Superscript l indexes the agent sensing node 7. Subscripts “real", “spur1" and “spur2" index the networks where node 7 appears in both layers, layer 1 only, and layer 2 only, respectively. Increasing c increases cost of response to spurious signals relative to benefit of response to real signals. Given c, the optimal sequence of protocols is $\mathcal{U}^c = \text{argmax}_{\mathcal{U}} Q(\mathcal{U}, c)$.

Using Algorithm 1, we can calculate the utility given c and $\mathcal{U}$. We can then get the optimal configuration for a given c. Fig. 2.8 shows the optimal configuration for c ranges from 0 to 3. With a low value of c, the disturbance rejection is not taken into account in the utility function, so that the optimal configuration is that all six agents are vigilant, choosing Protocol OR. With a high value of c, the utility function cares more about disturbance rejection then actually responding to a real input, so all the agents in the optimal configuration choose Protocol AND. When c is in between, we see from Fig. 2.8 how the optimal configuration changes from all ORs to all ANDs.

2.7 Computation Complexity of Influence Spread

After knowing how Algorithm 1 can help us gain insight into the role of multiple communication modalities and the role of heterogeneity, we find that Algorithm 1 is slow on large networks, which indicates calculating influence spread accurately is computationally complex, shown in the following theorem.

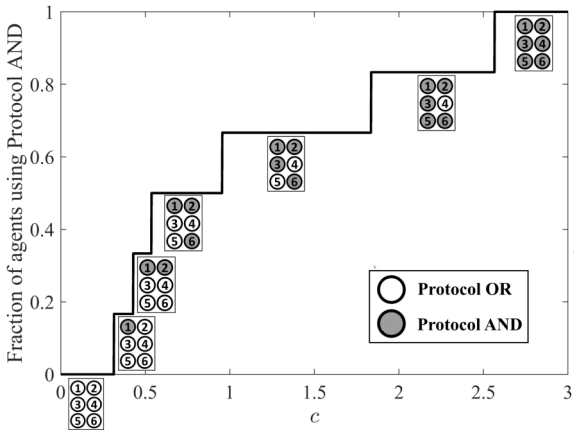

Figure 2.8: The optimal fraction of agents using Protocol AND with illustration of optimal solution $\mathcal{U}^c$ as c varies from 0 to 3. Symmetry is implied. Repeated from Fig. 9.8 [103].

Theorem 2 ([103])**.** *Consider a multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG, with seed set S_0 and sequence of protocols $\mathcal{U}$. Computing $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$ for the multiplex LTM is #P-complete.*

The theorem says that computing influence spread in general is computationally complex and an accurate algorithm is not practical for large networks. In the next section, we propose an approximate algorithm of computing influence spread by leveraging probabilistic inference on Bayesian networks, so that when the network is large, we can approximately calculate influence spread.

2.8 Multiplex LTM as a Bayesian Network

A Bayesian network is a graphical model that represents a set of variables and conditional dependencies among the variables. Probabilistic inference in a Bayesian network is to compute the marginal probabilities variables. By converting a multiplex LTM to a Bayesian network, we can calculate multiplex cascade centrality by leveraging probabilistic inference in Bayesian networks. First, we formally define a Bayesian network.

Definition 8 (Bayesian network [103])**.** *Let $G = (V, E)$, where $V = 1, 2, ..., n$ and $E \subset V \times V$, be a directed acyclic graph (DAG). Each node $i \in V$ is associated with a random variable $x_i' \in \mathcal{X}_i'$. Denote the set of out-neighbors of $i \in V$ as N_i'. Let $\mathbb{P}(x_i'|x_{N_i'}')$ be the probability of x_i' conditioned on the states of nodes in N_i'. Then G is a* Bayesian network *if the joint distribution of the random variables is factorized into conditional probabilities: $\mathbb{P}(x_1', x_2', ..., x_n') = \prod_{i=1}^{n} \mathbb{P}(x_i'|x_{N_i'}')$.*

The belief propagation (BP) algorithm proposed by Pearl [68] is a messsage-passing algorithm to solve probabilistic inference in Bayesian networks. Pearl [68] showed that the algorithm is exact on

trees and polytrees, but suffers from convergence issues when applied to DAGs with loops. Murphy et al. [58] investigated the convergence issue and found that it provides a good approximation when it converges. The application of Pearl's algorithm on DAGs with loops is called loopy belief propagation (LBP). For general graph structures, the junction tree algorithm can perform exact inference by first turning the graph into a junction tree and then applying belief propagation on the modified graph.

We focus on a class of multiplex networks where the projection networks are DAGs. We show that the joint probability of agents being active can be naturally expressed as a Bayesian network.

Algorithm 2 (Bayesian network from multiplex LTM [103])**.** *Given multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG and sequence of protocols $\boldsymbol{\mathcal{U}}$:*

1. *Let G = proj($\mathcal{G}$) be the underlying DAG for the Bayesian network. Then $N_i' = N_i = \cup_{k=1}^{m} N_i^k$.*
2. *Let the random variable x_i' of node i in the Bayesian network be $\bar{x}_i$, the steady-state value of agent i for the multiplex LTM on $\mathcal{G}$. Then $x_i' \in \mathcal{X}_i' = \{0, 1\}$.*
3. *Construct the conditional probabilities for the Bayesian network in terms of the conditional probabilities for the multiplex LTM: $\mathbb{P}(x_i'|x_{N_i}') = \mathbb{P}^{\mathcal{G},u_i}(\bar{x}_i|\bar{x}_{N_i})$.*

We use the multiplex network in Fig. 2.9 as an illustrative example of using Algorithm 2 to turn a multiplex LTM into a Bayesian network (quoted from [103]):

> Since all random variables are discrete, the conditional probability $\mathbb{P}(x_i'|x_{N_i}') = \mathbb{P}^{\mathcal{G},u_i}(\bar{x}_i|\bar{x}_{N_i})$ can be fully described with a conditional probability table (CPT). We show how to construct a CPT for $\mathbb{P}^{\mathcal{G},u_i}(\bar{x}_i|\bar{x}_{N_i})$ for the Fig. 2.9 example. The CPT of i has $2^{|N_i|}$ rows. For agent 6, $N_6 = \{3, 4, 5\}$, $\bar{x}_{N_6} = \{\bar{x}_3, \bar{x}_4, \bar{x}_5\}$ and its CPT has $2^{|N_6|} = 8$ rows. The CPT provides $\mathbb{P}^{\mathcal{G},u_6}(\bar{x}_6 = 0|\bar{x}_3, \bar{x}_4, \bar{x}_5)$ and $\mathbb{P}^{\mathcal{G},u_6}(\bar{x}_6 = 1|\bar{x}_3, \bar{x}_4, \bar{x}_5)$. Table 2.1 and Table 2.2 are the CPTs for agent 6 when it uses Protocol OR and Protocol AND, respectively.

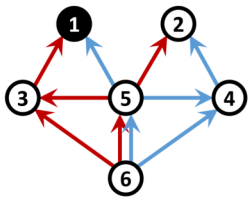

Figure 2.9: Multiplex network $\mathcal{G}_s$ with two unweighted layers and six agents. Red (blue) arrows represent edges in layer 1 (layer 2). $S_0 = \{1\}$. Repeated from Fig. 9.4 [103].

Table 2.1: CPT of agent 6 with u_6 = OR for $\mathcal{G}_s$ of Fig. 2.9. Repeated from 9.1.

$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$	$\mathbb{P}^{\text{OR}}(\bar{x}_6=0\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$	$\mathbb{P}^{\text{OR}}(\bar{x}_6=1\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$
0	0	0	1.00	0.00
0	0	1	0.25	0.75
0	1	0	0.50	0.50
0	1	1	0.00	1.00
1	0	0	0.50	0.50
1	0	1	0.00	1.00
1	1	0	0.25	0.75
1	1	1	0.00	1.00

Table 2.2: CPT of agent 6 with u_6 = AND for $\mathcal{G}_s$ of Fig. 2.9. Repeated from Table 9.2.

$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$	$\mathbb{P}^{\text{AND}}(\bar{x}_6=0\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$	$\mathbb{P}^{\text{AND}}(\bar{x}_6=1\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$
0	0	0	1.00	0.00
0	0	1	0.75	0.25
0	1	0	1.00	0.00
0	1	1	0.50	0.50
1	0	0	1.00	0.00
1	0	1	0.50	0.50
1	1	0	0.75	0.25
1	1	1	0.00	1.00

Algorithm 2 handles the case where the projection network is a DAG. A general multiplex network can be handled by the junction tree algorithm. The probability of each agent being active, however, needs to be obtained by a further marginalization. In the next theorem, we show that the marginal probability is indeed what we need to calculate influence spread.

Theorem 3 ([103]). *Given a multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG, with seed set S_0 and sequence of protocols $\mathcal{U}$, the following two probabilities are the same:*

1. $\mathbb{P}_{S_0}^{\mathcal{G},\mathcal{U}}(\bar{x}_i = 1)$, *the probability that agent i is active at steady state for the multiplex LTM.*
2. $\mathbb{P}^{\mathcal{G},\mathcal{U}}(\bar{x}_i = 1|\bar{x}_j = 1, \bar{x}_l = 0, j \in S_0, l \notin S_0, N_l = \varnothing)$, *the marginal probability of node i in the corresponding Bayesian network of Algorithm 5, conditioned on observed nodes in the seed set and those not in the seed set that have no out-neighbors.*

The next corollary directly follows from Theorem 3, which says we can compute influence spread by probabilistic inference on Bayesian networks, which can be solved by BP algorithms.

Corollary 2 ([103]). *Given a multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG, with seed set S_0 and sequence*

of protocols $\mathcal{U}$, multiplex influence spread $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$ can be computed as

$$\sum_{i=1}^{n} \mathbb{P}^{\mathcal{G},\mathcal{U}}(\bar{x}_i = 1 | \bar{x}_j = 1, \bar{x}_l = 0, j \in S_0, l \notin S_0, N_l = \varnothing).$$

2.9 Examples - Calculate Influence Spread Approximately

We examine multiplex cascade centrality for random multiplex networks with 20 agents. The network has two layers and the projection network is a DAG. We fix a topological order of nodes and generate edges randomly with probability p_e. A lower p_e generates a less connected network and a higher p_e generates a more connected network. We are interested in the cascade centrality of the root node with difference protocol configurations. We study the homogeneous groups with all ORs and all ANDs as well as the heterogeneous group where the agents randomly and independently choose Protocol OR or AND with equal chance.

We change p_e from 0 to 1 and with each p_e, we generate 400 random networks and plot the average cascade centrality over the networks of the three protocol configurations using the methods in Section 2.8. Fig. 2.10 shows how the cascade centrality changes with p_e. When p_e is close to 0, the network is disconnected and the root node can trigger no other agents to be active, regardless of the protocols. Thus, the cascade centrality goes to 1. When p_e is close to 1, the root node triggers every other agent regardless of the protocols. Thus, the cascade centrality goes to 20. When p_e is in the middle. We observe a gap among the three protocol configurations.

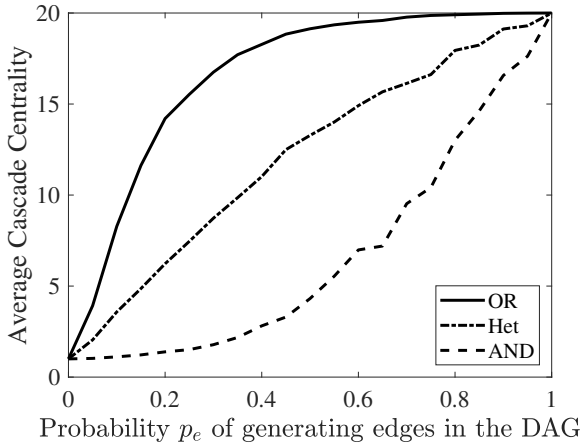

Figure 2.10: Multiplex cascade centrality of root node, averaged over 400 networks, as a function of probability p_e of edges in the DAG. Repeated from Fig. 9.9 [103].

In this chapter we derived the LTM in multiplex networks and analyzed two protocols which

models responsive and conservative agents. We proved the equivalence of the LTM and the LEM with the proposed $\mathcal{U}$-reachability. By leveraging the LEM, we are able to calculate the probability of agents becoming active based on network structure instead of conducting simulations with the LTM. This, in turn, enables the calculation of size of spread. We generalized influence spread and cascade centrality into multiplex networks. We showed analytical results of cascade centrality in two homogeneous networks with symmetry. We showed that Protocol OR enhances a cascade and Protocol AND diminishes a cascade, which is consistent with our intuition about the protocols. The analytical results also showed that distinguishing multiple communication modalities gives us different results than projecting all the communication modalities into a traditional graph. We also studied a two-layer multiplex network with 6 heterogeneous agents, motivated by fish schools with directed sensing and proximity sensing. We showed that heterogeneity in thresholds plays an important role in the tradeoff between sensitivity to a real signal and robustness to disturbances. Our model then provides an explanation of how an animal group could distinguish real signals from disturbances. By showing that the problem of computing influence spread is computationally complex, we know Algorithm 1 is not practical for large networks. For large networks, we proposed to leverage probabilistic inference to calculate the influence spread. We proposed an algorithm to turn a problem of calculating influence spread in a multiplex network, of which the projection network is a DAG, into a problem of probabilistic inference problem in Bayesian networks. Then we are able to leverage algorithms such as loopy belief propagation to calculate the influence spread. One limitation of the method is that it requires a special structure of the multiplex network. An efficient algorithm for general multiplex networks are underexplored and could be pursued in the future.

The main model in this chapter is the LTM - a discrete dynamical model. Although it is nice to develop the LEM to analyze the LTM, the tools in the field of discrete dynamical models are limited. We would like to further investigate cascade dynamics with continuous dynamical models and leverage the tools from continuous dynamical system to help us understand the role of network structure and heterogeneity in cascade. We introduce one of such models in the next chapter.

Chapter 3

Continuous Cascade Dynamics

The LTM presented in the previous chapter is studied with random thresholds and the LTM with deterministic thresholds lacks analytical tractability. Moreover, the LTM is a discrete dynamics and how fast an agent responds to an external signal is not captured. In biological systems, the response to a threat is fast. For example, in the presence of a predator, all the fish in a fish school suddenly turn their bodies away from it and flee. Although the LTM is able to model a spread, it fails to model how quickly a cascade occurs. In order to capture the speed of a cascade, in this chapter, we generalize the LTM to a continuous threshold model (CTM), which is presented in Part II: Chapter 10 which appears as Zhong and Leonard [98]. The CTM is a continuous dynamical system approach where the speed of a change of state can be observed. The CTM also allows us to study deterministic thresholds and how heterogeneity in thresholds influences a cascade.

We empirically observe the cascade response - a quick change of activity levels of agents - with the CTM on a class of networks. This is in contrast to a contained response, where the activity levels change slowly. We show that the cascade response is caused by a subcritical pitchfork bifurcation in the CTM. We show how the cascade response depend on the network structure and heterogeneous agents by proving the condition of a subcriticial pitchfork bifurcation. We conclude that a large disparity in network structure and individual preferences lead to a cascade.

3.1 Continuous Threshold Model

We consider a network of agents and the interaction among agents are encoded in the graph. Each agent i is associated with a state $x_i \in \mathbb{R}$, representing the agent's activity level. Different from the LTM, here each agent has a continuous activity level. We proposed the continuous threshold model

(CTM), which describes how the states of agents evolve over time, as follows

$$\dot{x}_i = \underbrace{-d_i x_i}_{\text{resistance}} + \underbrace{\sum_{j=1}^{N} a_{ij} u S(v x_j)}_{\text{social}} + \underbrace{d_i(1-2\mu_i)}_{\text{preference}}. \tag{3.1}$$

The dynamics is inspired by the Hopfield network [40] and adapted from nonlinear consensus dynamics [32]. Similar dynamics has also been used to model opinion dynamics [9]. The dynamics consist of three terms. The first term is an inertia term, where d_i is the degree of agent i. The second term contains the social cues from other agents where $a_{ij} = 1(0)$ if agent j is (is not) a neighbor of agent i. The function $S(\cdot)$ is a symmetric sigmoidal function, e.g., $\tanh(\cdot)$. The parameter u represents "social sensitivity" since a large u means a large attention to social signals. The parameter v represents "social effort" since a large v means a strong cue sent by neighbors. The third term is a preference term where $\mu_i \in [0, 1]$ is the threshold of agent i, which captures agent i's tendency to raise its activity level.

The CTM is a generalization of the LTM. By letting $u = 1$ and $v \to \infty$, the CTM becomes

$$\frac{1}{d_i}\dot{x}_i = -x_i + 2\Big(\underbrace{\sum_{j=1}^{N} \frac{a_{ij}}{d_i}\frac{S(v x_j)+1}{2}}_{\text{fraction of active ngbrs}} - \mu_i\Big). \tag{3.2}$$

In particular, we interpret a positive(negative) state as active(inactive). Then the dynamics behave as the LTM with deterministic thresholds, which are challenging to understand with the LTM. The CTM formalism generalizes the binary activity into a continuous activity level and the rules of spreading in LTM into continuous dynamics. The CTM enables us to study cascade with tools in dynamical systems.

3.2 Networks with a Chain of Three Clusters

Our goal is to investigate how heterogeneity in thresholds influence cascade in the network. We start with a class of networks where agents are clustered into multiple groups. The agents in the same group have the same threshold but different groups have different thresholds. This class of networks is motivated by the scenarios where the thresholds represent a trait of agent that is determined by spatial locations of agents. For example, people in a country with a large number of confirmed cases of a potentially deadly infectious disease, such as COVID-19, have a tendency

to adopt advice, such as social distancing or wearing a mask, from medical experts. People in a neighboring country with less confirmed cases might have a tendency to do so. The spread of mitigation strategies, e.g. wearing a mask, can be modelled by CTM where people in a region with larger confirmed cases choose a lower threshold. We start with a class of networks with a chain of three clusters, as illustrated in Figure 3.1. The left cluster (cluster 1) is a high response cluster since every agent i in it has a threshold $\mu_i = 1/2 - \epsilon$, representing a tendency to become active. The right cluster is a low response cluster (cluster 2) since every agent j in it has a threshold $\mu_j = 1/2 + \epsilon$, representing a reluctance to become active. The middle cluster (cluster 3) is a neutral response cluster since every agent k in it has a threshold $\mu_k = 1/2$.

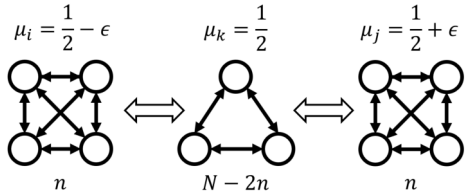

Figure 3.1: Network with three clusters: $N = 11$ and $n = 4$. Cluster 1 (high response) is on the left, cluster 2 (low response) is on the right, and cluster 3 (neutral response) is in the middle. White arrows indicate all-to-all, undirected connections between nodes in clusters.Repeated from Figure 10.1 [98].

There is an underlying symmetry in the network structure. There are n agents in clusters 1, n agents in cluster 2 and $N - 2n$ agents in cluster 3. All the edges are undirected. The connections within a cluster are all-to-all. Each agent in cluster 1 is connected to each agent in cluster 3 and each agent in cluster 2 is connected to each agent in cluster 3.

To analyze CTM on this class of networks, we let $v = 1$ and u as a feedback control. We treat u as a tunable parameter of the system for now. We show that trajectories of Eqn (3.1) converge exponentially to the following three-dimensional manifold

$$\dot{y}_1 = -(N - n - 1)y_1 + (n - 1)uS(y_1) + (N - 2n)uS(y_3) + 2(N - n - 1)\epsilon \tag{3.3}$$

$$\dot{y}_2 = -(N - n - 1)y_2 + (n - 1)uS(y_2) + (N - 2n)uS(y_3) - 2(N - n - 1)\epsilon \tag{3.4}$$

$$\dot{y}_3 = -(N - 1)y_3 + (N - 2n - 1)uS(y_3) + nuS(y_1) + nuS(y_2). \tag{3.5}$$

Let $\mathbf{y} = [y_1, y_2, y_3]^T$ and $F(\mathbf{y}, u, \epsilon)$ the right hand side of (3.3)-(3.5). Next, we show that F exhibits a pitchfork bifurcation and we prove the condition for the transition from a supercritical pitchfork to a subcritical pitchfork. We also show that the supercritical pitchfork implies a contained response and the subcritical pitchfork implies a cascade.

3.3 Condition for Cascade - Subcritical Pitchfork Bifurcation

By symmetry, $\mathbf{y}^\star = [y^\star, -y^\star, 0]^T$ is an equilibrium of (3.3)-(3.5). For a given ϵ and u, $\mathbf{y}^\star$ satisfies

$$-(N-n-1)y^\star + (n-1)uS(y^\star) + 2(N-n-1)\epsilon = 0. \tag{3.6}$$

Here we apply a perturbation analysis to investigate if there is a perturbed solution around $\mathbf{y}^\star$ as $\mathbf{y}^\star + \Delta\mathbf{y} = [y^\star + \Delta y_1, -y^\star + \Delta y_2, \Delta y_3]^T$, where $\Delta\mathbf{y} \neq \mathbf{0}$. The change of existence of those perturbed solution indicates a bifurcation in the dynamics. In the following analysis, we will use Taylor series expansions to the third order.

$$S(y^\star+\Delta y_1) = S(y^\star) + S'(y^\star)\Delta y_1 + \frac{1}{2}S''(y^\star)(\Delta y_1)^2 + \frac{1}{6}S'''(y^\star)(\Delta y_1)^3 + o((\Delta y_1)^3) \tag{3.7}$$

$$S(-y^\star+\Delta y_2) = S(-y^\star)+S'(-y^\star)\Delta y_2+\frac{1}{2}S''(-y^\star)(\Delta y_2)^2 + \frac{1}{6}S'''(-y^\star)(\Delta y_2)^3 + o((\Delta y_2)^3) \tag{3.8}$$

$$S(\Delta y_3) = S'(0)\Delta y_3 + \frac{1}{2}S''(0)(\Delta y_3)^2 + \frac{1}{6}S'''(0)(\Delta y_3)^3 + o((\Delta y_3)^3). \tag{3.9}$$

Without loss of generality, we use $\tanh(\cdot)$ as the sigmoidal function in the analysis. For other types of sigmoidal functions, the analysis is similar. With $S(\cdot) = \tanh(\cdot)$, Eqn. (3.9) becomes

$$\tanh(\Delta y_3) = \Delta y_3 - \frac{1}{3}\Delta y_3^3 + o((\Delta y_3)^3). \tag{3.10}$$

The following lemma and proposition reduce the dynamics to one-dimensional dynamics which match the normal form of a pitchfork bifurcation. We can then derive the condition for a subcritical bifurcation and thus a cascade.

Lemma 3 ([98]). *Assume $\Delta x \in \mathbb{R}$ and $\Delta y \in \mathbb{R}$ have small magnitudes and satisfy*

$$a(\Delta x)^3 + b(\Delta x)^2 + c\Delta x = -\Delta y + \frac{1}{3}(\Delta y^3) + o((\Delta y)^3). \tag{3.11}$$

Then we have

$$\Delta x = -\frac{1}{c}\Delta y - \frac{b}{c^3}(\Delta y)^2 + \left(\frac{1}{3c} - \frac{2b^2}{c^5} + \frac{a}{c^4}\right)(\Delta y)^3 + o((\Delta y^3)).$$

Proposition 3 ([98]). *Consider dynamics (3.3)-(3.5) with $S(\cdot) = \tanh(\cdot)$. The conditions for equilibria of*

the perturbed dynamics of (3.3)-(3.5) around $y^\star$ can be reduced to the following single condition:

$$\frac{\mathrm{d}}{\mathrm{d}t}(\Delta y_3) = \lambda_1 \Delta y_3 + \lambda_3 (\Delta y_3)^3 + o((\Delta y_3)^3) = 0, \tag{3.12}$$

where λ_1 and λ_3 depend on $y^\star$, u, N, n as follows:

$$\begin{aligned}\lambda_1(y^\star, u, N, n) =& -(N-1) - \Big(1 + \frac{n+1}{n-1}(N-2n)\Big)u \\ & - \frac{n(N-n-1)}{n-1}\frac{2}{c}\end{aligned} \tag{3.13}$$

$$\begin{aligned}\lambda_3(y^\star, u, N, n) =& \frac{1}{3}\Big(1 + \frac{n+1}{n-1}(N-2n)\Big)u \\ & + \frac{n(N-n-1)}{n-1}\Big(\frac{2}{3c} - \frac{4b^2}{c^5} + \frac{2a}{c^4}\Big)\end{aligned} \tag{3.14}$$

with

$$a(y^\star, N, n) = \frac{n-1}{N-2n}\frac{1}{6}\tanh'''(y^\star) \tag{3.15}$$

$$b(y^\star, N, n) = \frac{n-1}{N-2n}\frac{1}{2}\tanh''(y^\star) \tag{3.16}$$

$$c(y^\star, u, N, n) = \frac{n-1}{N-2n}\tanh'(y^\star) - \frac{N-n-1}{(N-2n)u}. \tag{3.17}$$

Eqn. (3.11) exhibits the normal form of a pitchfork bifurcation. In particular, if $\lambda_3 < 0$, Δy_3 undergoes a supercritical pitchfork bifurcation; if $\lambda_3 > 0$, Δy_3 undergoes a subcritical pitchfork bifurcation. Fig. 3.2 shows the supercritical and subcritical pitchforks. The green curves represent how the average state $\bar{y} = (n y_1 + n y_2 + (N-2n) y_3)/N$ changes as the bifurcation parameter u slowly increases because of feedback. As we can see, if the pitchfork is supercritical, the system has a contained response; if the pitchfork is subcritical, the average state suddenly changes to a stable branch when u crosses the bifurcation point u_c. The sudden change of states indicates a cascade. Fig. 3.3 shows the bifurcation diagram when there is a small positive disturbance to the dynamics (3.3). In this case, the contained response and the cascade response still exist. Moreover, even a negative initial condition of average state $\bar{y}(0) < 0$ would trigger a cascade.

As the subcritical pitchfork indicates a cascade, the following proposition shows when the transition from a supercritical pitchfork to a subcritical pitchfork exists.

Proposition 4. *The transition from a supercritical pitchfork bifurcation to a subcritical pitchfork bifurcation of dynamics (10.3)-(10.5) with $S(\cdot) = \tanh(\cdot)$ occurs when λ_3 crosses zero from negative to positive. The*

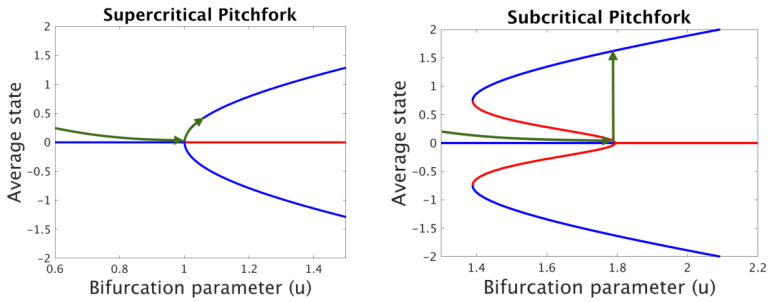

Figure 3.2: Pitchfork bifurcation diagrams: supercritical (left) and subcritical (right). Blue (red) curves are stable (unstable) solutions. Green curves are trajectories as u slowly increases. Repeated from 10.2 [98].

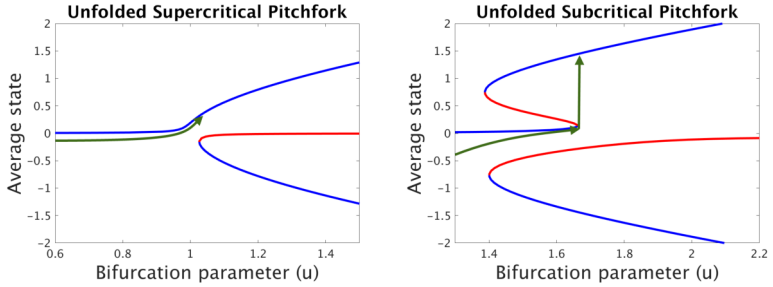

Figure 3.3: Unfolded pitchfork bifurcation diagrams: supercritical (left) and subcritical (right). Colors are as in Fig. 3.2. Repeated from Fig. 10.3 [98].

condition for the transition is

$$\lambda_3(y^\star, N, n) = 0, \tag{3.18}$$

where

$$\lambda_3(y^\star, N, n) = -\frac{1}{3}(N-1) + \frac{n(N-n-1)}{n-1} \times \frac{2a(y^\star, N, n)c(y^\star, u(y^\star, N, n), N, n) - 4b^2(y^\star, N, n)}{c^5(y^\star, u(y^\star, N, n), N, n)}, \tag{3.19}$$

and

$$u(y^\star, N, n) = \frac{-c_1 + \sqrt{c_1^2 - 4c_2c_0}}{2c_2} = \frac{-2c_0}{\sqrt{c_1^2 - 4c_2c_0} + c_1}. \tag{3.20}$$

Here, a, b, c are given by (10.15), (10.16), (10.17) and

$$c_2(y^\star, N, n) = \left(n + 1 + \frac{n-1}{N-2n}\right)\tanh'(y^\star) \tag{3.21}$$

$$c_1(y^\star, N, n) = \frac{(N-2n-1)(N-n-1)}{(N-2n)} + \frac{(N-1)(n-1)}{N-2n}\tanh'(y^\star) \tag{3.22}$$

$$c_0(y^\star, N, n) = -\frac{(N-n-1)(N-1)}{N-2n}. \tag{3.23}$$

The value of $y^\star$ at the transition is the solution of (10.23). The value of u at the transition is a function of $y^\star$, N, and n (10.25). The value of ϵ at the transition is also a function of $y^\star$, N, and n:

$$\epsilon(y^\star, N, n) = \frac{1}{2}y^\star - \frac{(n-1)}{2(N-n-1)}u(y^\star, N, n)\tanh(y^\star). \tag{3.24}$$

With Proposition 4, the following theorem shows the condition for the existence of a cascade response in dynamics (3.3) to (3.5). The existence depends on the network structure N, n and the disparity between thresholds ϵ.

Theorem 4 ([98])**.** *Given N and n, if there exists a $y_+^\star > 0$ such that $\lambda_3(y_+^\star, N, n) > 0$, then there exists $y_0^\star \in (0, y_+^\star)$ and $y_1^\star \in (y_+^\star, +\infty)$ such that $\lambda_3(y_0^\star, N, n) = \lambda_3(y_1^\star, N, n) = 0$. In particular, the existence of $y_0^\star$ indicates a transition from supercritical pitchfork bifurcation to subcritical pitchfork bifurcation at $\epsilon(y_0^\star)$ and $u(y_0^\star)$. This implies a cascade in the network with three clusters. If there does not exist such a $y_+^\star$, then there is no such transition and thus no cascade.*

Based on Theorem 4, Fig. 3.4 shows for a fixed N, how n and ϵ determine the existence of $y_+^\star > 0$ and, in turn, a cascade. We observe that a large disparity - in the sense of large n, indicating large clusters in the ends of the chain and large ϵ, indicating large difference in the thresholds - lead to a cascade.

3.4 An Example of CTM

From previous sections, we derived conditions under which the cascade happens. We showed the bifurcation parameter is the social sensitivity u and the cascade happens when u crosses a subcritical bifurcation. The social sensitivity of a group depends on the states of agents in the group. This dependence is captured by modelling social sensitivity u as a feedback control, where $u = u_0 S(\kappa|\bar{x}_s|)$. Here $\bar{x}_s$ is the slow filtered average state with the dynamics $\dot{\bar{x}}_s = \kappa_s(\bar{x} - \bar{x}_s)$, where $\bar{x} = \sum_{i=1}^N x_i/N$ and $u_0, \kappa, \kappa_s > 0$. As the average state moves away from zero, the social sensitivity

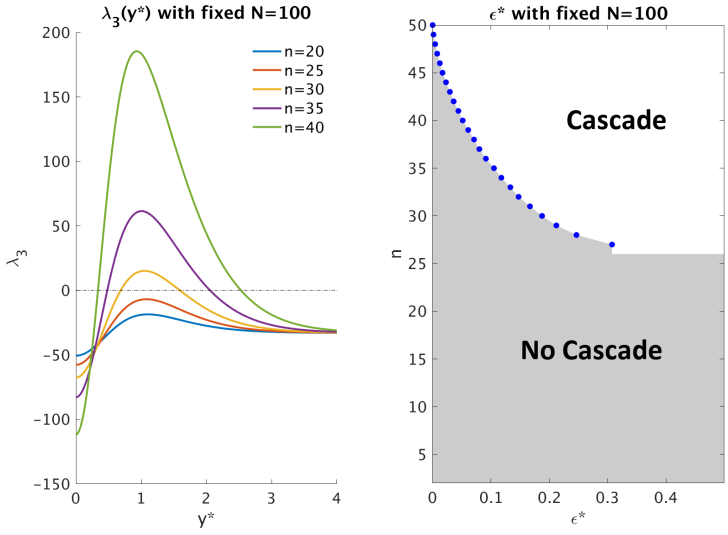

Figure 3.4: The curves of $\lambda_3(y^\star)$ for different values of n and fixed N (left). For lower n, $\lambda_3(y^\star)$ remains negative. For higher n, $\lambda_3(y^\star) = 0$ has two solutions, and thus at the smaller solution $y_0^\star$, there is a transition from supercritical to subcritical pitchfork, and the possibility of a cascade. Critical disparity ϵ^* for different values of n and fixed N (right). For $n \geq 27$, $\epsilon > \epsilon^*$ leads to a cascade. Repeated from Fig. 10.4 [98].

u would slowly ramp up and eventually crosses the bifurcation point and lead to a cascade. We illustrate this with the example of CTM on the network structure in Fig. 3.1 and threshold disparity $\epsilon = 0.2$. The initial conditions of the state of agents are generated randomly, where the average is negative. In the simulation we choose $u_0 = 3$, $\kappa = 10$, and $\kappa_s = 0.05$ for the feedback control. Fig. 3.5 shows how states evolve over time. We observe that the trajectories converge to three clusters, where the red, green and blue trajectories correspond to agents in cluster 1, 2, and 3. An external perturbation $\beta = 1$ is added to an agent in cluster 1, resulting in the agent's state perturbed above the red clustered states. The fast convergence is a perturbation of the dynamics (3.3)-(3.5). The state values converges to a perturbed $\mathbf{y}^\star = [y^\star, -y^\star, 0]^T$. The perturbation leads to a slow increase in $\bar{x}_s$, which results in a slow increase in u. When u crosses the bifurcation point, the states of all agents quickly change and a cascade response happens.

Here the network structure $N = 11$ and $n = 4$ together with the threshold disparity $\epsilon = 0.2$ determines a subcritical pitchfork in the unperturbed dynamics. The perturbation $\beta = 1$ results in a unfolded subcritical pitchfork thus a negative average initial state results in a cascade as shown in the right of Fig. 3.3.

In this chapter, we generalized the LTM to a continuous dynamical model - the CTM. We showed that with a chosen set of parameters, the CTM reduced to the LTM with deterministic thresholds.

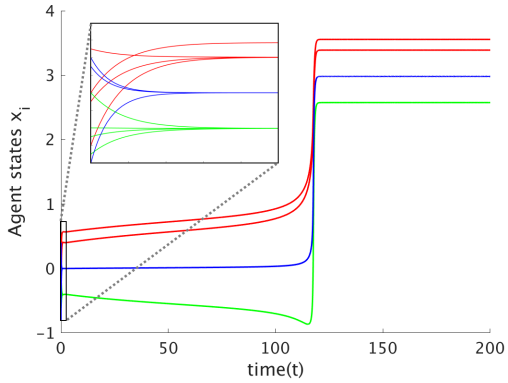

Figure 3.5: Agent state trajectories of the CTM in a network with three clusters, $N = 11$, $n = 4$. There is a cascade corresponding to the unfolded subcritical pitchfork as can be expected since $0.2 = \epsilon > \epsilon(y_0^\star, N, n) = 0.11$. Repeated from 10.5 [98].

As the LTM with deterministic thresholds lacks analytical tractability, we studied the CTM to leverage the tools in the field of continuous dynamical systems. Our motivating examples are networks where agents are grouped into clusters. We studied the CTM on chains of three clusters and showed that the cascade response of the group is due to an underlying subcritical pitchfork bifurcation. We rigorously proved the condition for the existence of such a subcritical pitchfork, or equivalently proved the condition for the cascade. The bifurcation parameter is the social sensitivity u, which is designed to be a feedback parameter dependent on the average state. In this way, we showed that the model exhibits a quick change of states, where the speed of the change is not captured in the LTM. The mechanism of the fast change in state values - subcritical pitchfork - is only one possible way to model cascade. Our model paves the way for using continuous dynamical systems to model and understand the cascade response observed from animal groups. Both the LTM and CTM assumes agents can directly sense the activity levels of neighboring agents, but this might not be the case in engineering applications such as a robot team. We address this in the next two chapters.

Chapter 4

Learning Lagrangian and Hamiltonian Dynamics from Trajectory Data

We have investigated cascade dynamics with the LTM and the CTM. Both the LTM and the CTM assume that agents possess a scalar state variable representing the activity, which can be directly observed by the others. In the following two chapters, we relax this assumption and study the scenarios that the activity need to be inferred from observation data. Consider a robot team where each agent has its own dynamics. A robot in the team need to infer the activity of a neighboring robot from its configuration or dynamics. For instance, a lifted robot arm or a rotating wheeled robot indicates high activity levels while a legged robot in a sit-down position or a humanoid at rest suggests a low activity level. In these scenarios, the activity level can be inferred from the configuration or dynamics of agents and the dynamics of a neighboring agent need to be inferred from observation data or sensor data. In this chapter, we assume the agents can sense the position and velocity of neighboring agents and study how to learn the dynamics from the sensor data. In the next chapter, we assume the agents are equipped with camera sensors and they need to infer dynamics from image data. We demonstrate that our methods can successfully achieve these goals. We leave the learning of cascade dynamics as a future work.

This chapter summarizes results from our work presented in Part II: Chapter 11 and 12 which appear as Zhong et al. [101] and Zhong et al. [102]. The goal in this chapter is to learn dynamics from

trajectory data, i.e., position and velocity data. Moreover, we hope we can learn dynamics with less data, achieve better generalization and gain interpretability by incorporating physics priors. The physics prior we choose to incorporate is Lagrangian/Hamiltonian dynamics. We first introduce the class of dynamics which serves as a physics prior in the learning algorithm.

4.1 Lagrangian/Hamiltonian Dynamics

Lagrangian and Hamiltonian dynamics are both reformulation of Newtonian dynamics. In this section, we introduce Lagrangian dynamics and Hamiltonian dynamics and apply them on planar rigid body systems. Although the resulting equations of motion is the same, Lagrangian and Hamiltonian frameworks interpret dynamics from an energy perspective. We will use this formulation as a model prior in the learning algorithm.

4.1.1 Lagrangian Dynamics

For a physical system, the configuration of the system, e.g., positions of objects, at time t is described by m independent generalized coordinates $\mathbf{q}(t) = (q_1(t), q_2(t), ..., q_n(t))$, where m is the number of degrees of freedom (DOF). The configuration of the system traces out a trajectory over time. Lagrangian dynamics describe how this trajectory evolves. From D'Alembert's principle, the equations of motion (EOM) of the physical system is given by the following Euler-Lagrange equation

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{Q}^{nc}, \tag{4.1}$$

where the scalar function $L(\mathbf{q}, \dot{\mathbf{q}})$ is referred to as the Lagrangian, $\dot{\mathbf{q}} = \mathrm{d}\mathbf{q}/\mathrm{d}t$ is the time derivative of the generalized coordinates called generalized velocities, and $\mathbf{Q}^{nc}$ denote the non-conservative generalized forces. The Lagrangian $L(\mathbf{q}, \dot{\mathbf{q}})$ is defined as the difference between kinetic energy $T(\mathbf{q}, \dot{\mathbf{q}})$ and potential energy $V(\mathbf{q})$. For a rigid body system, the Lagrangian is

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V(\mathbf{q}), \tag{4.2}$$

where $\mathbf{M}(\mathbf{q})$ denotes the mass matrix. In the following, we assume the only non-conservative generalized forces are control inputs, i.e., $\mathbf{Q}^{nc} = \mathbf{g}(\mathbf{q})\mathbf{u}$, where $\mathbf{u}$ is a vector of control inputs such as forces or torques and $\mathbf{g}(\mathbf{q})$ is the input matrix. By substituting the Lagrangian and the non-conservative forces, we get the EOM in the form of m second-order ordinary differential equations

(ODE):

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\Big(-\frac{1}{2}\frac{\mathrm{d}\mathbf{M}(\mathbf{q})}{\mathrm{d}t}\dot{\mathbf{q}} - \frac{\mathrm{d}V(\mathbf{q})}{\mathrm{d}\mathbf{q}} + \mathbf{g}(\mathbf{q})\mathbf{u}\Big). \tag{4.3}$$

In the next section, we will derive the same EOM from the Hamiltonian framework.

4.1.2 Hamiltonian Dynamics with Control

Hamiltonian dynamics can be derived from Lagrangian dynamics by setting the non-conservative force as zero. As we will see later, an important property of Hamiltonian dynamics is the conservation of the Hamiltonian, usually representing the total system energy. This does not apply in general to control applications where the control input would in most cases make the energy of the system change. In this work, we consider an extension of Hamiltonian dynamics to include the control input. First, we introduce the original Hamiltonian dynamics. Hamiltonian dynamics describe the changes of generalized coordinates $\mathbf{q}(t) = (q_1(t), q_2(t), ..., q_n(t))$ and generalized momenta $\mathbf{p}(t) = (p_1(t), p_2(t), ..., p_n(t))$ of the system over time:

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \qquad \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}}. \tag{4.4}$$

where the dynamics is governed by a scalar function $H(\mathbf{q}, \mathbf{p})$, which is referred to as the Hamiltonian. In almost all physical systems, the Hamiltonian equals the total energy of the system, so we have

$$H(\mathbf{q}, \mathbf{p}) = T(\mathbf{q}, \mathbf{p}) + V(\mathbf{q}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}(\mathbf{q})\mathbf{p} + V(\mathbf{q}), \tag{4.5}$$

It can be shown that

$$\dot{H} = \left(\frac{\partial H}{\partial \mathbf{q}}\right)^T\dot{\mathbf{q}} + \left(\frac{\partial H}{\partial \mathbf{p}}\right)^T\dot{\mathbf{p}} = 0, \tag{4.6}$$

i.e., the total energy is conserved. In order to model systems for control applications, we propose the following generalization motivated by the port-Hamiltonian formalism:

$$\begin{pmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{pmatrix} = \begin{pmatrix}\frac{\partial H}{\partial \mathbf{p}}\\ -\frac{\partial H}{\partial \mathbf{q}}\end{pmatrix} + \begin{pmatrix}\mathbf{0}\\ \mathbf{g}(\mathbf{q})\end{pmatrix}\mathbf{u}. \tag{4.7}$$

As we can see, when $\mathbf{u} = \mathbf{0}$, the system is energy conserved. Notice that $\mathbf{p} = \mathbf{M}(\mathbf{q})\dot{\mathbf{q}}$, so we can express Eqn. (4.7) as a second-order ODE of $\mathbf{q}$, which is exactly (4.3), since Hamiltonian and Lagrangian are both reformulations of Newtonian dynamics.

4.2 Neural ODE for State-space Model

After formalizing the class of dynamics we aim to learn, we need a learning algorithm to learn the dynamics from data. Neural ODE [17] is a framework to learn an unknown first-order ODE from data using neural networks. As the Lagrangian/Hamiltonain dynamics can be formulated as state-space models, which are first-order ODEs, in this section, we summarize the adaptation of Neural ODE for state-space models proposed in Zhong et al. [101] (Chapter 11). This serves as the basis of learning Lagrangian/Hamiltonian dynamics.

Here we consider the following state-space model

$$\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{u}), \tag{4.8}$$

with state $\mathbf{s}$ and control $\mathbf{u}$. If the right hand side function $\mathbf{f}$ is known, i.e., derived by Lagrangian or Hamiltonian dynamics, we can predict how the state evolves given an initial condition $\mathbf{s}_0$ and a sequence of control input $\mathbf{u}_{t_0,t_1,\ldots,t_n}$, by integrating the state-space model, which is a first-order ODE. The problem we would like to solve, however, is that if $\mathbf{f}$ is unknown, how to infer it from data? If we have data of a sequence of state and control pair $(\mathbf{s}, \mathbf{u}^c)_{t_0,t_1,\ldots,t_n}$, where $\mathbf{u}^c$ remains constant in a trajectory, the state-space model (4.8) can be written in the following augmented form.

$$\begin{pmatrix} \dot{\mathbf{s}} \\ \dot{\mathbf{u}} \end{pmatrix} = \begin{pmatrix} \mathbf{f}(\mathbf{s}, \mathbf{u}) \\ \mathbf{0} \end{pmatrix} = \tilde{\mathbf{f}}(\mathbf{s}, \mathbf{u}). \tag{4.9}$$

Now the dimension of the range and the domain of $\tilde{\mathbf{f}}$ are the same, which allows us to leverage Neural ODE to learn $\tilde{\mathbf{f}}$, and equivalently $\mathbf{f}$, by parametrizing $\tilde{\mathbf{f}}$ with a neural network $\tilde{\mathbf{f}}_\psi$. With this parametrization, we can integrate the ODE with an initial condition $(\mathbf{s}, \mathbf{u}^\mathbf{c})_{t_0}$ and an ODE solver,

$$(\mathbf{s}, \mathbf{u}^c)_{t_1}, (\mathbf{s}, \mathbf{u}^c)_{t_2}, \ldots, (\mathbf{s}, \mathbf{u}^c)_{t_n} = \text{ODESolve}((\mathbf{s}, \mathbf{u}^\mathbf{c})_{t_0}, \tilde{\mathbf{f}}_\psi, t_1, t_2, \ldots, t_n), \tag{4.10}$$

and get estimation for future states. We then minimize the difference between true states and estimated states $L(\mathbf{s}_{t_1}, \ldots, \mathbf{s}_{t_n}; \psi) = \sum_{i=1}^{n} ||\mathbf{s}_{t_i} - \mathbf{s}_{t_i}||_2^2$, so that the neural network parameters ψ get updated by stochastic gradient descent and we get a better and better approximation $\tilde{\mathbf{f}}_\psi$, and equivalently $\mathbf{f}_\psi$. In this way, we are able to infer the state space model (4.8) from data.

The implementation of neural networks with ODE concerns the ODE solver and how we perform backpropagation. In fact, with an arbitrary solver, we can write down the equations in each

integration step. All the computations in each integration step are differentiable, so backpropagation works as in any other neural network models. The problem arises if we use an adaptive solver. As the solver uses finer and finer adaptive time steps, the memory usage would become larger and larger since all the intermediate variables and their derivatives need to be stored in each training step. Chen et al. [17] proposed to use the adjoint method to achieve constant memory cost. We leverage the adjoint method when memory cost is a concern and use the traditional backpropagation otherwise.

4.3 Symplectic ODE-Net: Lagrangian and Hamiltonian Dynamics as State-space Models

As we know how to learn a state-space model in the previous section, our goal is to put Lagrangian/Hamiltonian dynamics (Eqn. (4.3)) into a state-space model, so that we can leverage Neural ODE to infer the dynamics.

The traditional way to turn a second-order ODE into a set of first-order ODE is to define $\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}})$. However, this is not ideal when dealing with data, especially angle data. For example, if we are inferring dynamics of a pendulum, it is problematic if we are given the angle data in the form of q itself since data in the form of q treats the angle as a variable in $\mathbb{R}^1$ while the angle is a variable in $\mathbb{S}^1$. Treating the angle in $\mathbb{R}^1$ does not respect the fact that $q = 0$ and $q = 2\pi$ represent the same configuration of the pendulum. This is the reason that angle data are usually given in the form of $(\cos q, \sin q)$, which represents the angle in $\mathbb{S}^1$. For example, in OpenAI Gym [13] Pendulum task, the trajectory data are given in the form of $(\cos q, \sin q, \dot{q})$.

For a general planar rigid-body system, we can write the generalized coordinates $\mathbf{q}$ as $(\mathbf{r}, \boldsymbol{\phi})$, where $\mathbf{r}$ represents translational generalized coordinates and $\boldsymbol{\phi}$ represents rotational generalized coordinates. In order to respect the geometry of rotational generalized coordinates, we propose the state as $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5) = (\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathbf{r}}, \dot{\boldsymbol{\phi}})$. To incorporate Lagrangian/Hamiltonian dynamics (Eqn. (4.3)), we take the derivative of $\mathbf{s}$ with respect to t and substitute in (Eqn. (4.3)). We

get

$$\dot{\mathbf{s}} = \begin{pmatrix} \mathbf{s}_4 \\ -\mathbf{s}_3 \circ \mathbf{s}_5 \\ \mathbf{s}_2 \circ \mathbf{s}_5 \\ \mathbf{M}^{-1}(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3)\left(-\frac{1}{2}\frac{\mathrm{d}\mathbf{M}(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3)}{\mathrm{d}t}\begin{pmatrix}\mathbf{s}_4\\ \mathbf{s}_5\end{pmatrix} + \begin{pmatrix} -\frac{\partial V(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3)}{\partial \mathbf{s}_1} \\ \frac{\partial V(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3)}{\partial \mathbf{s}_2}\mathbf{s}_3 - \frac{\partial V(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3)}{\partial \mathbf{s}_3}\mathbf{s}_2 \end{pmatrix} + \mathbf{g}(\mathbf{s}_1,\mathbf{s}_2,\mathbf{s}_3)\mathbf{u} \right) \end{pmatrix}, \tag{4.11}$$

where $\circ$ is the Hadamard (element-wise) product. Here the mass matrix $\mathbf{M}(\cdot)$, potential energy $V(\cdot)$ and input matrix $\mathbf{g}(\cdot)$ are not functions of $\mathbf{q}$, but functions of $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3) = (\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi})$, since we need to express the right hand side as a function of $\mathbf{s}$ and $(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$ contains the same information as q.

We assume we are given trajectory data in the form of $(\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathbf{r}}, \dot{\boldsymbol{\phi}}, \mathbf{u}^c)_{t_0,\ldots,t_n}$, then we use three neural networks, $\mathbf{M}_{\psi_1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, $V_{\psi_2}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, and $\mathbf{g}_{\psi_3}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$ to parametrize the mass matrix, potential energy and input matrix, respectively. We use ψ to denote neural network parameters. Then Eqn. (4.11) is a state-space model where the right hand side is parametrized by a neural network. We can then use the augmented dynamics and Neural ODE to learn $\mathbf{M}_{\psi_1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, $V_{\psi_2}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, and $\mathbf{g}_{\psi_3}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$ that explain the data (Section 4.2). This learning framework is referred to as *Symplectic ODE-Net (SymODEN)*[1] since it is first derived from the Hamiltonian framework and the name indicates that the symplectic gradient is incorporated into the model as a physic prior.

4.4 Model Variant: Unstructured Symplectic ODE-Net

In this section, we introduce a model variant by parametrizing the Hamiltonian as a neural network instead of exploiting the structure of the Hamiltonian. We refer to this model variant as *Unstructured Symplectic ODE-Net (Unstr. SymODEN)*. Similar to the previous section, our goal is to put the Hamiltonian dynamics with control (4.7) into the state space form with state $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5) = (\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathbf{r}}, \dot{\boldsymbol{\phi}})$. We use three neural networks as function approximators to approximate mass matrix, Hamiltonian and input matrix i.e., $\mathbf{M}_{\psi_1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, $H_{\psi_2}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{p})$, and $\mathbf{g}_{\psi_3}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, where

$$\mathbf{p} = \mathbf{M}_{\psi_1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)\begin{pmatrix}\mathbf{s}_4\\ \mathbf{s}_5\end{pmatrix}. \tag{4.12}$$

[1] In the original proposal of SymODEN [101], we parametrized the inverse of mass matrix as a neural network. In our later work [99], we parametrized the mass matrix as a neural network and observed that both parametrization have similar effects.

Then Hamiltonian dynamics with control (4.7) can be written as

$$\dot{\mathbf{q}} = \begin{pmatrix} \dot{\mathbf{r}} \\ \dot{\boldsymbol{\phi}} \end{pmatrix} = \frac{\partial H_{\psi_2}}{\partial \mathbf{p}} \tag{4.13}$$

$$\dot{\mathbf{p}} = -\frac{\partial H_{\psi_2}}{\partial \mathbf{q}} + \mathbf{g}_{\psi_3}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)\mathbf{u} = \begin{pmatrix} -\frac{\partial H_{\psi_2}}{\partial \mathbf{s}_1} \\ \mathbf{s}_3 \circ \frac{\partial H_{\psi_2}}{\partial \mathbf{s}_2} - \mathbf{s}_2 \circ \frac{\partial H_{\psi_2}}{\partial \mathbf{s}_3} \end{pmatrix} + \mathbf{g}_{\psi_3}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)\mathbf{u}. \tag{4.14}$$

Then the state-space model is

$$\dot{\mathbf{s}} = \begin{pmatrix} \dot{\mathbf{r}} \\ -\mathbf{s}_3\dot{\boldsymbol{\phi}} \\ \mathbf{s}_2\dot{\boldsymbol{\phi}} \\ \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}_{\psi_1}^{-1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3))\mathbf{p} + \mathbf{M}_{\psi_1}^{-1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)\dot{\mathbf{p}} \end{pmatrix}, \tag{4.15}$$

where $\dot{\mathbf{r}}$, $\dot{\boldsymbol{\phi}}$ and $\dot{\mathbf{p}}$ come from Eqn. (4.13) and (4.14). The derivatives are taken care of by automatic differentiation. Now the right hand side is a function parametrized by neural networks where we can leverage techniques in Section 4.2 to learn from data.

4.5 Model Variant: Dissipative Symplectic ODE-Net

Symplectic ODE-Net and Unstructured Symplectic ODE-Net enforce energy conservation. They are not appropriate for learning a dissipative system such as a damped pendulum. In order to properly learn dissipative system, we leverage the following port-Hamiltonian dynamics:

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = \Bigg(\underbrace{\begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{I} & \mathbf{0} \end{bmatrix}}_{\text{symplectic gradient}} - \underbrace{\mathbf{D}(\mathbf{q})}_{\text{Dissipation}} \Bigg) \begin{bmatrix} \frac{\partial H}{\partial \mathbf{q}} \\ \frac{\partial H}{\partial \mathbf{p}} \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{g}(\mathbf{q}) \end{bmatrix} \mathbf{u}}_{\text{control input}}, \tag{4.16}$$

Compared with Hamiltonian dynamics with control (4.7), the above dynamics have an extra term that represents dissipation. Similar to previous two sections, we can approximate this unknown dissipation with a neural network $\mathbf{D}_{\psi_4}(\cdot)$, and learn it along with other unknown properties using Neural ODE. We refer to this model as *Dissipative Symplectic ODE-Net (Dissi. SymODEN)*. When the structure of Hamiltonian is not exploited, we refer to it as *Unstructured Dissipative Sympletic ODE-Net (Unstr. Dissi. SymODEN)*.

4.6 Experimental Setup and Results

We evaluate proposed models on 8 tasks:

- Task 1: a pendulum with data $(q, p, u^c)_{t_0,\ldots,t_n}$
- Task 2: a pendulum with data $(\cos q, \sin q, \dot{q}, u^c)_{t_0,\ldots,t_n}$
- Task 3: a fully-actuated CartPole system with data $(r, \cos\phi, \sin\phi, \dot{r}, \dot{\phi}, u_1^c, u_2^c)_{t_0,\ldots,t_n}$
- Task 4: a fully-actuated Acrobot with data $(\cos\phi_1, \cos\phi_2, \sin\phi_1, \sin\phi_2, \dot{\phi_1}, \dot{\phi_2}, u_1^c, u_2^c)_{t_0,\ldots,t_n}$
- Task 5: a dissipative pendulum with data $(q, p, u^c)_{t_0,\ldots,t_n}$
- Task 6: a dissipative pendulum with data $(\cos q, \sin q, \dot{q}, u^c)_{t_0,\ldots,t_n}$
- Task 7: a fully-actuated dissipative CartPole system with data $(r, \cos\phi, \sin\phi, \dot{r}, \dot{\phi}, u_1^c, u_2^c)_{t_0,\ldots,t_n}$
- Task 8: a fully-actuated dissipative Acrobot with data
  $(\cos\phi_1, \cos\phi_2, \sin\phi_1, \sin\phi_2, \dot{\phi_1}, \dot{\phi_2}, u_1^c, u_2^c)_{t_0,\ldots,t_n}$

In Task 1, the training data treat the angle of the pendulum as a variable in $\mathbb{R}^1$. We run this task to compare it with Task 2, where the training data treat the angle in $\mathbb{S}^1$. We will show in the next section the issues caused by failing to respect the geometry of angle coordinates.

We evaluate SymODEN, Unstr. SymODEN and two baseline models on Task 1-4 (tasks with no dissipation). The *Geometric Baseline* and *Naive Baseline* do not incorporate Lagrangian/Hamiltonian dynamics and use a multi-layer perceptron (MLP) to model the dynamics. The Geometric Baseline respects the geometry of variables while Naive Baseline does not. We evaluate Dissi. SymODEN, SymODEN, Unstr. Dissi. SymODEN, Geometric Baseline and Naive Baseline on Task 5-8 (tasks with dissipation).

We generate the training data using OpenAI Gym [13]. For all tasks, we randomly generate initial conditions for simulation and combine each initial condition with 5 constant control values $u = -2.0, -1.0, 0.0, 1.0, 2.0$ to generate the trajectories for 20 time steps. We use the 4th order Runge-Kutta method (RK4) for integration.

We propose two metrics to evaluate the performance of models. *Train error* per trajectory is the mean-squared error (MSE) between the estimated states and the true states over 20 time steps (in a trajectory). *Prediction error* per trajectory is the MSE between the estimated states and the true states over 40 time steps using the same initial conditions as the training data. Prediction error measures how well the models perform long-term predictions.

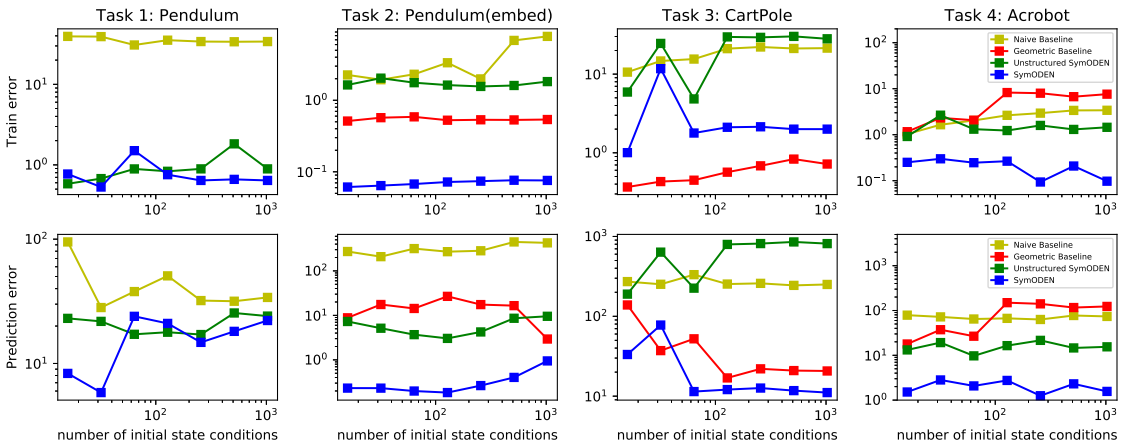

Figure 4.1: Train error per trajectory and prediction error per trajectory for all Task 1-4 with different number of training trajectories. Horizontal axis shows number of initial state conditions (16, 32, 64, 128, 256, 512, 1024) in the training set. Both the horizontal axis and vertical axis are in log scale. Repeated from Fig. 11.5 [101].

Fig. 4.1 shows the train error and prediction error for different models in all four tasks. We observe Symplectic ODE-Net outperforms the other models in Task 1, Task 2 and Task 4 in terms of both metrics. Although Symplectic ODE-Net is not the best in term of train error in Task 3, its better performance as compared to Geometric Baseline in terms of prediction error indicates that Geometric Baseline overfits training data.

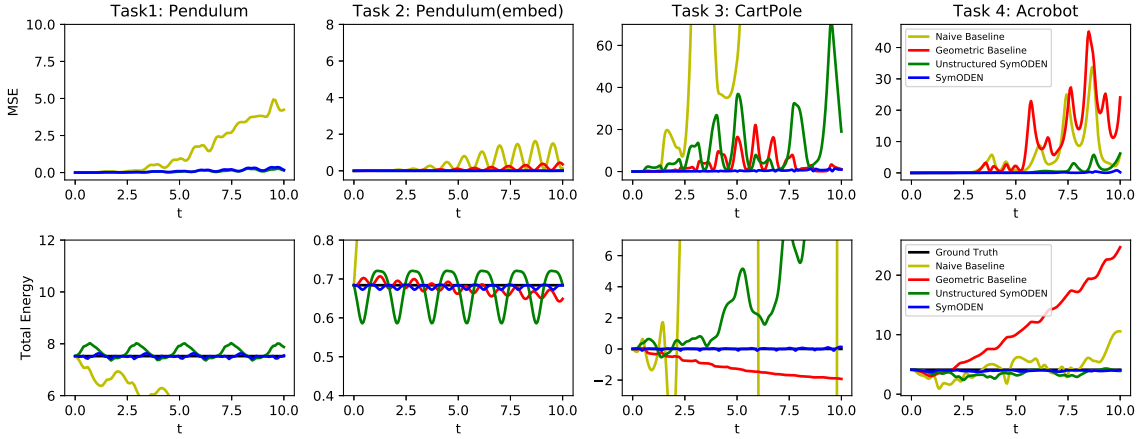

Figure 4.2: Mean square error and total energy of test trajectories of Task 1-4. SymODEN works the best in terms of both MSE and total energy. Since SymODEN has learned the Hamiltonian and discovered the conservation from data, the predicted trajectories match the ground truth. The ground truth of energy in all four tasks stay constant. Repeated from Fig. 11.6 [101].

Fig. 4.2 shows the MSE and total energy over a test trajectory. The MSE of Symplectic ODE-Net stays closer to zero as compared to other models. From the total enery, we observed that both

Symplectic ODE-Net and its unstructured variant conserve total energy over a trajectory. This is because they incorporate Lagrangian/Hamiltonian dynamics. This explains why they perform better in prediction as compared to the baseline models.

Table 4.1: Train, Test and Prediction Errors of Tasks 5-8. Adapted from Table 12.1.

	Naive Baseline	Geometric Baseline	UnStr. Dissi. SymODEN	SymODEN	Dissipative SymODEN
Task 5					
#Params	0.36M	N/A	0.22M	0.13M	0.15M
Train error	26.38 ± 38.00	N/A	34.80 ± 68.53	4.47 ± 6.40	0.88 ± 1.41
Test error	35.03 ± 49.89	N/A	49.44 ± 81.31	7.52 ± 10.13	1.25 ± 1.81
Pred. error	32.544 ± 36.203	N/A	219.36 ± 296.86	96.50 ± 99.56	34.03 ± 47.83
Task 6					
#Params	0.65M	0.46M	0.41M	0.14M	0.16M
Train error	2.02 ± 4.41	0.42 ± 1.16	1.90 ± 3.85	2.37 ± 2.71	0.15 ± 0.27
Test error	2.01 ± 4.99	0.33 ± 1.22	1.61 ± 3.36	2.67 ± 2.83	0.13 ± 0.25
Pred. error	40.18 ± 78.10	0.81 ± 0.68	7.04 ± 13.65	72.78 ± 90.42	1.04 ± 1.3
Task 7					
#Params	1.01M	0.82M	0.69M	0.51M	0.53M
Train error	12.92 ± 15.58	0.48 ± 0.50	12.09 ± 18.38	3.33 ± 3.85	0.88 ± 0.89
Test error	20.07 ± 26.42	1.34 ± 3.19	19.87 ± 23.16	3.80 ± 3.71	1.37 ± 1.30
Pred. error	268.24 ± 204.15	60.12 ± 96.18	366.38 ± 405.45	30.21 ± 34.33	8.32 ± 7.81
Task 8					
#Params	1.46M	0.97M	0.80M	0.51M	0.53M
Train error	1.76 ± 2.26	1.90 ± 2.82	77.56 ± 111.50	2.92 ± 2.58	0.47 ± 0.64
Test error	5.12 ± 9.14	4.87 ± 7.42	122.70 ± 190.90	5.27 ± 6.55	0.81 ± 1.10
Pred. error	36.65 ± 77.16	44.26 ± 95.70	590.77 ± 807.88	68.26 ± 103.46	12.72 ± 32.12

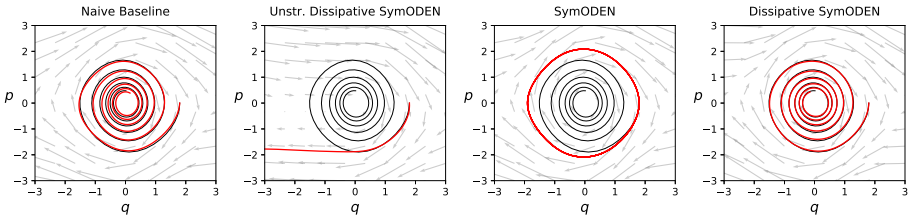

Figure 4.3: Learned trajectories of different models. Red and black lines represent the learned and ground truth trajectories, respectively and the gray arrows show the vector fields learned by each model. *Dissipative SymODEN* learns a more accurate vector field than the *naive baseline* model. Moreover, it appears that whereas *SymODEN* learns an energy-conserved vector field slightly different from the ground truth, *Unstructured Dissipative SymODEN* learns it completely wrong. Adapted from Fig. 12.2 [102].

Table 4.1 shows the train, test and prediction error for Task 5-8, where test errors are computed by predicting 20 time steps given a previously unseen initial condition. We observed that Dissi. SymODEN performs the best in all four tasks.

Fig. 4.3 shows the learned trajectories of different models in Task 5. We can see that by incorporating the dissipation matrix, Dissipative SymODEN predict the trajectory well and outperforms the other models.

4.7 Interpretability

An advantage of incorporating Lagrangian/Hamiltonian dynamics is that we are able to infer the learned properties and gain insight into the system. In this section, we analyze interpretability in Task 1 and 2 (without dissipation) and Task 5 and 6 (with dissipation).

4.7.1 Pendulum Without Dissipation

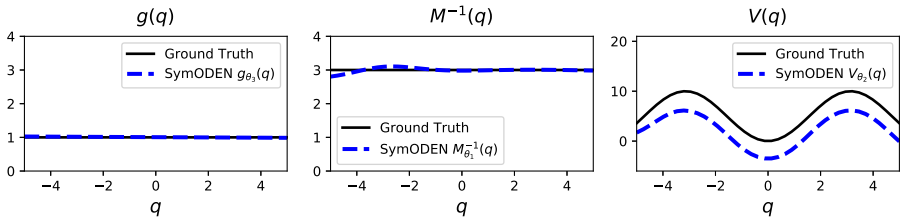

Figure 4.4: Learned functions in Task 1. Adapted from Figure 11.2 [101].

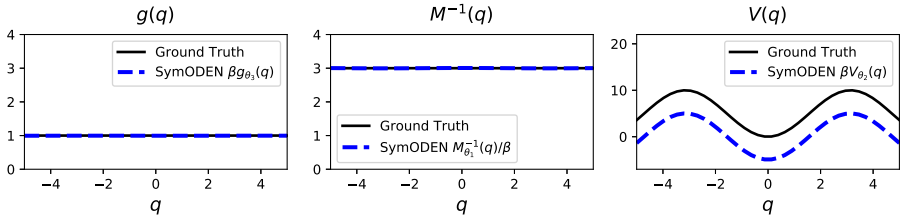

Figure 4.5: Learned functions in Task 2. Without true generalized momentum data, the learned functions match the ground truth with a scaling. Adapted from Figure 11.3 [101].

Fig. 4.4 shows learned input, mass and potential energy in Task 1. The learned potential energy differs from the ground truth by a constant. This is acceptable since potential energy is a relative concept and only the derivative of potential energy plays a role in the dynamics. Around $q = -4$, the mass and the potential energy are not well learned as compared to other range of q. This is because in Task 1 we treat the angle of the pendulum as in $\mathbb{R}^1$. As our training data are not able to cover the whole $\mathbb{R}^1$, our learned model would not be able to generalize to those q that's not in our training data.

Fig. 4.5 shows the learned input, mass and potential energy in Task 2. Compared with Task 1, all the physical properties are well-learnt. This indicates that treating angle data properly benefits learning since our training data covers the whole $\mathbb{S}^1$.

4.7.2 Pendulum With Dissipation

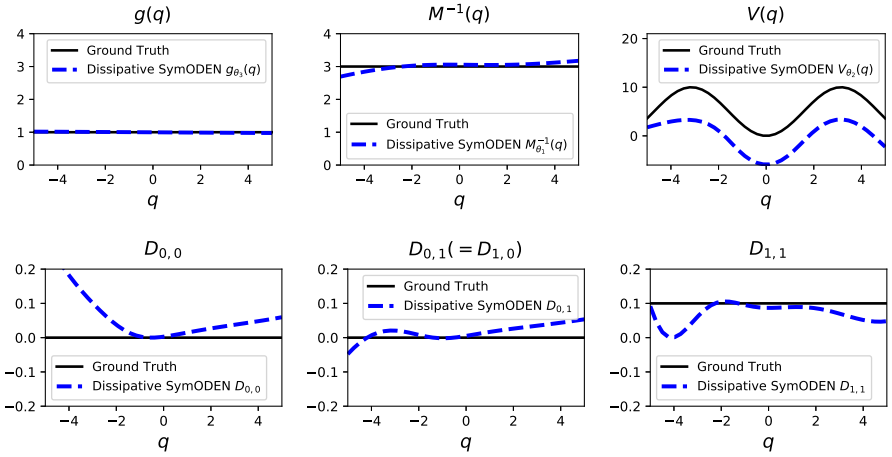

Figure 4.6: Learned functions in Task 5. Adapted from Fig. 12.1 [102].

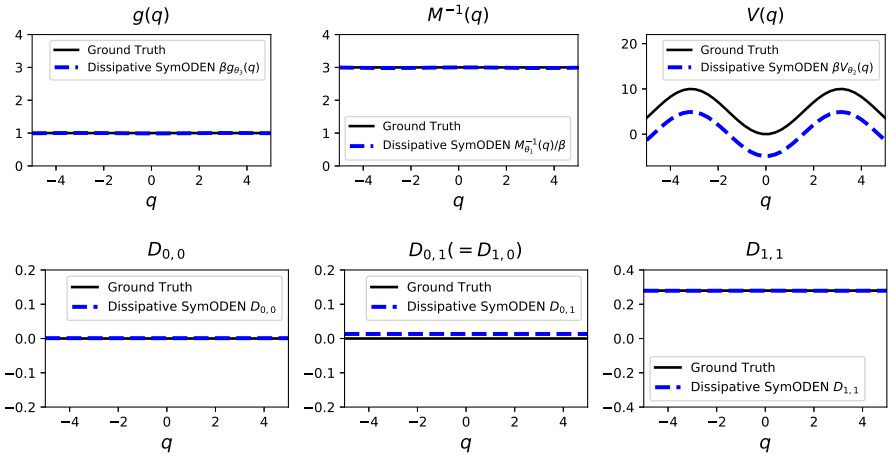

Figure 4.7: Learned functions in Task 6. Adapted from Fig. 12.3 [102].

Fig. 4.6 shows learned functions in Task 5. Similar to Task 1, around $q = -4$, the functions are not well-learned since training data do not cover this range. In particular, the learned dissipation matrix is not satisfactory.

Fig. 4.7 shows learned functions in Task 6. Compared with Fig. 4.6, there is significant improvement in learning the dissipation matrix. We achieve this interpretability by designing the

model to respect the geometry of the coordinates.

4.8 Energy-based Control

Learning dynamics with the Lagrangian/Hamiltonian prior not only allows us to perform prediction, but also allows us to design controllers. In this section, we discuss energy-based control for fully actuated systems, i.e., potential energy shaping and damping injection. We derive the controller with the Lagrangian framework, but the same controller can be derived from the Hamiltonian framework [65, 10]. The designed controller can be integrated into the learned models to control the systems.

For a fully-actuated system, we have control over every degree of freedom. Our control goal is to steer the system to a desired configuration $\mathbf{q}^\star$. For Lagrangian dynamics, the controller design is $\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\dot{\mathbf{q}})$, with $\boldsymbol{\beta}(\mathbf{q})$ the potential energy shaping and $\mathbf{v}(\dot{\mathbf{q}})$ the damping injection. The potential energy shaping lets the system behave as if it is governed by a desired Lagrangian L_d without non-conservative generalized forces, i.e.,

$$\frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\partial L}{\partial \dot{\mathbf{q}}}\Big) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{g}(\mathbf{q})\boldsymbol{\beta}(\mathbf{q}) \quad \Longleftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\partial L_d}{\partial \dot{\mathbf{q}}}\Big) - \frac{\partial L_d}{\partial \mathbf{q}} = 0, \tag{4.17}$$

where the desired Lagrangian differs from the original Lagrangian by the potential energy,

$$L_d(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V_d(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V_d(\mathbf{q}). \tag{4.18}$$

In other words, with $\boldsymbol{\beta}(\mathbf{q})$, we shape the potential energy of the system to a desired potential energy V_d. We design the desired potential energy V_d to have a global minimum at $\mathbf{q}^\star$, so that the system with potential energy shaping would oscillate around $\mathbf{q}^\star$. By (4.17), we have

$$\boldsymbol{\beta}(\mathbf{q}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \frac{\partial V_d}{\partial \mathbf{q}}\Big). \tag{4.19}$$

In practice, a popular choice of V_d is the quadratic form

$$V_d(\mathbf{q}) = \frac{1}{2}(\mathbf{q} - \mathbf{q}^\star)^T\mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star), \tag{4.20}$$

where $\mathbf{K}_p > 0$. To ensure exponential convergence to $\mathbf{q}^\star$, we design damping injection of the form

$$\mathbf{v}(\dot{\mathbf{q}}) = -\mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}(\mathbf{K}_d\dot{\mathbf{q}}), \tag{4.21}$$

where $\mathbf{K}_d > 0$.

4.9 Control Results

Based on the learned dynamics, we can design energy-based controllers following the procedures in the previous section and control systems to a desired configuration. As an illustrative example, we control the pendulum to the inverted position. We define $V_d(q) = -V_{\phi_2}(q)$ so that the desired energy V_d has a global minimum in the inverted position of the pendulum. We choose $K_d = 3$ for damping injection. From Eqn. (4.19) and (4.21), we have

$$u(\cos q, \sin q, \dot{q}) = g_{\psi_3}^{-1}(\cos q, \sin q)\Big(2\big(-\frac{\partial V_{\psi_2}}{\partial \cos q}\sin q + \frac{\partial V_{\psi_2}}{\partial \sin q}\cos q\big) - 3\dot{q}\Big). \tag{4.22}$$

Out of all proposed models, only SymODEN and Dissip. SymODEN directly learn potential energy so that the learned potential energy can be used to construct the controller.

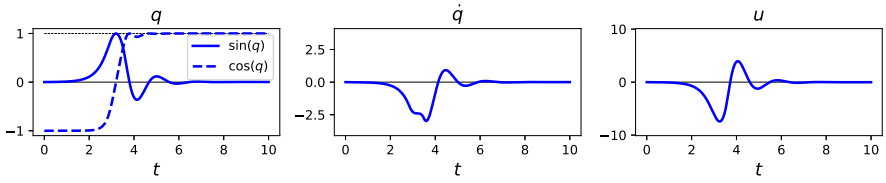

Figure 4.8: Time-evolution of the state variables $(\cos q, \sin q, \dot{q})$ when the closed-loop control input $u(\cos q, \sin q, \dot{q})$ is governed by Equation (11.27). The thin black lines show the expected results. Repeated from Fig. 11.4 [101].

Fig. 4.8 shows the state and control of a controlled trajectory. We construct the controller based on SymODEN trained in Task 2. We feed the output of the controller into the OpenAI Gym simulator.

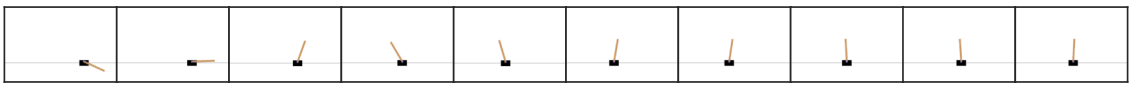

Figure 4.9: Snapshots of a controlled trajectory of the fully-actuated CartPole system with a 0.3s time interval. Repeated from Fig. 11.8 [101].

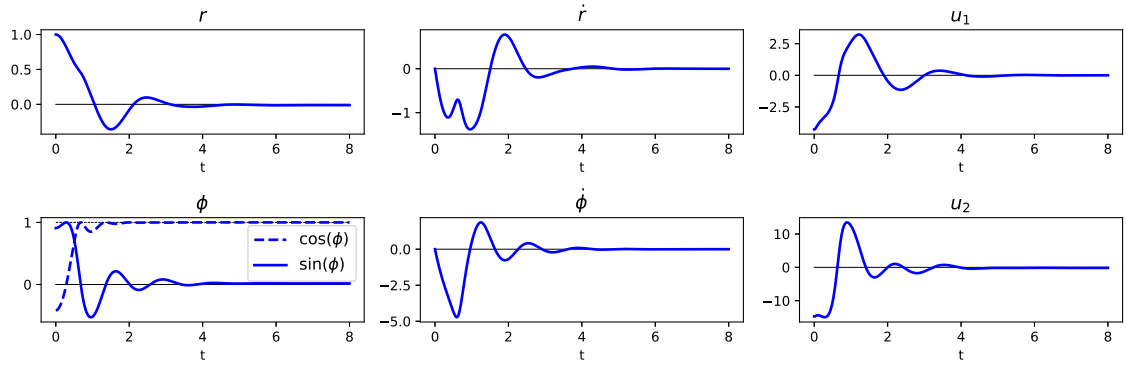

Figure 4.10: Time series of state variables and control inputs of a controlled trajectory shown in Figure 4.9. Black reference lines indicate expected value in the end. Repeated from Fig. 11.9 [101].

The CartPole and the Acrobot are underactuated systems. In general, potential energy shaping alone is not enough to control an underactuated system. Thus, we trained fully-actuated versions of CartPole (Task 3) and Acrobot (Task 4). Fig. 4.9 shows snapshots of a controlled trajectory of the fully-actuated CartPole. Fig. 4.10 shows the states and control inputs vary over time in the trajectory shown in Fig. 4.9. The results shows that we can successfully control the CartPole to the inverted position with energy-based controllers.

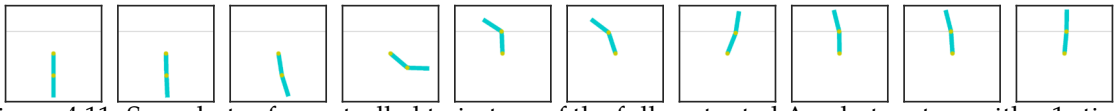

Figure 4.11: Snapshots of a controlled trajectory of the fully-actuated Acrobot system with a 1s time interval. Repeated from Fig. 11.10 [101].

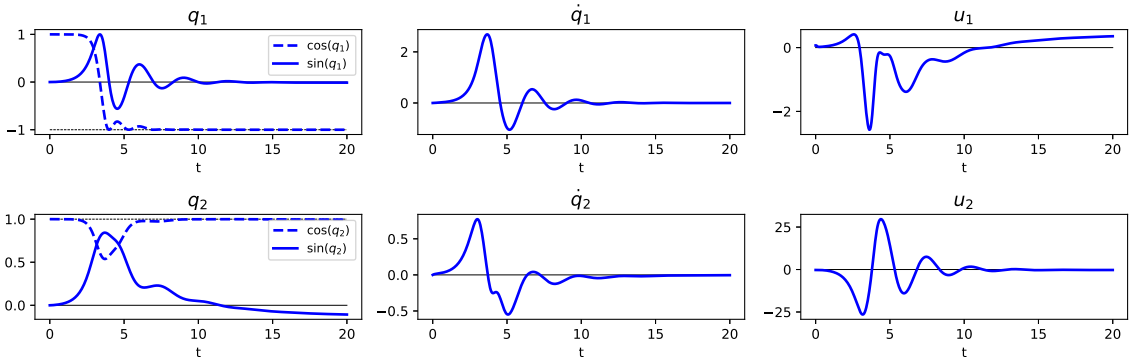

Figure 4.12: Time series of state variables and control inputs of a controlled trajectory shown in Figure 4.11. Black reference lines indicate expected value in the end. Repeated from 11.11 [101].

Fig. 4.11 shows the snapshots of a controlled trajectory of the fully-actuated Acrobot. Fig. 4.12 shows how the states and control inputs evolve in the trajectory shown in Fig. 4.11. We observe that $q_2 = \phi_2$ does not converge to the goal value, 0, in the end. This is because the training data

probably does not cover the range of states around the inverted position of the Acrobot.

In this chapter, we studied the problem of learning Lagrangian/Hamiltonian dynamics from trajectory data. We showed that by incorporating Lagrangian/Hamiltonian prior into deep learning, we are able to learn accurate dynamics with less data and better generalization. Our default model, SymODEN, assumes no dissipation in the system and our method is able to learn energy conserved trajectories when no control is applied. We also proposed a model variant, Dissi. SymODEN, to explicitly model possible dissipation in the system with a dissipation term. We demonstrated both models with examples. Moreover, we showed that the learned model are interpretable in the sense that we can infer physical properties such as mass and potential energy from the learned model. This interpretability enables design of energy-based controllers to control the system to a target configuration based on learned dynamics. This transparent way of learning paves the way for more challenging tasks, such as learning dynamics from image data, which is introduced in the next chapter.

Chapter 5

Learning Lagrangian and Hamiltonian Dynamics from Image Data

In this chapter, we study the problem of learning Hamiltonian/Lagrangian dynamics from image data. We relax our assumption that we are given trajectory data in the previous chapter. Instead, we need to infer trajectories in the configuration space from image data, which is much harder. To tackle this challenge, we propose two neural netowrk models that are jointly trained. The first model - the coordinate-aware variational autoencoder (VAE) - infers interpretable generalized coordinates from images. The second model learns dynamics on the inferred generalized coordinates, and it is the Symplectic ODE-Net introduced in the previous chapter. As we can see, the key to the success of learning dynamics is to infer interpretable generalized coordinates from images, so in this chapter we focus on coordinate-aware VAE. We begin with an formulation of our problem. We then briefly introduce VAE and coordinate-aware VAE. We demonstrate that the traditional VAE is not able to infer interpretable dynamics while our proposed coordinate-aware VAE infers interpretable coordinates which enables prediction and control. This chapter is based on materials presented in Part II: Chapter 13, which appears as Zhong and Leonard [99].

5.1 Problem Formulation

To formalize the problem, we assume that we are given sequences of image and control pair $\mathbf{X} = ((\mathbf{x}^0, \mathbf{u}^c), (\mathbf{x}^1, \mathbf{u}^c)), ..., (\mathbf{x}^{T_{\text{pred}}}, \mathbf{u}^c))$ where $\mathbf{x}^\tau$, $\tau = 0, 1, \ldots, T_{\text{pred}}$, is the image of a planar rigid-body system under constant control $\mathbf{u}^c$ at time $t = \tau\Delta t$. We assume the number of rigid bodies n is given and segmentation of each object in the image is known, i.e., each image can be written as $\mathbf{x}^\tau = (\mathbf{x}_1^\tau, ..., \mathbf{x}_n^\tau)$, where $\mathbf{x}_i^\tau$ is the segmentation of the ith rigid body at $t = \tau\Delta t$. Our goal is to infer the initial state $\mathbf{s}^0 = (\mathbf{r}^0, \cos\boldsymbol{\phi}^0, \sin\boldsymbol{\phi}^0, \dot{\mathbf{r}}^0, \dot{\boldsymbol{\phi}}^0)$ from images. We briefly introduce VAE, which is a neural network model that can infer low dimensional variables that generate high dimensional images. In the next few sections, we will show that the traditional VAE fails to learn interpreable coordinates.

5.2 Variational Autoencoder

Variational Autoencoder (VAE) was proposed by Kingma and Welling [47] to perform variational inference using neural networks. In this section, we briefly introduce VAE and the evidence lower bound associated with it.

We consider learning coordinates of a physical system $\mathbf{q}$ from images of the physical system $\mathbf{x}$. Each image can be generated by specifying the coordinates first, and then render the coordinates into an image. This is referred to as a latent variable model and $\mathbf{q}$ is referred to as the latent variable. It can be presented by the following joint probability distribution of $\mathbf{q}$ and $\mathbf{x}$:

$$P(\mathbf{q}, \mathbf{x}) = P(\mathbf{q})P(\mathbf{x}|\mathbf{q}). \tag{5.1}$$

The problem of variational inference is to find the probability distribution of the latent variable $\mathbf{q}$. As the true probability distribution is usually intractable, we specify a family of densities $\mathcal{D}$ over the latent variables and our goal is to find a probability distribution in $\mathcal{D}$ that is closest to the distribution given image data, where the closeness is measured by KL divergence, i.e.,

$$Q^\star(\mathbf{q}) = \underset{Q(\mathbf{q})\in\mathcal{D}}{\arg\min}\, \text{KL}(Q(\mathbf{q})||P(\mathbf{q}|\mathbf{x})). \tag{5.2}$$

The family $\mathcal{D}$ is usually chosen to be the family of Gaussian distributions. The objective, however,

cannot be computed since it depends on the unknown $\log P(\mathbf{x})$:

$$\mathrm{KL}(Q(\mathbf{q})||P(\mathbf{q}|\mathbf{x})) = \mathbb{E}[\log Q(\mathbf{q})] - \mathbb{E}[\log P(\mathbf{q}|\mathbf{x})] \tag{5.3}$$

$$= \mathbb{E}[\log Q(\mathbf{q})] - \mathbb{E}[\log P(\mathbf{q}, \mathbf{x})] + \log P(\mathbf{x}). \tag{5.4}$$

The expectation in the above equation are taken with respect to $Q(\mathbf{q})$. As we cannot compute the objective, we optimize another computable objective, called evidence lower bound (ELBO). The ELBO is equivalent to our original objective up to a constant $\log P(\mathbf{x})$,

$$\mathrm{ELBO} = \mathbb{E}[\log P(\mathbf{q}, \mathbf{x})] - \mathbb{E}[\log Q(\mathbf{q})] \tag{5.5}$$

$$= \mathbb{E}[\log P(\mathbf{q})] + \mathbb{E}[\log P(\mathbf{x}|\mathbf{q})] - \mathbb{E}[\log Q(\mathbf{q})] \tag{5.6}$$

$$= \mathbb{E}[\log P(\mathbf{x}|\mathbf{q})] - \mathrm{KL}(Q(\mathbf{q})||P(\mathbf{q})). \tag{5.7}$$

In Eqn (5.7), $P(\mathbf{q})$ is the prior distribution of $\mathbf{q}$ and $Q(\mathbf{q}) \in \mathcal{D}$ is the posterior distribution we would like to obtain. In order to leverage neural network to solve this variational inference problem, we let $Q(\mathbf{q})$ depend on image data $\mathbf{x}$, i.e.,

$$\mathrm{ELBO} = \mathbb{E}[\log P(\mathbf{x}|\mathbf{q})] - \mathrm{KL}(Q(\mathbf{q}|\mathbf{x})||P(\mathbf{q})). \tag{5.8}$$

Now, this objective looks like the structure of an autoencoder, since we need an inference model $Q(\mathbf{q}|\mathbf{x})$ which serves as the encoder and a generative model $P(\mathbf{x}|\mathbf{q})$ which serves as the decoder. We can use two neural networks to approximate these probability distributions and learn good approximations by maximizing the ELBO.

It is intuitive to combine the standard VAE with the Symplectic ODE-Net in Chapter 4 to learn the embedding of coordinates and the dynamics simultaneously. However, in practice this intuitive approach fails. After analyzing the failure, we propose the coordinate-aware VAE to learn the value of chosen coordinates in a given planar rigid-body system.

5.3 Coordinate-aware Encoder

As discussed in the previous section, the encoder is an inference model that learns the distribution $Q(\mathbf{q}|\mathbf{x})$ within a family of distributions. Gaussian distribution is the default family of distribution for modelling the latent variable $\mathbf{q}$. For a translational coordinate $r \in \mathbb{R}$, Gaussian distribution

is an appropriate candidate since it is a distribution on $\mathbb{R}^1$. However, for a rotational coordinate $\phi \in \mathbb{S}^1$, Gaussian distribution is not appropriate since it is not a distribution on $\mathbb{S}^1$. Instead of Gaussian distribution, we use von Mises (vM) distribution to model an angle coordinate ϕ. A von Mises distribution is specified by two statistical parameters: $\mu \in \mathbb{R}^2$, $||\mu||^2 = 1$ is the mean, and $\kappa \in \mathbb{R}_{\geq 0}$ is the concentration around μ. When $\kappa = 0$, the von Mises distribution reduces to a uniform distribution on $\mathbb{S}^1$. To sum up, we have the following posterior and prior distribution assumptions in our model

- for ϕ, a posterior distribution $Q(\phi|\mathbf{x}) = \text{vM}((\cos\phi^{\text{m}}, \sin\phi^{\text{m}}), \phi^{\kappa})$ with prior $P(\phi) = \text{vM}(\cdot, 0) = U(\mathbb{S}^1)$.
- for r, a posterior distribution $Q(r|\mathbf{x}) = \mathcal{N}(r^{\text{m}}, r^{\text{var}})$ with prior $P(r) = \mathcal{N}(0, 1)$.

Although we choose the correct probabilistic model, a black box neural network encoder fails to learn interpretable generalized coordinates for a dynamical system. We propose a coordinate-aware encoder by incorporating the geometry of the system into the design of encoder.

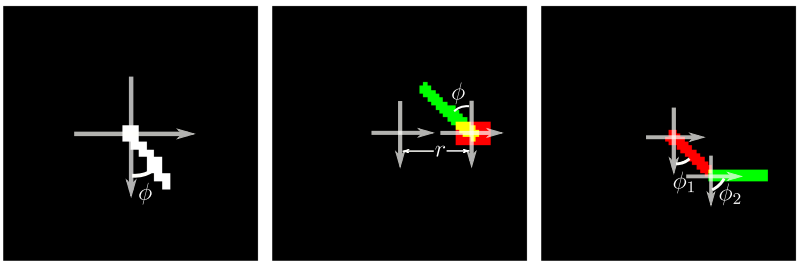

Figure 5.1: One choice of generalized coordinates and their corresponding reference frames in three example systems. Repeated from Fig. 13.2 [99].

Each generalized coordinate q_j determines the position/rotation of a rigid body i_j in the system. Intuitively, the coordinate can be learned from the image segmentation of i_j. The reference frame of a generalized coordinates, however, might change across images since it depends on other generalized coordinates. For example, the CartPole system in Fig. 5.1 has two degrees of freedom. The choices of generalized coordinates are the horizontal position of the cart $q_1 = r$ and the rotation of the pole $q_2 = \phi$. The origin of the reference frame of r is at the center of the image and this applies to all the images. The origin of the reference frame of ϕ is at the center of the cart, which is not at the same position across images since the cart can move horizontally. In order to learn ϕ, we place the center of our encoding attention window of the pole segmentation image at the center of the cart. We achieve this by shift our encoding attention window horizontally with direction and magnitude given by r and then feed it into a neural network to learn ϕ.

The default attention window is the image grid which corresponds to the default reference frame, where the origin is at the center of the image. The above shift of encoding attention window can be formalized by the transformation from the default reference frame to the reference frame of each generalized coordinate. A point (x_d, y_d) in the default reference frame and the corresponding point (x_t, y_t) in the target reference frame is related by the following transformation $\mathcal{T}(x, y, \theta)$

$$\begin{pmatrix} x_t \\ y_t \\ 1 \end{pmatrix} = \mathcal{T}(x, y, \theta) \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix}, \quad \text{where} \quad \mathcal{T}(x, y, \theta) = \begin{pmatrix} \cos\theta & \sin\theta & x \\ -\sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix}. \tag{5.9}$$

Here the transformation is a translation by (x, y) and a rotation by θ. We let $\mathcal{T}((x, y, \theta)_j^{\text{enc}})$ be the transformation from default reference frame to the reference frame of generalized coordinate q_j. This transformation might depend on constant parameters $\mathbf{c}$ associated with size of rigid bodies and generalized coordinates $\mathbf{q}_{-j}$, which represents the vector of all generalized coordinates except q_j. Let $(x, y, \theta)_j^{\text{enc}} = T_j^{\text{enc}}(\mathbf{q}_{-j}, \mathbf{c})$, where both $\mathbf{q}_{-j}$ and $\mathbf{c}$ are learned from images. The function T_j^{enc} is predefined by leveraging the geometry of the system. For example, in the CartPole system, $(q_1, q_2) = (r, \phi)$, and $T_1^{\text{enc}} \equiv (0, 0, 0)$ and $T_2^{\text{enc}}(q_1) = (q_1, 0, 0)$. In the Acrobot system, $(q_1, q_2) = (\phi_1, \phi_2)$, and $T_1^{\text{enc}} \equiv (0, 0, 0)$ and $T_2^{\text{enc}}(q_1, l_1) = (l_1 \sin q_1, l_1 \cos q_1, 0)$.

This transformation can be implemented by a spatial transformer network (STN) [41], i.e., $\tilde{\mathbf{x}}_{i_j} = \text{STN}(\mathbf{x}_{i_j}, \mathcal{T}(T_j^{\text{enc}}(\mathbf{q}_{-j}, \mathbf{c})))$, where $\tilde{\mathbf{x}}_{i_j}$ is the transformed image from $\mathbf{x}_{i_j}$. To sum up, to encode q_j, we use a multilayer perceptron (MLP) that takes $\tilde{\mathbf{x}}_{i_j}$ as input and infers the parameters of the q_j distribution. For a translational coordinate q_j, we have $(q_j^{\text{m}}, \log q_j^{\text{var}}) = \text{MLP}_j^{\text{enc}}(\tilde{\mathbf{x}}_{i_j})$. For a rotational coordinate q_j, we have $(\alpha_j, \beta_j, \log q_j^{\kappa}) = \text{MLP}_j^{\text{enc}}(\tilde{\mathbf{x}}_{i_j})$, where the mean of the von Mises distribution is computed by normalization $(\cos q_j^m, \sin q_j^m) = (\alpha_j, \beta_j)/\sqrt{\alpha_j^2 + \beta_j^2}$. In this way, we can infer $(\mathbf{r}^\tau, \cos\boldsymbol{\phi}^\tau, \sin\boldsymbol{\phi}^\tau)$ from $\mathbf{x}^\tau$. We will use $(\mathbf{r}^0, \cos\boldsymbol{\phi}^0, \sin\boldsymbol{\phi}^0)$ and $(\mathbf{r}^1, \cos\boldsymbol{\phi}^1, \sin\boldsymbol{\phi}^1)$.

5.4 Velocity Estimator

In order to infer the initial state $\mathbf{s}^0 = (\mathbf{r}^0, \cos\boldsymbol{\phi}^0, \sin\boldsymbol{\phi}^0, \dot{\mathbf{r}}^0, \dot{\boldsymbol{\phi}}^0)$, we still need to infer the initial velocity $(\dot{\mathbf{r}}^0, \dot{\boldsymbol{\phi}}^0)$. Here we use a simple first-order finite difference estimator

$$\dot{\mathbf{r}}^0 = (\mathbf{r}^{\text{m}1} - \mathbf{r}^{\text{m}0})/\Delta t, \tag{5.10}$$

$$\dot{\boldsymbol{\phi}}^0 = \big((\sin\boldsymbol{\phi}^{\text{m}1} - \sin\boldsymbol{\phi}^{\text{m}0}) \circ \cos\boldsymbol{\phi}^{\text{m}0} - (\cos\boldsymbol{\phi}^{\text{m}1} - \cos\boldsymbol{\phi}^{\text{m}0}) \circ \sin\boldsymbol{\phi}^{\text{m}0}\big)/\Delta t, \tag{5.11}$$

where $(\mathbf{r}^{\mathrm{m}0}, \cos\boldsymbol{\phi}^{\mathrm{m}0}, \sin\boldsymbol{\phi}^{\mathrm{m}0})$ and $(\mathbf{r}^{\mathrm{m}1}, \cos\boldsymbol{\phi}^{\mathrm{m}1}, \sin\boldsymbol{\phi}^{\mathrm{m}1})$ are the means of the generalized coordinates encoded from the image at time $t = 0$ and $t = \Delta t$.

5.5 Coordinate-aware Decoder

The decoder is a generative model that generate $P(\mathbf{x}|\mathbf{q}) = \mathcal{N}(\hat{\mathbf{x}}, \mathbf{I})$, where $\hat{\mathbf{x}}$ is the reconstruction image of the image data $\mathbf{x}$. We proposed the following coordinate-aware decoder inspired by Jaques et al. [42] to better deal with generalized coordinates. The coordinate-aware decoder first generates a static image $\mathbf{x}_i^c$ of every rigid body i in the system, at a default position and orientation, using a MLP with a nonzero constant input, i.e., $\mathbf{x}_i^c = \mathrm{MLP}_i^{\mathrm{dec}}(1)$. The coordinate-aware decoder then transform the rigid body from a default position into a position and orientation specified by the generalized coordinates to provide $\hat{\mathbf{x}}$. As in Jaques et al. [42], we use the inverse transformation matrix $\mathcal{T}^{-1}((x, y, \theta)_i^{\mathrm{dec}})$ where $\mathcal{T}$ is given by (5.9) and $(x, y, \theta)_i^{\mathrm{dec}} = T_i^{\mathrm{dec}}(\mathbf{q}, \mathbf{c})$. For example, in the CartPole system, $(q_1, q_2) = (r, \phi)$, and $T_1^{\mathrm{dec}}(r) = (r, 0, 0)$ and $T_2^{\mathrm{dec}}(r, \phi) = (r, 0, \phi)$. In the Acrobot system, $(q_1, q_2) = (\phi_1, \phi_2)$, and $T_1^{\mathrm{dec}}(\phi_1) = (0, 0, \phi_1)$ and $T_2^{\mathrm{dec}}(\phi_1, \phi_2) = (l_1 \sin\phi_1, l_1 \cos\phi_1, \phi_2)$. The reconstruction image is then $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n)$, where $\hat{\mathbf{x}}_i = \mathrm{STN}(\mathbf{x}_i^c, \mathcal{T}^{-1}(T_i^{\mathrm{dec}}(\mathbf{q}, \mathbf{c})))$.

5.6 Loss Function

Fig. 5.2 shows the model architecture using CartPole as an illustrative example. The images are fed into the coordinate-aware encoder to infer the initial condition of the state-space model. The state-space model is then integrated to get states at future time steps. The position information in the states at different time step are put into the coordinate-aware decoder to get the reconstruction image. We then minimize the following loss function

$$\mathcal{L}(\mathbf{X}) = \underbrace{-\mathbb{E}_{\mathbf{q}^0\sim Q}[\log P(\mathbf{x}^0|\mathbf{q}^0)] + \mathrm{KL}(Q(\mathbf{q}^0|\mathbf{x}^0)||P(\mathbf{q}^0))}_{\text{VAE loss}} + \underbrace{\sum_{\tau=1}^{T_{\mathrm{pred}}} ||\hat{\mathbf{x}}^\tau - \mathbf{x}^\tau||_2^2}_{\text{prediction loss}} + \underbrace{\lambda \sum_j \sqrt{\alpha_j^2 + \beta_j^2}}_{\text{vM regularization}}. \tag{5.12}$$

The VAE loss is the negative of ELBO (Section 5.2). The prediction loss measures how inaccurate the latent Lagrangian/Hamiltonian dynamics is. The vM regularization with weight λ penalizes large norms of vectors (α_j, β_j), so that these vectors would not blow up during training.

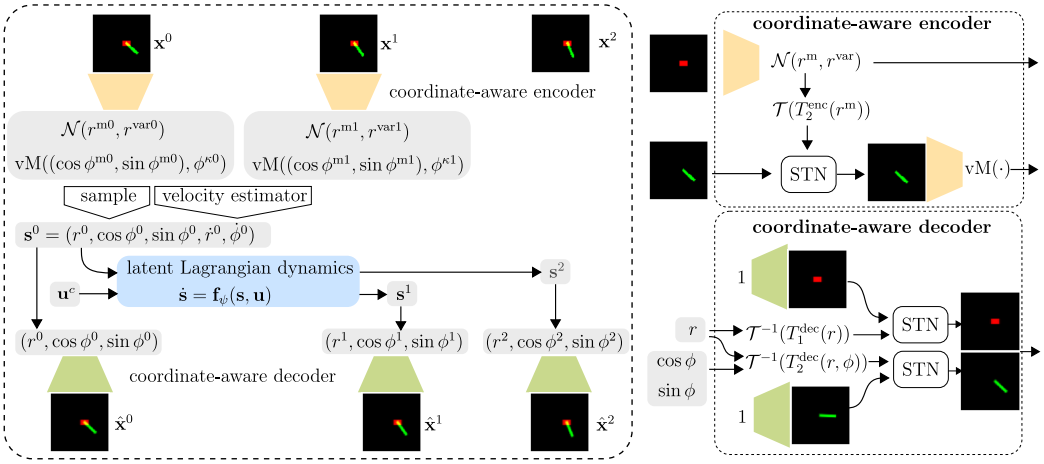

Figure 5.2: **Left**: Model architecture. (Using CartPole as an illustrative example.) The initial state $\mathbf{s}^0$ is constructed by sampling the distribution and a velocity estimator. The latent Lagrangian dynamics take $\mathbf{s}^0$ and the constant control $\mathbf{u}^c$ for that trajectory and predict future states up to T_{pred}. The diagram shows the $T_{\text{pred}} = 2$ case. **Top-right**: The coordinate-aware encoder estimates the distribution of generalized coordinates. **Bottom-right**: The initial and predicted generalized coordinates are decoded to the reconstruction images with the coordinate-aware decoder. Repeated from Fig. 13.1 [99].

5.7 Results

The model is trained on the pendulum, the fully-actuated CartPole and the fully-actuated Acrobot. The images in the training set are generated by OpenAI Gym simulator [13]. The mean square error (MSE) in the image space is not a good metric of measuring prediction accuracy, as indicated by Minderer et al. [57]. Here we show the results of predicting a test image sequence. Fig. 5.3 shows the results of our proposed model, denoted as *Lagrangian+caVAE*. We also compare the results with two model variants: *MLPdyn+caVAE*, which replaces the Lagrangian latent dynamics with MLP latent dynamics, and *Lagrangian+VAE*, which replaces the coordinate-aware VAE with a traditional VAE. The traditional VAE completely fails to reconstruct images of the CartPole system, though it succeeds for the much simpler pendulum system. This indicates that coordinate-aware VAE is crucial to inferring interpretable generalized coordinates. The MLP dynamics fails to learn the dynamics accurate enough to perform long term prediction in both systems. Incorporating Lagrangian/Hamiltonian dynamics helps learn trajectories that preserves energy. At the bottom of the figure, we show the controlled trajectories of the three systems to the inverted position with the designed energy-based controllers. We discuss the details in the next section.

It is important to understand which component in our model contributes the most to learning interpretable dynamics. We design the following four ablations.

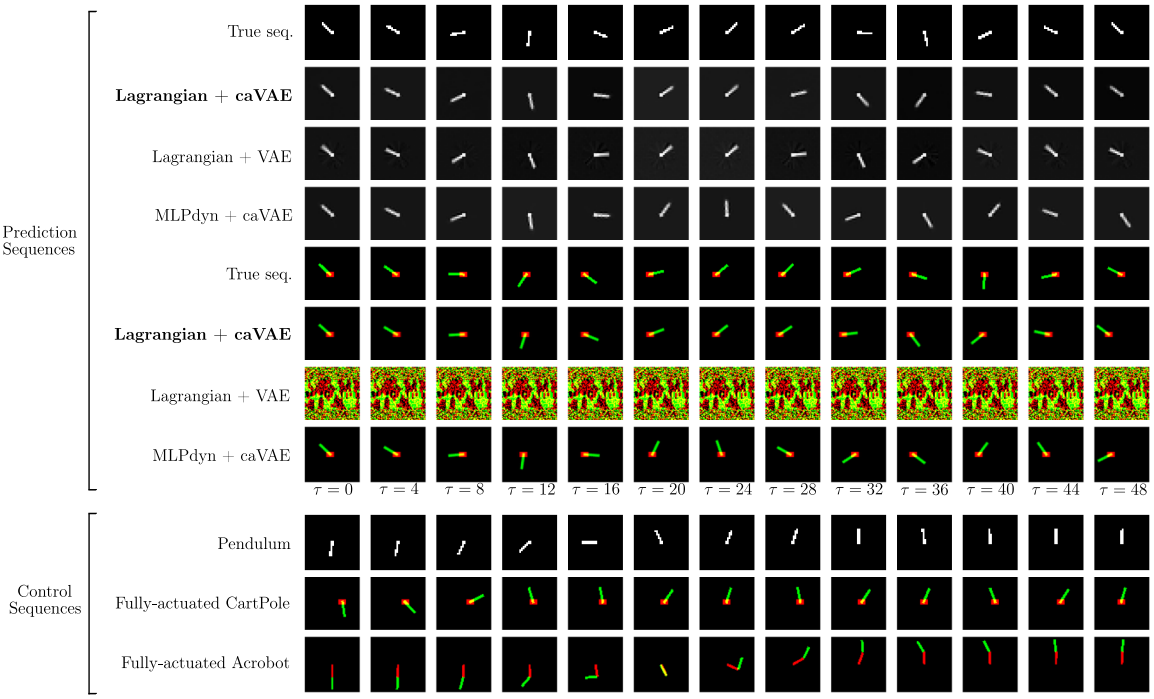

Figure 5.3: **Top**: Prediction sequences of Pendulum and CartPole with a previously unseen initial condition and zero control. Prediction results show both Lagrangian dynamics and coordinate-aware VAE are necessary to perform long term prediction. **Bottom**: Control sequences of three systems. Energy-based controllers are able to control the systems to the goal positions based on learned dynamics and encoding with Lagrangian+caVAE. Repeated from Fig. 13.3 [99].

1. *tradEncoder+caDecoder*: replacing the coordinate-aware encoder with a black-box MLP,
2. *caEncoder+tradDecoder*: replacing the coordinate-aware decoder with a black-box MLP,
3. *caAE*: replacing the coordinate-aware VAE with a coordinate-aware AE,
4. *PAIG*: a Physics-as-inverse-graphics (PAIG) model [42].

The prediction sequences for the ablation models of Pendulum and CartPole are shown in Fig. 5.4. All models can correctly reconstruct pendulum images. Out of all the ablation models, only *tradEncoder+caDecoder* seems to generate realistic CartPole images, but the coordinate values are not well-learned. The Acrobot is a chaotic system so long term prediction is not possible. Fig. 5.5 shows short term reconstruction sequences of the ablation models for the Acrobot. Only *tradEncoder+caDecoder* learns realistic reconstruction. Thus, we conclude that the coordinate-aware decoder has a primary contribution to learning interpretable coordinates and reconstruction images, while the coordinate-aware encoder has a secondary contribution.

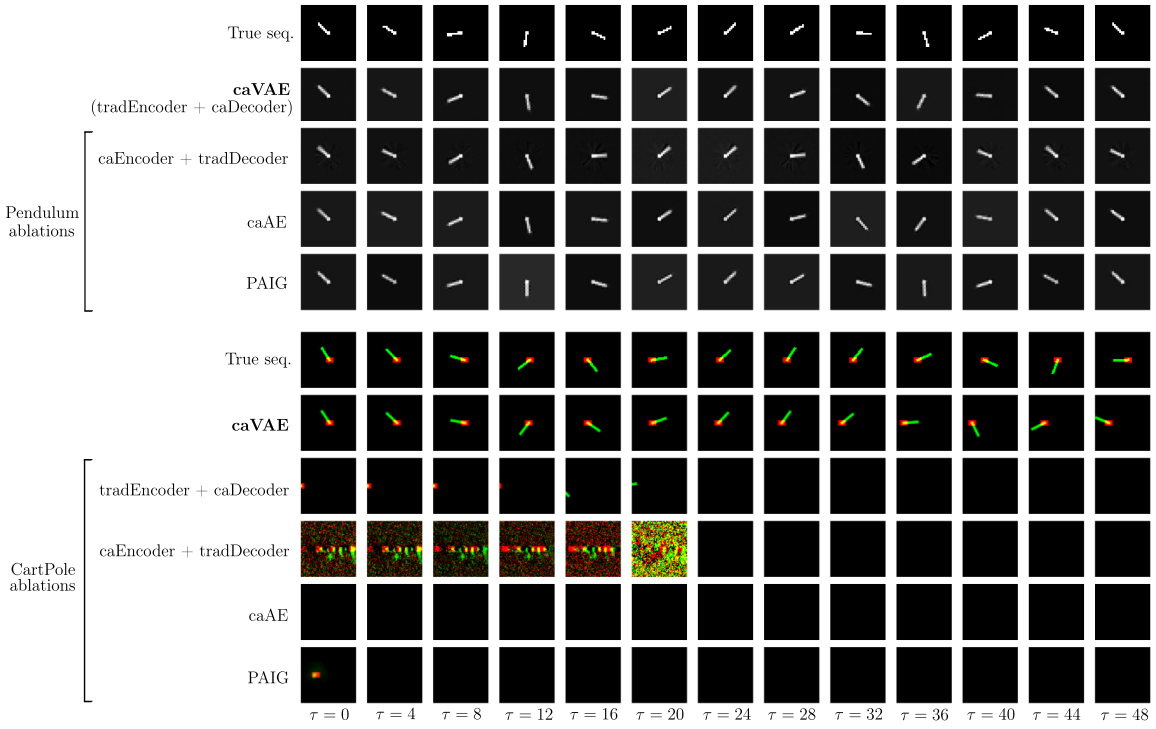

Figure 5.4: Prediction sequences of ablations of Pendulum and CartPole with a previously unseen initial condition and zero control. For the Pendulum experiment, the coordinate-aware encoder is a traditional MLP encoder. All the ablations get good predictions. For the CartPole experiment, all the ablations fail to get good predictions. The *PAIG* is able to reconstruct the cart initially but it fails to reconstruct the pole and make prediction. The *caAE* fails to reconstruct anything. The *caEncoder+tradDecoder* fails to reconstruct meaningful rigid bodies. The *tradEncoder+caDecoder* seems to extract meaningful rigid bodies but it fails to put the rigid bodies in the right place in the image, indicating the coordinates are not well learned. Repeated from Fig. 13.5 [99].

5.8 Interpretability

The key to performing long term prediction is the interpretability of our proposed model. Fig. 5.6 shows the learned potential energy of pendulum, CartPole and Acrobot along with reconstruction images at selected coordinates. The learned potential energy and coordinates are consistent with physics, i.e., when the pendulum/pole/links are in a higher position, the potential energy of the system is higher. This interpretability is a key contribution of our work.

Moreover, the interpretable potential energy allows us to design energy-based controllers (Section 4.8). The control sequences shown in Fig. 5.3 are generated in the following way. First, an image of the goal position $\mathbf{x}^\star$ is provided and it is fed into the coordinate-aware encoder to infer the goal generalized coordinates $\mathbf{q}^\star$. As each time step, the OpenAI Gym simulator of the system takes a control input, integrate one time step forward, and generate an image of the system at the next time step. The control input to the simulator is $\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\dot{\mathbf{q}})$ which is designed as

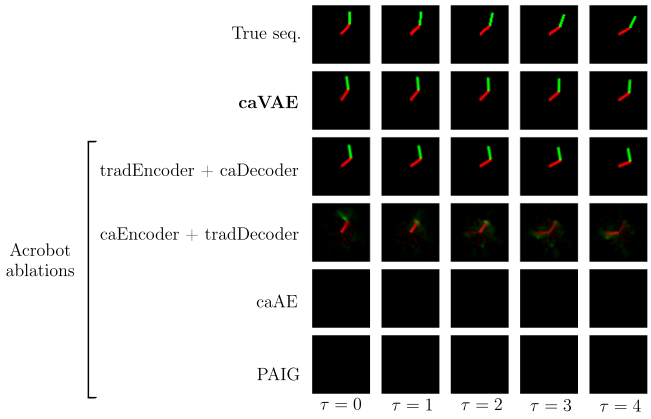

Figure 5.5: Reconstruction image sequences of ablations of Acrobot with a previously unseen initial condition and zero control. The *PAIG* and *caAE* fail to reconstruct anything. The *caEncoder+tradDecoder* fails to reconstruct the green link at all. The *tradEncoder+caDecoder* makes good reconstruction. Repeated from Fig. 13.6 [99].

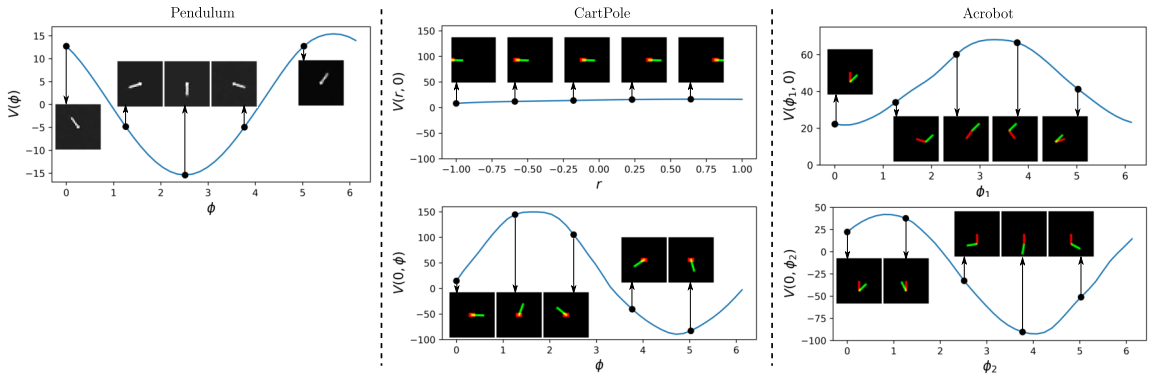

Figure 5.6: Learned potential energy with Lagrangian+caVAE of three systems and reconstruction images at selected coordinates. Both the learned coordinates and potential energy are interpretable. Repeated from Fig. 13.4 [99].

in Section 4.8 with learned potential energy, input matrix, coordinates encoded from the output images and $\mathbf{q}^\star$. From the sequences, we know that we can successfully control the pendulum, fully-actuated CartPole and fully-actuated Acrobot to the inverted positions based on the learned dynamics, coordinates and energy-based controllers.

In this chapter, we studied the problem of learning Lagrangian/Hamiltonian dynamics from image data. As we have shown how to incorporate Lagrangian/Hamiltonian dynamics in the previous chapter, the challenge in this chapter is to infer generalized coordinates from image data in a unsupervised way. Without proper inductive bias, the neural network might learn anything, not necessarily the generalized coordinates, that can help it reconstruct images. One might think using a traditional VAE with a prior on the dynamics would guide the neural network to learn interpretable generalized coordinates. We showed that this is not the case. We showed that additional inductive

biases are required to learn interpretable coordinates. We presented one possible inductive bias by proposing the coordinate-aware VAE. The proposed coordinate-aware VAE takes the geometry of coordinates into account and it is able to learn interpretable coordinates. We showed that the learned potential energy of three systems - the pendulum, the Cartpole and the Acrobot - as a function of generalized coordinates and the reconstruction images at sampled coordinates. We observed that the potential energy and the corresponding configuration of systems in the reconstruction images are consistent with physics. Our approach focuses on interpretability and learns dynamics in a transparent and explainable way. It would be great to extend this kind of approaches to more complex settings, where we can solve challenging problems in an explainable way. This chapter also paves the way for designing engineering multi-agent systems where the activity levels of neighboring agents need to be inferred from the dynamics. For example, in a robot team where each robot is equipped with camera sensors, our method can help a robot to learn the dynamics of other agents based on sensor data in order to infer activity or use the dynamics for downstream tasks such as control.

Chapter 6

Final Remarks

In this dissertation, we analyzed the linear threshold model on multiplex networks with heterogeneous agents. We also generalized the linear threshold model to the continuous threshold model, which can potentially be extended to multiplex networks. To address the scenarios where an agent need to infer activity of other agents from their dynamics, we investigated machine learning of rigid body dynamics from trajectory and image data. We conclude our work and list future directions on cascade dynamics and learning dynamics separately in the following sections.

6.1 Cascade Dynamics

6.1.1 Conclusions

In Chapter 2, we formalize the multiplex LTM and prove its equivalence to the multiplex LEM. We then propose an accurate algorithm to calculate the influence spread by leveraging multiplex LEM. We show the examples with symmetry so that we can write down the analytical expression of the influence spread by multiplex LEM. We also investigate the role of heterogeneity in the tradeoff between sensitivity to a real signal and the robustness to a disturbance. We show that computing influence spread in general is computationally intractable, so we propose to leverage probabilistic inference in Bayesian networks to compute the influence spread approximately. We apply this method on random multiplex networks and again show that Protocol OR enhances cascade while Protocol AND diminishes cascade. In Chapter 3, we generalize the LTM into a continuous dynamics - the CTM. We study the CTM on a family of networks - chains of three clusters. We prove the existence of a pitchfork bifurcation in the dynamics and observe cascade

when the pitchfork is subcritical. We derive the condition for the transition from a supercritical bifurcation to a subcritical bifurcation, thus gives the condition for a cascade. The condition depends on the sizes of the clusters and the disparity in thresholds among the clusters. We show that large disparity in cluster sizes and thresholds lead to a cascade.

Our study on multiplex LTM helps us gain insight into the remarkable phenomenon in animal groups that they are good at distinguishing real threats nearby from the disturbances in the environment. By analyzing this phenomenon using a mathematical model, we are able to apply the mechanism to engineering systems such as a team of robots. Understanding the mechanism also helps us design strategies to control the spread in multi-agent systems, such as controlling the spread of disease in a group of people. The study of spreading using continuous dynamics is usually done with compartmental models, where agents and individual interactions are not explicitly modelled. The CTM we propose models agents and their interactions with a continuous dynamical model. We demonstrate, from a modelling perspective, that a cascade can be related to subcritical bifurcations in the underlying dynamics. When the bifurcation parameter crosses a subcritical bifurcation, the stable branch which the trajectory stays on becomes unstable. Because of the subcritical bifurcation, there are no stable branch in the neighborhood of the trajectory. The trajectory would be attracted to a stable branch far away. This attraction is observed as a quick, switch-like change in the state values, which captures the speed of a cascade observed in biological systems. This sparks new opportunities to investigate spreading dynamics.

6.1.2 Future Directions

Our work on cascade dynamics sparks many exciting future directions. It indicates how heterogeneity and multiple sensing modalities play important roles in the sensitivity and robustness of a multi-agent system. Spreading dynamics on traditional homogeneous networks can be extended in this direction to explore richer behaviors that helps us understand collective behaviors. We list a few future directions below.

Efficient algorithm for general multiplex networks. We have shown that computing influence spread in multiplex networks of which the projection networks are DAGs is computationally intractable. That also means in general the problem of computing influence spread in multiplex networks is computationally complex and intractable. We have proposed to leverage probabilistic inference to compute influence spread efficiently for multiplex networks of which the projection networks are DAGs. For general multiple networks, the problem remains unsolved. The junction

tree algorithm can be used to solve probabilistic inference on general networks but the algorithm requires adding edges to network to form a junction tree. Junction trees are a class of networks with the running intersection property, which is not satisfied by most of the network we study or in applications. It is not clear how to turn the problem of computing influence spread into a probabilistic inference problem in a junction tree. Moreover, even if we can solve the marginalization problem in junction tree, we still need to perform further marginalization within each node in the junction tree. How to leverage junction tree algorithm to compute the influence spread is unclear, not trivial and requires further efforts to investigate.

Control of spread. Our work on cascade dynamics focuses on the mechanisms of spread and provides insights into how cascade happens and how heterogeneity and network structure influence the spread. These insights can help us control the spread of an activity. For example, we might want to increase the number of people wearing a mask during a pandemic in order to slow down the spread of diseases. The cascade centrality we proposed in this dissertation measures the ability of an individual in terms of spreading an activity. It would be most effective to target those agents with high cascade centrality or a group of agents with high influence spread in order to control the spread. For example, letting a person with high cascade centrality wear a mask would influence a large number of people to start wearing a mask. Taking down a video that goes viral but with fake news or rumors would slow down the spread of misleading information. It would be interesting to study how to mitigate or enhance spread in multiplex network in a systematic way.

Cascade dynamics in time-varying networks. Our work assumes the network structure is fixed and this assumption holds true when the network structure remains unchanged within the period of cascade. In some biological and engineering systems, however, the network structure would change while an activity spreads among the agents. For example, an agent's response might change its location in the space, e.g., a fish school fleeing away from a predator. If the sensory network is dependent on spatial locations of agents, the network structure would change. Time-varying graphs have been studied with Markov switching graphs [15]. Similar ideas can be leveraged to study cascade dynamics in time-varying networks. It is interesting to generalize Markov switching graphs into multiplex networks and gain insights into how a spread of an activity would change network structure and how time-varying network structures would influence the spread of an activity.

Refined models for arbitrary network structure. The CTM we have studied explores one possible mechanism for switch-like change of states - subcritical pitchfork bifurcation. By generalizing the LTM to continuous dynamics we hope the great number of tools from continuous dynamical systems, including but not limited to subcritical bifurcations, can be leveraged to understand various kinds of cascade dynamics in natural systems and how animal groups distinguish a real threat from noise and disturbances. Subcritical bifurcations might not be the only mathematical mechanism that exhibits a cascade. Other types of nonlinear dynamical models can be explored to understand the nature of multi-agent biological systems. It would be great to extend the CTM or other types of continuous models to analyze more general network structures beyond the class of chains of three clusters. It would also be exciting to explore continuous cascade dynamics on multiplex networks with heterogeneous agents to better understand the role of distinguishing different sensing modalities.

6.2 Learning Dynamics

6.2.1 Conclusions

In Chapter 4, we study the problem of learning rigid body dynamics from trajectory data, motivated by the scenario of a robot team where the activity level of agents are not directly observable and need to be inferred from the dynamics. The neighboring agents then need to learn the dynamics from trajectory sensor data. We build a neural network model that learns dynamics and control. We show that incorporating physics prior such as Lagrangian/Hamiltonian dynamics benefits prediction accuracy and generalization outside of the training data. Based on the learnt dynamics, we can design energy-based controllers to control planar rigid body system to a desired configuration. In Chapter 5, we assume the sensor data are raw images from cameras and we need to infer both the embedding from images to coordinates and the dynamics on the coordinates at the same time. We propose a coordinate-aware VAE that incorporate the geometry of chosen coordinates and we show that this is crucial to learning meaningful coordinates as compared to a traditional VAE. The coordinate-aware VAE works well with the model we design in the previous chapter to learn embedding and dynamics at the same time. The learned model makes realistic prediction in the image space. As the energy of the systems are directly learned, the interpretable learned energy allows synthesis for energy-based controllers.

Our study on learning dynamics from trajectory and image data narrows the gap between

traditional system identification and black-box neural network approaches. Traditional system identification estimated unknown parameters in the systems based on the data using techniques such as regression. It fails to handle the scenario where the data are raw images. Neural networks are usually treated as black boxes. Even though they perform astonishingly well in certain tasks, it is hard to explain what is going on under the hood. By incorporating physics priors of Lagrangian/Hamiltonian dynamics, we gain interpretability of the neural network approach since we can interpret the learned potential energy. Moreover, the coordinate-aware VAE learns interpretable coordinates by explicitly taking into account the geometry of generalized coordinates. This enables the coordinate-aware VAE to learn coordinates, the change of which over time are captured by Lagrangian/Hamiltonian dynamics. Our methods also pave the way for learning cascade dynamics from data.

6.2.2 Future Directions

Our work on learning dynamics also paves the way for a few exciting future directions. We only explored the learning of planar fully-actuated rigid-body systems and similar methodology can be extended to 3D systems and under-actuated systems. The method can potentially be used to learn cascade dynamics too. Possible future directions are elaborated as follows.

Learning three-dimensional rigid body dynamics. Our work on learning dynamics focuses on planar rigid body systems. The ideas we have built into the model can be extended into three-dimensional rigid body dynamics. From a data-driven perspective, image data might not be sufficient for us to infer three-dimensional rigid body dynamics. Previous research has explored using 3D point cloud data to infer SE(3) dynamics [14]. It would be great to explore how to incorporate Lagrangian or Hamiltonian dynamics to directly infer physics-consistent energy of a 3D rigid body system. It is interesting to compare it with previous methods to see how much data efficiency we gain by incorporating physics priors. Also, the rotation of 3D rigid bodies cannot be modelled by von Mises distributions any more, since the rotation is on the manifold SO(3). One possibility is to leverage the Bingham distribution on the space of quaternions to model 3D rotational coordinates [26].

Learning to control under-actuated systems. From a control perspective, the systems we considered are all fully actuated, so that we are able to control the systems by potential energy shaping. For under-actuated systems, potential energy shaping alone might not be sufficient to perform the

control task. Kinetic energy shaping might also be required. It is unclear how to incorporate kinetic energy shaping into deep learning, but this presents an avenue of future research. It is an interesting and promising direction to explore the control of under-actuated systems based on learned dynamics. The whole framework would enable learning dynamics and control from data and designing safe, interpretable controllers. The black-box nature of neural networks is one of the main reasons neural networks have not yet convinced people to use it in scenarios such as human-robot interactions. Neural network models usually lack theoretical guarantees and deploying them with robots that interact with people might cause harm to people. Hopefully the interpretable approach would enable us to design models that can leverage the vast amount of data and at the same time provide interpretability and guarantees for us to safely deploy them in applications in the physical world.

Design interpretable models for complex mechanical systems. We have tested our models on three simple mechanical systems - the pendulum, the CartPole and the Acrobot. All of these systems have small degrees of freedom and we demonstrated that our frameworks can successfully learn dynamical systems with small degrees of freedom. However, it is unclear if the same framework would face any challenge when applying to more complex systems with large degrees of freedom. For a complex system, more training data might be needed to contain the rich dynamical behaviors of the system. It is not feasible to get data that exhibits the whole range of dynamics for such systems. However, maybe only part of the dynamics is relevant to applications or downstream tasks. Similar philosophy has already shown in the Acrobot system we studied. The Acrobot is a chaotic system so that long term prediction is not possible. In fact, for chaotic systems, it is intuitively impossible to learn the full range of dynamical behaviors from data. In our work we focused on the control of Acrobot to a target position, so that those trajectories in the state space with large speeds are not relevant. When generating training data, we generate data where the speeds are small. Of course, the learned dynamics would not be able to generalize to regions with high speeds but as long as we are concerned about the control of the Acrobot, the learned dynamics are sufficient for us to perform the control task. As for complex systems, the question is how do we design methods and generate training data that can help us learn relevant dynamics efficiently for downstream tasks? This is a challenging problem. Most of the existing methods that work well on more complex systems use large neural networks and incorporate heuristics to restrict the search space. Although they perform well in given tasks, it is hard to decipher why they perform well and why they fails occasionally on some tasks. The famous example in image classification is helpful to

illustrate this point. Although convolution neural networks achieve close-to-human performance in image classification tasks, a small change in pixel values, even not noticeable by human, would trick the learned neural network to classify an image to the wrong category [30]. Why neural networks are so brittle and at the same time so powerful remain to be explored. We believe interpretability would give us insights into the this mystery of neural networks.

Machine learning approaches to modelling cascade dynamics. Eventually we would like to leverage machine learning to study cascade dynamics from data. In the dissertation, we draw inspiration from biological systems and design models to explain observations from biologists. Nowadays with advanced sensors and computer software that tracks animals in videos, we could obtain a large amount of data of animal groups. For example, the sensory network formed by visual cues in fish groups can be extracted from videos [71]. We are able to record the cascade response of animal groups. A natural question is how can we leverage these data and hopefully learn the cascade dynamics from the data? An important theme in this dissertation is that incorporating prior knowledge can benefit learning. One way of incorporating the prior knowledge is to constrain the neural network architecture so that it searches solutions in a smaller domain. In Chapter 4, we constrain the neural networks to approximate Lagrangian/Hamiltonian dynamics so that the solutions are searched within the class of Lagrangian/Hamiltonian dynamics. This raises the question of how to choose a class of dynamics to incorporating into learning cascade dynamics so that we can use less data to learn accurate cascade dynamics? These are interesting and important questions that require further efforts to explore.

Part II

Papers

Chapter 7

Overview

Part II of this dissertation contains four published peer-reviewed papers and two papers that have been submitted for publication. Only minor modifications regarding formatting are present between the published or submitted articles and the papers presented here.

7.1 Outline

Each paper is organized into a chapter as follows.

- Chapter 8 [100] presents a generalization of the linear threshold model to two-layer multiplex networks and proposes protocols that synthesize information from different layers in the network. The analysis is performed by generalizing the live-edge model and reachability from monoplex network to the two-layer multiplex networks. The results show that two-layer multiplex networks with homogeneous agents exhibit richer behaviors than monoplex networks. In particular, Protocol OR enhances a spread and Protocol AND diminishes a spread.

- Chapter 9 [103] generalizes the linear threshold model to multiplex networks with heterogeneous agents. The reachability in the live-edge model is generalized to $\mathcal{U}$-reachability in multiplex networks. The main theorem in this chapter shows the equivalence of the linear threshold model and live-edge model in the multiplex setting. Based on the theorem, a direct algorithm is proposed to calculate the cascade centrality of agents in the network. An efficient algorithm based on probabilistic inference in Bayesian network is also proposed to approximately compute cascade centrality in large networks.

- Chapter 10 [98] presents a continuous threshold model (CTM) of cascade dynamics, which is motivated by the linear threshold model. Each agent has a real-valued state that changes continuously and a threshold between 0 and 1. The cascade dynamics are studied with the CTM on a network with three clusters. The agents are heterogeneous in thresholds. The analysis shows that the dynamics exhibit a pitchfork bifurcation. Moreover, if the pitchfork is supercritical, the response of agents will be contained, whereas if the bifurcation is subcritical, there will be a cascade. It is shown that a large enough disparity in the thresholds and network structure results in a cascade.

- Chapter 11 [101] addresses the problem of how to learn Hamiltonian dynamics from trajectory data of physical systems. With the prior knowledge of Hamiltonian dynamics, the proposed Symplectic ODE-Net model predicts energy-conserved trajectories and outperforms a baseline model which does not incorporate Hamiltonian dynamics. The model also paves the way for energy-based controllers since energy is directly learned.

- Chapter 12 [102] generalizes Symplectic ODE-Net to model physical systems with dissipation. Besides the mass, the potential energy and the control input matrices, a dissipation matrix that accounts for possible dissipation in the system is jointly learned. This generalization works well in several systems and energy-based controllers can be designed with those fully-actuated systems.

- Chapter 13 [99] presents an approach of unsupervised learning of Lagrangian dynamics from images for prediction and control. A coordinate-aware variational autoencoder is proposed to extract interpretable coordinates from images. A dynamical model on the coordinates incorporates Lagrangian dynamics to perform long-term prediction. The ablation study shows that both components are important in learning interpretable coordinates and making accurate long-term prediction. The model also allows synthesizing energy-based controllers to control fully-actuated physical systems to a goal position based on an image of that goal position.

7.2 Author Contributions

I am the lead author and lead contributor to the mathematical analysis, illustration, simulations, software development and writing presented in all six papers. My advisor, Professor Naomi Ehrich Leonard, advised me on almost all aspects of the research. I list specific author contributions below

- In Chapter 8, Vaibhav Srivastava provided Lim et al. [55] to me, which guided my initial questions and analysis. Naomi E. Leonard, Vaibhav Srivastava and I framed the main questions. I developed the mathematical model with the guidance of Naomi E. Leonard and Vaibhav Srivastava.

- In Chapter 9, I developed the extended models and performed analysis. Vaibhav Srivastava suggested to map the problem of calculate cascade centrality into a problem of marginalization on Bayesian network based on the results in Nguyen and Zheng [64]. Naomi E. Leonard suggested to get analytical results on networks with symmetry. Naomi E. Leonard and Vaibhav Srivastava provided suggestions of the structure of the manuscript. I wrote the initial draft and both Naomi E. Leonard and Vaibhav Srivastava revised and edited the draft.

- In Chapter 10, Naomi E. Leonard suggested the ideas of modelling switch-like behavior with continuous dynamical systems and suggested to look at the 12-agent example from [32] and study a more general class - chains of three clusters. I mapped the LTM to the CTM, inspired by the dynamics from [32]. I performed the bifurcation analysis on chains of three clusters. Naomi E. Leonard provided helpful suggestions of designing a feedback controller. I wrote the draft and Naomi E. Leonard helped reconstructured and edited the draft.

- In Chapter 11, I got the high-level idea of incorporating physics prior into deep learning from discussions with Naomi E. Leonard and Anirudha Majumdar. Biswadip Dey suggested using the port-Hamiltonian dynamics to add control components into the Hamiltonian dynamics. I developed the algorithms, wrote computer programs, and implemented the model on three dynamical systems. Biswadip Dey suggested using energy-based controllers to control the systems based on the learned models. I implemented the controllers. I wrote the initial draft and edited the draft together with Biswadip Dey. Amit Chakraborty supervised and provided supports for the work.

- In Chapter 12, I extended the work in Chapter 11 to include dissipation in the model. I ran all the experiments and wrote the manuscript. Biswadip Dey revised the manuscript. Amit Chakraborty supervised and provided supports for the work.

- In Chapter 13, I proposed to use a coordinate-aware VAE to learn dynamics from images. I developed the framework and ran all the experiments. I wrote the draft and Naomi E. Leonard provided invaluable suggestions on how to present the idea clearly and revised the draft.

Chapter 8

On the Linear Threshold Model for Diffusion of Innovations in Multiplex Social Networks

Yaofeng Desmond Zhong, Vaibhav Srivastava, Naomi Ehrich Leonard

Appears as Zhong et al. [100] in *2017 IEEE 56th Conference on Decision and Control*

Diffusion of innovations in social networks has been studied using the linear threshold model. These studies assume monoplex networks, where all connections are treated equally. To reflect the influence of different kinds of connections within social groups, we consider multiplex networks, which allow multiple layers of connections for a given set of nodes. We extend the linear threshold model to multiplex networks by designing protocols that combine signals from different layers. To analyze these protocols, we generalize the definition of live-edge models and reachability to the duplex setting. We introduce the live-edge tree and with it an algorithm to compute cascade centrality of individual nodes in a duplex network.

8.1 Introduction

In multi-agent network models, nodes represent agents and edges represent sensing, communication or physical connections among agents. Typically, the network has a single layer of connections, where each connection refers to the same kind of sensing, communication or interaction. In real

networks, however, there can be more than one mode of sensing, communication, or interaction among agents. For example, a group of friends or colleagues may be connected both through face-to-face interactions as well as through social media. Individuals in a crowd may be able to see people standing in front of them but may be able to hear people standing behind them. Here, we propose studying dynamics on *multiplex networks*, which allow multiple layers of connections for a given set of nodes [12].

Diffusion of innovations refers to the dynamic spread of an idea or activity. Young [97] discussed three approaches to modeling diffusion of innovation in networks: (i) dynamics in which agents adopt or reject an innovation deterministically by comparing the fraction of their neighbors that have adopted the innovation with a potentially random threshold; (ii) dynamics in which agents adopt or reject innovation probabilistically based on a coordination game played on the network; (iii) dynamics that allow network structure itself to evolve. The model from the first approach is sometimes referred to as the linear threshold model (LTM).

The LTM was first introduced in [31, 80] and its various applications including, riot behavior, voting, and migration were discussed. Watts [92] used the LTM to explain cascades in random networks that are triggered by small initial shocks. Kempe et al. [45] and Lim et al. [55] studied diffusion of innovations in standard single-layer (monoplex) networks using a LTM with randomly drawn thresholds. Como et al. [19] studied LTM in large scale networks using mean-field approximation and associated bifurcations. Lelarge [52] studied diffusion in random network using LTM and identified conditions for widespread adoption of innovation in the network. The LTM is discrete-time network dynamics, in which a set of agents S_0 is initially active (has an innovation) and over time other agents become active (adopt an innovation) if a sufficiently large number of their neighbors are active (number is above a threshold). In [45] an equivalence is established between the LTM and a live-edge process, This live-edge process, referred to as live-edge model (LEM) in this paper, can be studied without temporal iteration. In the LEM, one edge among incoming edges is randomly selected for each agent, and connectivity of the agent with S_0 is examined.

The LEM is used to evaluate the *social influence* of a set of agents S_0, defined in [45] as the expected number of active agents after iterating over the initially active set S_0. In [55] the social influence of a single active agent is called *cascade centrality* and a closed-form expression is provided using the LEM. This expression requires enumerating every path between the single active agent and the rest of the agents in the network. Acemoglu et. al. [1] studied the LTM for deterministic thresholds at each node; in this case, analysis of the LTM becomes very challenging and has limited analytic tractability. Yağan and Gligor [94] proposed a LTM for multiplex networks in which each

node computes the weighted average fraction of its active neighbors in each layer and compares it to a randomly drawn threshold. They analyzed their model for random multiplex networks.

The problem of finding the set of k agents in a monoplex network that maximizes social influence in the LTM was proved to be NP-hard in [45], and approximations were used. Nguyen and Zheng [64] studied the same problem using the independent cascade model [45]. They designed efficient algorithms to approximate social influence by casting the problem as statistical inference in a Bayesian network.

In this paper, we introduce the weighted linear threshold model with thresholds chosen uniformly at random in $[0, 1]$ for fixed multiplex networks and define protocols to combine inputs from different layers. We specialize to duplex networks (two-layer multiplex networks) and derive tools to analyze cascade dynamics. We make the following contributions for duplex networks. First, we define and analyze two LTM protocols: Protocol OR and Protocol AND, which describe a sensitive and a conservative response, respectively, to active neighbors. Second, we generalize the live-edge model and introduce the notion of reachability to duplex networks, and prove equivalence of the duplex LEM to the duplex LTM. Third, we introduce the *live-edge tree*, a representation of the network topology, to compute reachability in the duplex LEM. Fourth, we define and provide an algorithm to compute duplex cascade centrality.

In Section 8.2, we define multiplex networks and graph-related properties. In Section 8.3, we define the multiplex LTM and propose protocols for the duplex LTM. In Section 8.4, we generalize the LEM to duplex networks and define reachability corresponding to the duplex LTM protocols. In Section 8.5, we prove equivalence of the duplex LTM and LEM. In Section 8.6, we generalize social influence and cascade centrality to duplex networks and illustrate with an example. We conclude in Section 8.7.

8.2 Multiplex Networks

A *multiplex network* $\mathcal{G}$ is a family of $m \in \mathbb{N}$ directed weighted graphs $G_1, ..., G_m$. Each graph $G_k = (V, E^k)$, $k = 1, \ldots, m$, is referred to as a layer of the multiplex network. The agent set $V = 1, 2, 3, ..., n$ is the same in all layers. The edge set of layer k is $E^k \subseteq V \times V$ and can be different in different layers. Each edge $e^k_{u,v} \in E^k$, pointing from u to v in layer k, is assigned a weight $w^k_{u,v} \in \mathbb{R}^+$. Agent u is said to be an *in-neighbor* of agent v in layer k if $e^k_{u,v}$ exists. We denote the set of in-neighbors of v in layer k as N^k_v. For an agent v, we say that the weight of its in-neighbor u in layer k is the weight of the edge connecting them, i.e. $w^k_{u,v}$. We assume the weights of all in-neighbours

of an agent sum up to 1, i.e. $\sum_{u \in N_v^k} w_{u,v}^k = 1$ for any agent v.

For undirected graphs, every edge is modeled with two opposing directed edges. For unweighted graphs, every edge $e_{u,v}^k$ can be assigned a weight $w_{u,v}^k = 1/d_v^k$, where d_v^k is the *in-degree* (the number of in-neighbors) of node v.

A *projection network* [12] of $\mathcal{G}$ is the graph $\text{proj}(\mathcal{G}) = (V, E)$ where $E = \cup_{k=1}^m E^k$.

8.3 The Linear Threshold Model

The linear threshold model (LTM) is described by a discrete-time dynamical system where the state of each agent at iteration t is either active (on) or inactive (off). At iteration $t = 0$, all agents are inactive except an initial active set S_0 called seeds. The active state spreads through the network following the rules introduced below. Once an agent is active, it remains active. Let S_t be the set of agents that are active by the end of iteration t. S_t reaches a steady state when $S_{t-1} = S_t$. For n agents, steady state is reached by $t \leq n$.

8.3.1 Monoplex LTM

In a monoplex network, each agent $v = 1, \ldots, n$ chooses a threshold μ_v randomly and independently from a uniform distribution $U(0, 1)$. An inactive agent v becomes active at iteration t if the sum of weights of its active in-neighbors at $t - 1$ exceeds μ_v, that is, if $\mu_v < \sum_{u \in N_v \cap S_{t-1}} w_{u,v}$.

8.3.2 Multiplex LTM

In a multiplex network with m layers, each agent v chooses a threshold μ_v^k in each layer k for $v = 1, \ldots, n$ and $k = 1, \ldots, m$. Each μ_v^k is randomly and independently drawn from $U(0, 1)$. Each agent might have different neighbors in different layers. If the sum of weights of active in-neighbors of v in layer k exceeds μ_v^k, we say agent i receives a positive input from layer k. Otherwise, the input is negative. The inputs that an agent receives can be conflicting, so an inactive agent needs a protocol to make one decision, either to become active or not. We propose two protocols for duplex (two-layer) networks:

Protocol OR: an inactive agent v becomes active at iteration t if it receives a positive input from *either* layers at $t - 1$;

Protocol AND: an inactive agent v becomes active at iteration t if it receives positive inputs from *both* layers at $t - 1$.

Protocol OR models agents that become active more readily, whereas Protocol AND models agents that are more conservative in their decisions to become active.

8.4 The Live-edge Model and Reachability

We define the live-edge model (LEM) proposed in [55] for monoplex directed weighted networks in Section 8.4.1. In Section 8.4.2 we generalize the LEM to duplex networks and introduce two notions of reachability on the duplex LEM: Reachability OR and Reachability AND.

8.4.1 Monoplex LEM and Reachability

The LEM is defined as follows. Given a set of seeds S_0, each unseeded agent randomly selects one incoming edge among all of its incoming edges; an edge is selected with probability given by its weight. The selected edge is labeled as “live", while the unselected edges are labeled as “blocked". The seeds block all of their incoming edges. Every directed edge will thus be either live or blocked. Because the selection of edges can be done at the same time for every node, the LEM can be viewed as a static model. The LEM can alternatively be treated as an iterative process in the case the live edges are selected sequentially.

A *live-edge path* [45] is a directed path that consists only of live edges. If there exists a live-edge path from any of the seeds u to an unseeded agent v, we say v is *reachable* from u by a live-edge path.

8.4.2 Duplex LEM and Reachability

In a duplex network, each unseeded agent v randomly selects one incoming edge $e^1_{u,v}$ in layer 1 with probability $w^1_{u,v}$ and one incoming edge $e^2_{w,v}$ in layer 2 with probability $w^2_{w,v}$. The selected edges are labeled as “live", while the rest are labeled as “blocked". The seeds block all of their incoming edges in both layers. We will refer to such labeling process as *a selection of live edges*.

The challenge in generalizing the LEM to multiplex networks is to properly define reachability. Here we introduce the *live-edge tree* representation of a duplex network, which we will use to define two notions of reachability corresponding to the two duplex LTM protocols.

Definition 9. *Given a set of seeds S_0 and a selection of live edges, a* live-edge tree *associated with agent v is a tree that satisfies*

1. *Every node in the tree corresponds to an agent in the duplex network $\mathcal{G}$. The root corresponds to agent v;*

2. *For each parent p in the tree, p's left (respectively, right) child is the agent to which p's live edge in layer 1 (respectively, 2) is connected.*

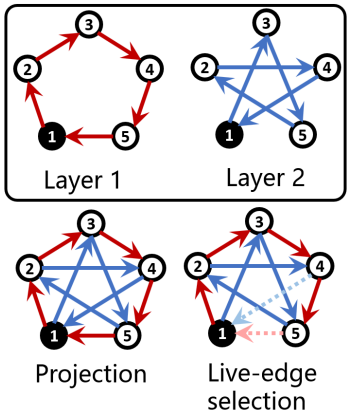

Figure 8.1: A duplex network. Blocked edges are light dashed arrows.

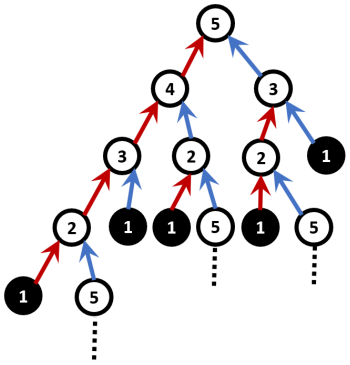

Figure 8.2: The live-edge tree for agent 5 in the example duplex network.

Different nodes in the live-edge tree can be the same agent in the duplex network, and branches of the tree can have infinite length. Figure 8.1 shows an example of duplex network with seed (1) and a selection of live edges. For this example, there is only one possible selection of live edges, since each unseeded agent has only one in-neighbour in each layer. Figure 8.2 shows the live-edge tree associated with (5). Some branches end with (1), the others come back to (5) again. The structure under (5) is the same as the structure shown in the figure, so we use dashed lines to show this repeated information. In this tree, some branches are infinite.

Now we are ready to propose two reachability definitions:

Reachability OR: for a given selection of live edges and a set S_0, an agent v is reachable from S_0

by a selection of live edges if the live-edge tree associated with v has at least one finite branch, and every finite branch ends with a seed;

Reachability AND: for a given selection of live edges and a set S_0, an agent v is reachable from S_0 by a selection of live edges if all branches of the live-edge tree associated with v are finite, and every branch ends with a seed.

Following these definition, the live-edge tree in Figure 8.2 shows that (5) is reachable from (1) under Reachability OR, but not under Reachability AND. A branch is infinite if an agent reappears in the branch. This simple condition can verify an infinite branch in the algorithm.

8.5 Equivalence of LEM and LTM

The monoplex LEM was introduced in [45] and proved to be equivalent to the monoplex LTM in the sense that the probability distributions of agents being reachable from a set S_0 in the LEM are equal to the probability distributions of agents being active at steady state after iterating over the set S_0 in the LTM. Computing these probability distributions for the LTM is challenging because it requires solving over the temporal iterations. However, leveraging the equivalence, the probability distributions can be computed without temporal iteration using the LEM treated as a static model.

Recall that S_t is the set of active agents at the end of iteration t for the LTM. S_t reaches steady state when $S_t = S_{t-1}$, and this takes no longer than n iterations.

To prove the equivalence, the LEM can also be treated as an iterative process by revealing the reachabilities of live edges gradually as follows [45]. From an initial set S'_0, check the reachability of the agents with at least one edge coming from S'_0. If an agent is determined to be reachable from S'_0, add it to S'_0 at the end of the iteration to get a new set S'_1. In the next iteration, follow the same procedure and get a sequence of sets $S'_0, S'_1, S'_2, \ldots$. The process ends at iteration t if $S'_t = S'_{t-1}$.

In this section, we first show the equivalence for the monoplex case whose proof can be found in [45]. We then prove the equivalence for the duplex case.

8.5.1 Monoplex Networks

Proposition 5. *[45] For a given set S_0, the probabilities of the following events regarding an arbitrary unseeded agent v are the same:*

1. *v is active by running the LTM under random thresholds given initial active set S_0;*
2. *v is reachable from set S_0 by live-edge paths under the random selection of live edges in the LEM.*

8.5.2 Duplex Networks: Protocol OR and Reachability OR

Lemma 4. *Given an initial active set S_0 and a selection of live edges, consider an agent i_0. Assume its live edge in layer 1 comes from agent i_1^1, and its live edge in layer 2 comes from agent i_1^2. Then i_0 is reachable from S_0 under Reachability OR if and only if either i_1^1 or i_1^2 is reachable from S_0 under Reachability OR.*

Proof. If i_1^1 is reachable from S_0 under Reachability OR, then there exists a finite branch in the live-edge tree associated with i_1^1. Denote the branch as $P_{i_1^1} = (i_1^1, i_2^1, i_3^1, ..., i_n^1)$, where $i_n^1 \in S_0$. Then the live-edge tree associated with i_0 has a finite branch $P_{i_0}^1 = (i_0, i_1^1, i_2^1, ..., i_n^1)$, which means i_0 is reachable from S_0 under Reachability OR. Following similar analysis for i_1^2, we prove the "if" part.

If neither i_1^1 nor i_1^2 is reachable under Reachability OR, there is no finite branch in their live-edge trees that end with the seeds. Consequently, there is no finite branch in the live-edge tree associated with i_0. i_0 is not reachable from S_0 under Reachability OR. This proves the "only if" part. □

In the following, we prove that the LTM under Protocol OR is equivalent to the LEM under Reachability OR.

Proposition 6. *For a given set S_0, the probabilities of the following events regarding an arbitrary unseeded agent v are the same:*

1. *v is active at steady state by running the LTM under Protocol OR given the initial active set S_0;*
2. *v is reachable from the set S_0 defined by Reachability OR by running the LEM.*

Proof. We prove by mathematical induction.

First, we define some events regarding the LTM. Let $X_k := \mu_v^k < \sum_{u \in N_v^k \cap S_t} w_{u,v}^k$ and $Y_k := \mu_v^k \geq \sum_{u \in N_v^k \cap S_{t-1}} w_{u,v}^k$, for $k \in \{1, 2\}$.

In the LTM, if agent v has not become active at the end of iteration t, then we denote the probability that it becomes active in iteration $t+1$ as P_v^{t+1}. In this case, both μ_v^1 and μ_v^2 have not been exceeded at the end of iteration t. Then the probability that μ_v^1 is exceeded in iteration $t+1$ is $P_1 = P(X_1|Y_1, Y_2) = P(X_1|Y_1)$. The last equality holds because random variables μ_v^1 and μ_v^2 are independent. Similarly, the probability that μ_v^2 is exceeded in iteration $t+1$ is $P_2 = P(X_2|Y_2)$. Using the inclusion-exclusion principle, we have that $P_v^{t+1} = P_1 + P_2 - P_1 \times P_2$.

Then we define some events regarding the LEM. We denote $f_v^k(t)$ as the event that v's live edge in layer k comes from reachable set by the end of iteration t. We denote $g_v^k(t)$ as v's live edge in layer k does not come from reachable set by the end of iteration t. Then we let $X_k' = f_v^k(t+1)$ and $Y_k' = g_v^k(t)$, for $k \in \{1, 2\}$.

We look at the LEM as an iterative process. If agent v has not become reachable at the end of iteration t, then we denote the probability that it becomes reachable in iteration $t+1$ as $P_v^{'t+1}$. In this case, the probability that v's live edge in layer 1 comes from S_t' in iteration $t+1$ is $P_1' = P(X_1'|Y_1', Y_2') = P(X_1'|Y_1')$. Similarly, the probability that v's live edge in layer 2 comes from S_t' in iteration $t+1$ is $P_2' = P(X_2'|Y_2')$. Then the probability that either v's live edge in layer 1 comes from S_t' or v's live edge in layer 2 comes from S_t' in iteration $t+1$ is $P_v^{'t+1} = P_1' + P_2' - P_1' \times P_2'$. By Lemma 4, we conclude that $P_v^{'t+1}$ is the probability of reachability of node v under Reachability OR.

Similar to [45], we can see that $P_1 = P_1'$ and $P_2 = P_2'$, so we have $P_v^{t+1} = P_v^{'t+1}$. Thus, by induction over the iterations, we have proved that the probabilities of the two events are the same. □

8.5.3 Duplex Network - Protocol AND and Reachability AND

Lemma 5. *Given an initial active set S_0 and a selection of live edges, consider an agent i_0. Assume its live edge in layer 1 (respectively, layer 2) comes from agent i_1^1 (respectively, i_1^2). Then i_0 is reachable from S_0 if and only if both i_1^1 and i_1^2 are reachable from S_0 under Reachability AND.*

Proof. Since i_1^1 and i_1^2 are reachable from S_0 under Reachability AND, their live-edge trees do not have any infinite branch. In the live-edge tree associated with i_0, i_0 has two children: i_1^1 and i_1^2. Since the branches under i_1^1 and i_1^2 are all finite, the live-edge tree associated with i_0 has no infinite branch. Since all the leaves in the tree are the union of leaves of the live-edge trees associated with i_1^1 and i_1^2, all of them are seeds. We conclude that i_0 is reachable from S_0 under Reachability AND. If i_1^1 or i_1^2 are not reachable under Reachability AND, then the infinite branch will result in an infinite branch in the live-edge tree associated with i_0, or a branch end with unseeded agent will result in a branch end with unseeded agent in the live-edge tree associated with i_0, which means i_0 is not reachable. □

We next prove that the LTM under Protocol AND is equivalent to the LEM under Reachability AND.

Proposition 7. *For a given set S_0, the probabilities of the following events regarding an arbitrary unseeded agent v are the same:*

1. *v is active at steady state by running the LTM under Protocol AND given the initial active set S_0;*
2. *v is reachable from the set S_0 defined by Reachability AND by running the LEM.*

Proof. We prove by mathematical induction.

In addition to X_k and Y_k in the previous proof, we let $Z_k := \mu_v^k < \sum_{u \in N_v^k \cap S_{t-1}} w_{u,v}^k$, for $k \in \{1, 2\}$.

In the LTM, if agent v has not become active at the end of iteration t, then we denote the probability that it becomes active in iteration $t+1$ as P_v^{t+1}. If v becomes active at the end of iteration $t+1$, X_1 and X_2 must both be true. If v is inactive at the end of iteration t, then at least one of the thresholds is not exceeded, for which there are three possibilities. The three corresponding probabilities are $P_1 = P(X_1, X_2|Y_1, Y_2)$, $P_2 = P(X_1, X_2|Z_1, Y_2)$ and $P_3 = P(X_1, X_2|Y_1, Z_2)$. As they are independent of one another, we have $P_v^{t+1} = P_1 + P_2 + P_3$.

In addition to X_k' and Y_k' in the previous proof, we let $Z_k' = f_v^k(t)$, for $k \in \{1, 2\}$.

We look at the LEM as an iterative process. If agent v has not become reachable by the end of iteration t, then we denote the probability that it becomes reachable in iteration $t+1$ as $P_v^{'t+1}$. By Lemma 5, if v is reachable by the end of iteration $t+1$, v's live edges in both layers must come from reachable set by the end of iteration $t+1$. Moreover, at the end of iteration t at least one of v's live edges has not come from the reachable set, for which there are three possible cases. The probabilities of the three cases are $P_1' = P(X_1', X_2'|Y_1', Y_2')$, $P_2' = P(X_1', X_2'|Z_1', Y_2')$ and $P_3' = P(X_1', X_2'|Y_1', Z_2')$. As they are independent of one another, we have $P_v^{'t+1} = P_1' + P_2' + P_3'$.

Similar to [45], we can show that $P(X_1|Y_1) = P(X_1'|Y_1')$ and $P(X_2|Y_2) = P(X_2'|Y_2')$ then we claim $P_i = P_i', i = 1, 2, 3$:

$$\begin{aligned} P_1 &= P(X_1, X_2, Y_1, Y_2)/P(Y_1, Y_2) \\ &= P(X_1, Y_1)/P(Y_1) \times P(X_2, Y_2)/P(Y_2) \\ &= P(X_1', Y_1')/P(Y_1') \times P(X_2', Y_2')/P(Y_2') \\ &= P(X_1', X_2', Y_1', Y_2')/P(Y_1', Y_2') = P_1'; \end{aligned}$$

$$\begin{aligned} P_2 &= P(X_1, X_2, Z_1, Y_2)/P(Z_1, Y_2) \\ &= P(X_1, Z_1)/P(Z_1) \times P(X_2, Y_2)/P(Y_2) \\ &= 1 \times P(X_2, Y_2)/P(Y_2) \\ &= 1 \times P(X_2', Y_2')/P(Y_2') \\ &= P(X_1', Z_1')/P(Z_1') \times P(X_2', Y_2')/P(Y_2') \\ &= P(X_1', X_2', Z_1', Y_2')/P(Z_1', Y_2') = P_2'; \end{aligned}$$

$$
\begin{aligned}
P_3 &= P(X_1, X_2, Y_1, Z_2)/P(Y_1, Z_2) \\
&= P(X_1, Y_1)/P(Y_1) \times P(X_2, Z_2)/P(Z_2) \\
&= P(X_1, Y_1)/P(Y_1) \times 1 \\
&= P(X_1', Y_1')/P(Y_1') \times 1 \\
&= P(X_1', Y_1')/P(Y_1') \times P(X_2', Y_2')/P(Z_2') \\
&= P(X_1', X_2', Y_1', Z_2')/P(Y_1', Z_2') = P_3'.
\end{aligned}
$$

Then we can show $P_v^{t+1} = P_v'^{t+1}$. Thus, by induction over the iterations, we see that the probabilities of the two events are the same. □

8.6 Social Influence and Cascade Centrality

8.6.1 Monoplex Social Influence and Cascade Centrality

The *social influence* of a set of agents S_0 is defined as the expected number of active agents at the steady state of the LTM given that S_0 is the initial active set [45]. This measure of social influence for a single agent as the set S_0 is called *cascade centrality* in [55].

8.6.2 Duplex Social Influence and Cascade Centrality

Duplex social influence and *duplex cascade centrality* can be defined analogously for each protocol of the duplex LTM. We define duplex cascade centrality under Protocol OR and under Protocol AND.

Definition 10. *The* duplex cascade centrality of agent v under Protocol OR *is the expected number of active agents at steady state of the duplex LTM under Protocol OR, given v is the only seed in the network.*

Definition 11. *The* duplex cascade centrality of agent v under Protocol AND *is the expected number of active agents at steady state of the duplex LTM under Protocol AND, given v is the only seed in the network.*

The expected number of active agents in the network is the sum of the probabilities of being active over the agents in the network. Calculating this probability distribution with the LTM requires doing simulations under different combinations of threshold values of all agents. However, by Propositions 6 and 7, the LEM gives us a way to calculate cascade centralities from the duplex network structure.

8.6.3 Algorithm for Duplex Cascade Centralities

The following algorithm is not intended to be efficient, but it serves as a way to accurately calculate duplex cascade centralities by leveraging the LEM.

Algorithm 3. *Calculate duplex cascade centralities*

1. *Find the D different selections of live edges, where $D = \prod_{j\in V\setminus\{i\}} d_j^1 \prod_{j\in V\setminus\{i\}} d_j^2$.*
2. *For each selection, construct the live-edge tree for each unseeded agent. Store reachability results under the two reachability definitions for each unseeded agent.*
3. *If agent j is reachable N_j^{OR} times under Reachability OR and N_j^{AND} times under Reachability AND, then the duplex cascade centralities of agent i are*
   $C_i^{OR} = 1 + \sum_{j\in V\setminus\{i\}} N_j^{OR}/D$,
   $C_i^{AND} = 1 + \sum_{j\in V\setminus\{i\}} N_j^{AND}/D$

Theorem 5. *The C_i^{OR} and C_i^{AND} computed by Algorithm 1 are the duplex cascade centrality of agent i under Protocol OR and Protocol AND, respectively.*

Proof. If follows directly from the equivalence of the LTM and the LEM. □

Using Algorithm 1, the two centralities can be calculated at the same time, whereas if we conduct simulations by the LTM, we must simulate the two protocols separately. From the live-edge tree associated with (5) in the example of Figure 8.2, we actually obtain the live-edge trees of all unseeded agents as they are part of this tree. More generally, we might not need to construct live-edge trees for all unseeded agents. For the example $C_1^{\text{OR}} = 5$ and $C_1^{\text{AND}} = 1$: if agent 1 is the seed, all agents are expected to be active at steady state of the LTM under Protocol OR and only agent 1 would be active under Protocol AND.

Table 8.1: Comparison of Different Networks

Network	P_1	P_2	P_3	P_4	P_5	Cascade Centrality of (1)
Duplex (OR)	1	1	1	$\frac{7}{8}$	$\frac{7}{8}$	4.75
Duplex (AND)	1	0	0	0	0	1
Layer 1	1	$\frac{1}{2}$	1	$\frac{1}{4}$	$\frac{1}{4}$	3
Layer 2	1	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{2}$	$\frac{1}{4}$	2
Projection	1	$\frac{16}{27}$	$\frac{5}{9}$	$\frac{5}{9}$	$\frac{13}{27}$	3.11

8.6.4 Ordering of probabilities

Let $\mathcal{G}$ be a duplex network with graphs G_1 and G_2 as its two layers. Given an initial active set S_0, we consider the probability of an unseeded agent v being active at steady state under Protocol OR (P_v^{OR}) and under Protocol AND (P_v^{AND}). The probabilities of v being active at steady state in monoplex networks G_1 and G_2 separately are denoted by P_v^1 and P_v^2, respectively.

Corollary 3. *Under the above setting, we have*

$$P_v^{AND} \leq P_v^1 \leq P_v^{OR}$$

$$P_v^{AND} \leq P_v^2 \leq P_v^{OR}$$

Proof. Under a selection of live edges in the duplex network, if v is reachable under Reachability AND, all branches are finite and end with the seeds. In particular, this holds true for the leftmost branch, which only consists of edges in G_1. In G_1, this selection of edges forms a live-edge path. Thus, v is reachable in G_1 as a monoplex network. Considering all possible selections of live edges, we conclude that whenever v is reachable under Reachability AND, v is reachable in G_1 as a monoplex network. Using the equivalence of the LTM and LEM, $P_v^{\text{AND}} \leq P_v^1$.

Under a selection of live edges, if v is reachable in G_1 as a monoplex network, then a live-edge path in G_1 is formed. Considering the live-edge tree of v for the duplex network, the leftmost branch only consists of edges in G_1 and it is exactly the live-edge path. Thus, this branch is finite and ends with the seeds and v is reachable under Reachability OR. Considering all possible selections of live edges, we conclude that whenever v is reachable in G_1 as a monoplex network, v is reachable under Reachability OR. Using the equivalence of the LTM and LEM, $P_v^1 \leq P_v^{\text{OR}}$.

The inequality for layer 2 is proved similarly. □

8.6.5 Example

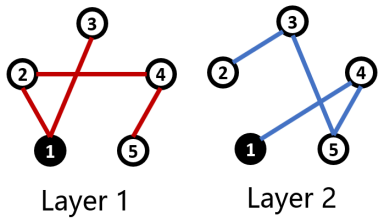

Figure 8.3: Example duplex network

Figure 8.3 shows a duplex network with undirected graphs. We calculate cascade centrality of ① in five cases: duplex cascade centrality under protocol OR, duplex cascade centrality under protocol AND, layer 1 as a monoplex network, layer 2 as a monoplex network and the projection network as a monoplex network. The results are shown in Table 8.1, where the middle columns are the probabilities of agents becoming active. We can see that for each agent, the probabilities follow the results of Corollary 3.

8.7 Final Remarks

We have generalized the linear threshold model with randomly selected thresholds to study diffusion of innovations in multiplex networks, deriving tools to compute social influence and cascade centralities in duplex (two-layer) networks. The LTM for duplex networks is more complicated than it is for monoplex networks as in [45, 55]. Similar to the monoplex case, the live-edge model is leveraged in the duplex case. However, the latter is inherently more complicated because a decision in the duplex case depends on selections in both layers, and we cannot simply analyze live-edge paths in each layer independently. In monoplex networks, the set of selections of live edges increases exponentially with number of agents, but formation of a live-edge path does not depend on selections of agents not in the path and different live-edge paths form independently. Thus, a closed form expression for cascade centrality can be provided for a monoplex network. In duplex networks, these properties do not hold.

We consider directed weighted networks, which is more general than previous research. Our approach does not require assumptions such as the connectedness of the networks. If we add some assumptions on the network, we may be able to give specialized algorithms. For instance, if the graphs in both layers are undirected and connected, we can prove that the finite branches in live-edge trees always end with seeds. Since infinite branches are caused by cycles in the projection network, checking for infinite branches can be implemented by checking for cycles in the selection of live edges and checking for finite branches can be implemented by checking for connectivity in the selection of live edges.

For multiplex networks with more than two layers, we can generalize our protocols by introducing another interlayer threshold parameter μ_{inter}. Then the protocol is stated as if the portion of positive inputs from all layers of agent v exceeds μ_{inter}, then v will become active. In duplex networks, $\mu_{\text{inter}} \in [0, 0.5)$ corresponds to Protocol OR, and $\mu_{\text{inter}} \in [0.5, 1)$ corresponds to Protocol AND.

Chapter 9

Influence Spread in the Heterogeneous Multiplex Linear Threshold Model

Yaofeng Desmond Zhong, Vaibhav Srivastava, Naomi Ehrich Leonard

In preparation and appears as Zhong et al. [103]

The linear threshold model (LTM) has been used to study spread on single-layer networks defined by one inter-agent sensing modality and agents homogeneous in protocol. We define and analyze the heterogeneous multiplex LTM to study spread on multi-layer networks with each layer representing a different sensing modality and agents heterogeneous in protocol. Protocols are designed to distinguish signals from different layers: an agent becomes active if a sufficient number of its neighbors in each of any a of the m layers is active. We focus on Protocol OR, when $a = 1$, and Protocol AND, when $a = m$, which model agents that are most and least readily activated, respectively. We develop theory and algorithms to compute the size of the spread at steady state for any set of initially active agents and to analyze the role of distinguished sensing modalities, network structure, and heterogeneity. We show how heterogeneity manages the tension in spreading dynamics between sensitivity to inputs and robustness to disturbances.

9.1 Introduction

The spread of an activity or innovation across a population of agents that sense or communicate over a network has critical consequences for a wide range of systems from biology to engineering. The adoption of a strategy, such as wearing a face mask during a pandemic, can spread across a social network even when there are only a few early adopters. The observation and response to a threat by one or more vigilant animals can spread through a social animal group. A robot that detects a change in the environment and takes action can spread its behavior across a networked robot team.

To predict and control spread, we present and analyze a new model that captures the realities of *multiple inter-agent sensing modalities* and *heterogeneity in responsiveness of agents to others*. We develop and prove the validity of new algorithms that provide the means to systematically determine the spreading influence of a set of agents as a function of multi-layer network structure and agent heterogeneity.

The linear threshold model (LTM), from Granovetter [31] and Schelling [80], describes the spread of an activity as discrete-time, discrete-valued state dynamics where an agent adopts or rejects an activity by comparing the fraction of its neighbors that have adopted the activity to its individual threshold. Kempe et al. [45] used the LTM with random thresholds to investigate spread of an activity over a population on a single-layer network. Lim et al. [55] introduced and analyzed the notion of cascade and contagion centralities in the model of [45]. The LTM on single-layer networks has also been studied in [1, 70, 25, 24, 72] and generalized to continuous-time, real-valued dynamics in [98].

The single-layer network in the LTM represents a single sensing modality or a projection of multiple sensing modalities. Yet, in real-world systems, agents may distinguish the different sensing modalities, rather than project them, in ways that impact spread. For example, someone deciding whether or not to wear a mask may consider as separate signals what they see others doing in the neighborhood and what they hear over social media that others are doing. And, how they act on the signals may differ from person to person. A more readily activated person starts wearing a mask when they observe enough of the first *or* second group wearing a mask. A less readily activated person starts wearing a mask only when they observe enough of the first *and* second groups wearing a mask.

In this paper we leverage multiplex (multi-layer) networks to model spread in a population of

heterogeneous agents that interact through, and distinguish, multiple sensing modalities. Multiplex networks have been used to study consensus dynamics [28, 87, 81, 3, 88]. Yağan and Gligor [94] studied a multiplex LTM using a weighted average of activity across layers. Other models of spread in the case of multiple sensing modalities are reviewed in [75], but most restrict to homogeneous agents.

In [100], we first introduced the LTM on multiplex networks with homogeneous agents, where the graph for each layer is associated with a different sensing modality, and Protocols OR and AND distinguish signals from different layers to model more and less readily activated agents, respectively. We analyzed the duplex (two-layer) LTM with agents that are homogeneous in protocol and showed how to compute cascade centrality, an agent's influence on the steady-state size of the cascade. Yang et al. [96] studied the influence minimization problem for the homogeneous model of [100].

Our contributions in the present paper are multifold. First, we define the heterogeneous multiplex LTM to analyze spreading dynamics on an arbitrary number of network layers with agents that employ protocols heterogeneously. Second, we define the heterogeneous multiplex live-edge model (LEM), which generalizes [45], and we introduce the live-edge tree to define reachability on this LEM. We prove a key result on equivalence of probabilities for the LTM and LEM.

Third, we derive Algorithms 1 and 5 to compute influence spread for the heterogeneous multiplex LTM. Algorithm 1 is provably correct and useful for small networks. Algorithm 5 maps the influence spread calculation to an inference problem in a Bayesian network and is efficient for large networks. We prove that calculating influence spread is #P-complete.

Fourth, we derive analytical expressions for influence spread in classes of multiplex networks. We show *how* ORs enhance and ANDs diminish spreading relative to the projected network. Fifth, we investigate heterogeneity in spreading and show how it can be used to manage the tradeoff between sensitivity to a real input and robustness to a spurious signal.

Section II describes multiplex networks. Sections III and IV introduce the heterogeneous multiplex LTM and LEM, respectively. We prove their equivalence in Section V. Sections VI and VII present Algorithms 1 and 2. Section VIII presents analytical expressions for influence spread. Heterogeneity is studied in Section IX. We conclude in Section X.

9.2 Multiplex Networks

A *multiplex network* $\mathcal{G}$ is a family of $m \in \mathbb{N}$ directed weighted graphs $G_1, ..., G_m$. Each graph $G_k = (V, E^k)$, $k = 1, \ldots, m$, is a *layer* of the multiplex network. The agent set $V = \{1, 2, 3, ..., n\}$ is the same in all layers. The edge set of layer k is $E^k \subseteq V \times V$ and can be different in different layers. Each edge $e^k_{i,j} \in E^k$, pointing from i to j in layer k, is assigned a weight $w^k_{i,j} \in \mathbb{R}^+$. Here we adopt the "sensing" convention for edges: edge $e^k_{i,j}$ exists if agent i can sense agent j in layer k. If edge $e^k_{i,j}$ exists, agent j is an *out-neighbor* of agent i in layer k. We denote the set of out-neighbors of i in layer k as N^k_i. We say that the weight of agent i's out-neighbor j in layer k is the weight $w^k_{i,j}$. We assume the weights of all out-neighbors for every agent sum up to 1, i.e., $\sum_{j \in N^k_i} w^k_{i,j} = 1$ for every agent i. A *monoplex network* is a multiplex network with $m = 1$, i.e., with only a single layer.

For undirected graphs, every edge is modeled with two opposing directed edges. For unweighted graphs, every edge $e^k_{i,j}$ can be assigned a weight $w^k_{i,j} = 1/d^k_i$, where d^k_i is the *out-degree* of node i in layer k and equal to the number of out-neighbors of node i in layer k. A *projection network* of $\mathcal{G}$ is the graph $\text{proj}(\mathcal{G}) = (V, E)$ where $E = \cup^m_{k=1} E^k$.

9.3 The Heterogeneous Multiplex LTM

The linear threshold model (LTM) is described by a discrete-time dynamical system in which the state $x_i(t) \in \{0, 1\}$ of each agent i at iteration t is inactive with $x_i(t) = 0$ or active with $x_i(t) = 1$. The LTM protocol determines how the active state spreads through the network. Our focus is on which agents will be active at steady state as a function of which agents are active initially. We define $\bar{x}_i = \lim_{t \to \infty} x_i(t)$.

In Section 9.3.1, we recall the LTM for monoplex networks. In Section 9.3.2, we generalize the LTM to multiplex networks by defining protocols for how the active state spreads when signals from different layers are distinguished. Our definition allows for heterogeneity among agents in protocol.

Let S_t be the set of agents that are active by the end of iteration t. Once active, an agent remains active so that $S_{t-1} \subseteq S_t$. At $t = 0$, all agents are inactive except the initially active set S_0. Every agent in S_0 is called a *seed*. The LTM protocol determines when inactive agents at iteration $t - 1$ become active at iteration t. A steady state is reached when $S_{t-1} = S_t$.

9.3.1 Monoplex LTM

The LTM protocol on a monoplex network is defined as follows (e.g., [45]). Each agent $i = 1, \ldots, n$ chooses a threshold μ_i randomly and independently from a uniform distribution $U(0, 1)$. An inactive agent i at iteration $t-1$ becomes active at iteration t if the sum of weights of its active out-neighbors at $t-1$ exceeds μ_i, that is, if $\mu_i < \sum_{j \in N_i \cap S_{t-1}} w_{i,j}$. For n agents, steady state is reached by $t \leq n$.

9.3.2 Multiplex LTM

We introduce the LTM on a multiplex network with m layers by defining a family of protocols as follows. Each agent i chooses a threshold μ_i^k in each layer k for $i = 1, \ldots, n$ and $k = 1, \ldots, m$. Each μ_i^k is randomly and independently drawn from the uniform distribution $U(0, 1)$. In general, each agent has different neighbors in different layers. If the sum of weights of active out-neighbors of agent i in layer k at $t-1$ exceeds μ_i^k, that is, $\mu_i^k < \sum_{j \in N_i^k \cap S_{t-1}} w_{i,j}^k$, we say agent i receives a positive input $y_i^k(t) = 1$ from layer k at t. Otherwise, agent i receives a neutral input $y_i^k(t) = 0$.

The protocols that determine whether or not an inactive agent at $t-1$ becomes active at t account for the possibility that the inputs it receives at t from the different layers may be conflicting. Let the average input agent i receives at t be $y_i(t) = \sum_{k=1}^m y_i^k(t)/m$.

Definition 12 (Multiplex LTM Protocol)**.** *Given multiplex network $\mathcal{G}$ with seed set S_0, the* multiplex LTM protocol for agent i *is parametrized by $\delta_i \in [1/m, 1]$ as follows:*

$$x_i(0) = 1, \quad \forall i \in S_0 \tag{9.1}$$

$$x_i(0) = 0, \quad \forall i \notin S_0 \tag{9.2}$$

$$x_i(t) = \begin{cases} 1, & \text{if } y_i(t) \geq \delta_i \text{ or } x_i(t-1) = 1 \\ 0, & \text{otherwise.} \end{cases} \tag{9.3}$$

We identify two protocols for the limiting values of δ_i:

Protocol OR*: $\delta_i = 1/m$. Inactive agent i at iteration $t-1$ becomes active at iteration t if it receives a positive input from* any *layer at t;*

Protocol AND*: $\delta_i = 1$. Inactive agent i at iteration $t-1$ becomes active at iteration t if it receives positive inputs from* all *layers at t.* □

The multiplex LTM protocol specifies that inactive agent i at iteration $t-1$ becomes active at iteration t if it receives a positive input from any $a_i \in \{1, \ldots, m\}$ of the m layers, where $(a_i - 1)/m <$

$\delta_i \leq a_i/m$. Asymmetric sensitivity to layers can be modelled with y_i a convex combination of y_i^k.

In this paper, we examine the two limiting cases: Protocol OR, where $a_i = 1$, and Protocol AND, where $a_i = m$. Analysis in these cases is sufficient for understanding heterogeneity and spreading dynamics on multi-layer networks. Our theory can be extended to protocols for $a_i \in \{2, \ldots, m-1\}$.

Remark 1. *Protocol OR models agents that are readily activated: there only needs to be sufficient activity among neighbors in one layer at $t-1$ in order for agents to become active at t. Protocol AND models agents that are conservatively activated: there needs to be sufficient activity among neighbors in every layer at $t-1$ in order for agents to become active at t. Thus, agents with Protocol OR enhance spreading and agents with Protocol AND diminish spreading.*

We study heterogeneous networks in which some agents use Protocol OR while the others use Protocol AND.

Definition 13 (Sequence of Protocols)**.** *Let $u_i \in \{OR, AND\}$ be the protocol used by agent i. We define the* sequence of protocols $\mathcal{U} = (u_1, u_2, ..., u_n)$ *to be the protocols used by the n agents ordered from agent 1 to agent n.*

Lemma 6. *For a multiplex network $\mathcal{G}$ with n agents, the multiplex LTM converges in at most n iterations.*

Proof. Assume the multiplex LTM converges in more than n iterations. Then at least one agent switches from inactive to active in each of the first n iterations and these agents are distinct. There is at least one agent in the initial active set that is not one of those n agents. This implies at least $n+1$ agents in the network, which is a contradiction. □

9.4 The Heterogeneous Multiplex LEM

Our approach to analyzing the multiplex LTM generalizes the approach in [45], which uses the live-edge model (LEM) for monoplex networks to analyze the monoplex LTM. In this section we define the multiplex LEM. In Section 9.4.1, we recall the LEM proposed in [45] for monoplex networks. In Section 9.4.2, we generalize the LEM to multiplex networks and introduce the notion of reachability-$\mathcal{U}$ on the multiplex LEM. Unlike in our earlier work [100], the reachability we propose here allows for heterogeneous protocols among agents.

9.4.1 Monoplex LEM and Reachability

The LEM for a monoplex network is defined as follows [45]. Let S_0 be the set of seeds. Each unseeded agent randomly selects one of its outgoing edges with probability given by the edge

weight. The selected edge is labeled as "live", while the unselected edges are labeled as "blocked". The seeds block all of their outgoing edges. Every directed edge will thus be either live or blocked. The choice of edges that are live is called a *selection of live edges*.

Let L be the set of all possible selections of live edges. The probability q_l of selection $l \in L$ is the product of the weights of the live edges in selection l. Because the selection of live edges can be done at the same time for every node, the LEM can be viewed as a static model. The LEM can alternatively be viewed as an iterative process in the case the live edges are selected sequentially.

A *live-edge path* [45] is a directed path that consists only of live edges. Let $\mathcal{L}_{ij}$ be the set of all possible distinct live-edge paths from agent $i \notin S_0$ to $j \in S_0$. The probability r_α of live-edge path $\alpha \in \mathcal{L}_{ij}$ is the product of the edge weights along the path. We say $i \notin S_0$ is *reachable from* $j \in S_0$ *by live-edge path* α *with probability* r_α, and $i \notin S_0$ is *reachable from* $j \in S_0$ *with probability* r_{ij}, where $r_{ij} = \sum_{\alpha \in \mathcal{L}_{ij}} r_\alpha$.

Alternatively, we can compute r_{ij} in terms of selections of live-edges. Let $L_{ij} \subseteq L$ be the set of all selections of live edges that contain a live-edge path from $i \notin S_0$ to $j \in S_0$. Then, $r_{ij} = \sum_{l \in L_{ij}} q_l$. Likewise, let $L_{iS_0} \subseteq L$ be the set of all selections of live edges that contain a live-edge path from $i \notin S_0$ to at least one node $j \in S_0$. Then, $i \notin S_0$ is *reachable from* S_0 *with probability* r_{iS_0}, where $r_{iS_0} = \sum_{l \in L_{iS_0}} q_l$.

9.4.2 Multiplex LEM and Reachability

We introduce the LEM on a multiplex network as follows.

Definition 14 (Multiplex LEM)**.** *Consider a multiplex network $\mathcal{G}$ with seed set S_0. In each layer k, each unseeded agent i randomly selects one of its outgoing edges e^k_{i,j_k} with probability w^k_{i,j_k}. The selected edges are labeled as "live", while the unselected edges are labeled as "blocked". The seeds block all of their outgoing edges in every layer. The choice of edges that are live is a* multiplex selection of live edges*. Let L be the set of all possible multiplex selections of live edges. The probability q_l of selection $l \in L$ is the product of the weights of all live edges in selection l.*

The challenge in generalizing the LEM to multiplex networks is in properly defining reachability. Here we introduce the live-edge tree, which we use to define reachability.

Definition 15 (Live-edge Tree)**.** [1] *Given a set of seeds S_0 and a multiplex selection of live edges $l \in L$, the* live-edge tree *T^l_i associated with agent $i \notin S_0$ is constructed as follows with agent i as the root node. Let e^k_{i,j_k} be the live edge of agent i in layer k, $k = 1, \ldots, m$. Then the children of the root node are agents*

[1] To highlight key differences between multiplex and monoplex networks, we assume each $i \notin S_0$ has at least one neighbor in each layer. If not, with a slight modification of Defs. 15- 16, the theory and computation are still valid.

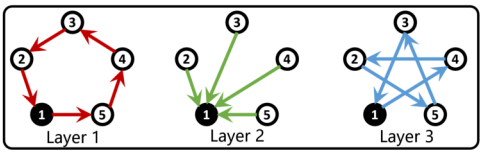

Figure 9.1: An example of a three-layer multiplex network with five agents. Agent 1 is the seed, which is denoted by the black circle.

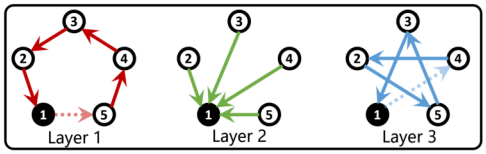

Figure 9.2: The unique multiplex selection of live edges for the network in Fig. 9.1.

$j_1, j_2, \ldots, j_m$, and the root node is connected to each child with the live edge in the corresponding layer. The tree is constructed recursively in this way for each child that itself has at least one child. Any agent in the network may appear multiple times as a node in the tree.

Fig. 9.1 provides an example of a three-layer multiplex network with five agents. The network has only one possible multiplex selection of live edges, given in Fig. 9.2. Fig. 9.3 shows the corresponding live-edge tree associated with agent 5.

We next define reachability from S_0 of an unseeded agent in a multiplex network under a sequence of protocols $\mathcal{U}$.

Definition 16 ($\mathcal{U}$-Reachability). *Consider multiplex network $\mathcal{G}$ with seed set S_0 and multiplex selection of live edges $l \in L$. Let T_i^l be the live-edge tree associated with agent $i \notin S_0$. Suppose there are b distinct branches in T_i^l indexed by $\beta = 1, \ldots, b$ and of the form: $B_\beta = (i, e_{i,i_1}^{k_0}, i_1, e_{i_1,i_2}^{k_1}, i_2, \ldots, i_s)$, where $i_j \in V$, $j = 1, \ldots, s$, $i_s \in S_0$, and each agent in V appears at most once in B_β. We call each B_β a* distinct branch that ends in a seed. *Denote the set of these branches as $\mathcal{B}_i^l = \{B_1, B_2, \ldots, B_b\}$. For any subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$, let*

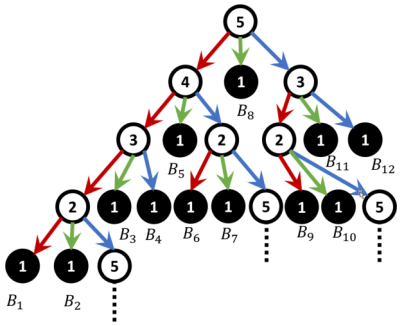

Figure 9.3: The live-edge tree associated with agent 5 for the example three-layer multiplex network of Fig. 9.1 and the unique selection of live edges of Fig. 9.2. $\mathcal{B}_5 = \{B_1, B_2, \ldots, B_{12}\}$ is the set of distinct branches that end with a seed.

the set of agents in $\hat{\mathcal{B}}$ be $\hat{V}$ and the set of edges in $\hat{\mathcal{B}}$ be $\hat{E}$.

Given a sequence of protocols $\boldsymbol{\mathcal{U}}$, we say that branch subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$ is $\boldsymbol{\mathcal{U}}$-feasible for *i if $\hat{\mathcal{B}} \neq \emptyset$ and for every $\hat{i} \in \hat{V} \setminus S_0$ for which $u_{\hat{i}} =$* AND*, all of $\hat{i}$'s live edges belong to $\hat{E}$. Then, i is* $\boldsymbol{\mathcal{U}}$-reachable from S_0 by the selection of live edges l with probability q_l *if there exists at least one $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$ that is $\boldsymbol{\mathcal{U}}$-feasible for i. Let $L_{iS_0}^{\boldsymbol{\mathcal{U}}} \subseteq L$ be the set of all selections of live edges by which i is $\boldsymbol{\mathcal{U}}$-reachable from S_0. Then, i is* $\boldsymbol{\mathcal{U}}$-reachable from S_0 with probability $r_{iS_0}^{\boldsymbol{\mathcal{U}}}$, *where $r_{iS_0}^{\boldsymbol{\mathcal{U}}} = \sum_{l \in L_{iS_0}^{\boldsymbol{\mathcal{U}}}} q_l$.* □

Remark 2. *The condition for $\boldsymbol{\mathcal{U}}$-feasibility for i of a branch subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_i^l$ does not make explicit a condition on any $\hat{i} \in \hat{V} \setminus S_0$ for which $u_{\hat{i}} =$ OR. This follows since for any such agent $\hat{i}$, the condition is that at least one of its live edges must be in $\hat{E}$ and this is always true by definition.*

To illustrate $\boldsymbol{\mathcal{U}}$-reachability, consider the live-edge tree associated with agent 5 in Fig. 9.3 for the unique selection of live edges in Fig. 9.2 for the multiplex network of Fig. 9.1 with seed set $S_0 = \{1\}$. Because the selection of live edges in Fig. 9.2 is unique, it is chosen with probability $q = 1$. Therefore, agent $i \notin S_0$ is $\boldsymbol{\mathcal{U}}$-reachable from S_0 with probability 1 if there exists at least one $\hat{\mathcal{B}} \subseteq \mathcal{B}_i$ that is $\boldsymbol{\mathcal{U}}$-feasible for i. For agent 5, there are 12 distinct branches that end in a seed, as shown in Fig. 9.3; thus, $\mathcal{B}_5 = \{B_1, B_2, ..., B_{12}\}$. For example, $B_8 = (5, e_{5,1}^2, 1)$.

We compute $\boldsymbol{\mathcal{U}}$-reachability from S_0 for agent 5 for each the following three sequences of protocols used by the five agents in the three-layer multiplex network:

$$\boldsymbol{\mathcal{U}}_1 = (\text{OR}, \text{AND}, \text{AND}, \text{AND}, \text{OR}) \tag{9.4}$$

$$\boldsymbol{\mathcal{U}}_2 = (\text{OR}, \text{OR}, \text{AND}, \text{AND}, \text{AND}) \tag{9.5}$$

$$\boldsymbol{\mathcal{U}}_3 = (\text{OR}, \text{AND}, \text{AND}, \text{AND}, \text{AND}). \tag{9.6}$$

1. Let $\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{U}}_1$. Consider $\hat{\mathcal{B}} = \{B_8\} \subset \mathcal{B}_5$. Then, $\hat{V} = \{1, 5\}$ and $\hat{E} = \{e_{5,1}^2\}$. Since 5 is the only unseeded node in $\hat{V}$ and $u_5 =$ OR, $\hat{\mathcal{B}}$ is $\boldsymbol{\mathcal{U}}$-feasible for 5. Thus, agent 5 is $\boldsymbol{\mathcal{U}}$-reachable from S_0 with probability 1.

2. Let $\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{U}}_2$. Consider $\hat{\mathcal{B}} = \mathcal{B}_5$. Then, $\hat{V} = \{1, 2, 3, 4, 5\}$. The unseeded nodes $j \in \hat{V}$ for which $u_j =$ AND are $j = 3, 4, 5$. From Fig. 9.3, observe that all the live edges of nodes 3, 4, and 5, belong to $\hat{E}$, the edge set of $\hat{\mathcal{B}} = \mathcal{B}_5$. Thus, $\hat{\mathcal{B}}$ is $\boldsymbol{\mathcal{U}}$-feasible for 5, and agent 5 is $\boldsymbol{\mathcal{U}}$-reachable from S_0 with probability 1.

3. Let $\boldsymbol{\mathcal{U}} = \boldsymbol{\mathcal{U}}_3$. In this case there is no $\boldsymbol{\mathcal{U}}$-feasible subset $\hat{\mathcal{B}} \subseteq \mathcal{B}_5$, since $u_2 =$ AND and agent 2 has a live edge $e_{2,5}^3$, which is not in the edge set of any branch in $\mathcal{B}_5$. Thus, agent 5 is not $\boldsymbol{\mathcal{U}}$-reachable from S_0.

In the next section we prove the equivalence of the probability that agent i is $\mathcal{U}$-reachable from S_0 for the multiplex LEM and the probability that agent i is active at steady state for the multiplex LTM with seed set S_0. For our example, this implies that under $\mathcal{U}_1$ or $\mathcal{U}_2$, agent 5 will become active at steady state with probability 1 and under $\mathcal{U}_3$, agent 5 will remain inactive at steady state.

9.5 Equivalence of LTM and LEM

The monoplex LEM was introduced in [45] and proved to be equivalent to the monoplex LTM in the sense that the probabilities of agents being reachable from a set S_0 in the LEM are equal to the probabilities of agents being active at steady state given seed set S_0 in the LTM. Computing these probabilities for the LTM is challenging because it requires solving over temporal iterations. However, leveraging the equivalence, the probability distributions can be computed without temporal iteration using the LEM as a static model.

The equivalence is recalled in Section 9.5.1 for monoplex networks and proved in Section 9.5.2 for multiplex networks.

9.5.1 Equivalence for Monoplex Networks

The LTM and LEM were proved to be equivalent in [45] in the following sense. For a given monoplex network G with seed set S_0, the probabilities of the following two events for arbitrary agent $i \notin S_0$ are the same:

1. i is active at steady state for the LTM with random thresholds and initial active set S_0;
2. i is reachable from set S_0 under the random selection of live edges in the LEM.

9.5.2 Equivalence for Multiplex Networks

We generalize the equivalence of LTM and LEM to multiplex networks in this section. First, we prove the following lemma that infers an agent's $\mathcal{U}$-reachability from the $\mathcal{U}$-reachability of its children in the live-edge tree. We then leverage this lemma to prove the equivalence in Theorem 6.

Lemma 7. *Given a multiplex network $\mathcal{G}$ with seed set S_0, multiplex selection of live edges $l \in L$ and sequence of protocols $\mathcal{U}$, consider agent $i \notin S_0$ and its associated live-edge tree T_i^l. Assume i's live edge in layer k connects to agent i_1^k, $k = 1, \ldots, m$. Then the $\mathcal{U}$-reachability of i from S_0 by selection l can be inferred from the reachability of its children i_1^k and its protocol u_i as follows:*

1. *Let u_i = OR. Then, i is $\mathcal{U}$-reachable from S_0 by selection of live edges l if and only if at least one child i_1^k is $\mathcal{U}$-reachable from S_0 by selection of live edges l.*

2. *Let u_i = AND. Then, i is $\mathcal{U}$-reachable from S_0 by selection of live edges l if and only if every child i_1^k is $\mathcal{U}$-reachable from S_0 by selection of live edges l.*

Proof. Let $T^l_{i_1^k}$ be the corresponding live-edge tree associated with agent i's child i_1^k for $k = 1, \ldots, m$.

1) Let u_i = OR and suppose i_1^k is $\mathcal{U}$-reachable from S_0 by selection l for some k. By Definition 16, the set $\mathcal{B}^l_{i_1^k}$ of distinct branches that end with a seed in $T^l_{i_1^k}$ is nonempty and there exists a subset $\hat{\mathcal{B}}_k \subseteq \mathcal{B}^l_{i_1^k}$ that is $\mathcal{U}$-feasible for i_1^k such that for every $\hat{i} \in \hat{V}_k \setminus S_0$ for which $u_{\hat{i}}$ = AND, all of $\hat{i}$'s live edges belong to $\hat{E}_k$, where $\hat{V}_k$ and $\hat{E}_k$ are the sets of agents and edges in $\hat{\mathcal{B}}_k$, respectively.

For every branch $B_r^k = (i_1^k, e^{k'}_{i_1^k,i_2}, i_2, \ldots, i_s) \in \hat{\mathcal{B}}_k$, there exists a branch $B_r^{k0} = (i, e^k_{i,i_1^k}, i_1^k, e^{k'}_{i_1^k,i_2}, i_2, \ldots, i_s)$ in T_i^l. Let $\hat{\mathcal{B}}_{k0} \subseteq \mathcal{B}_i^l$ be the set of all branches in T_i^l that correspond to branches in $\hat{\mathcal{B}}_k$. Then, $\hat{V}_{k0} = \hat{V}_k \cup \{i\}$ is the set of agents in $\hat{\mathcal{B}}_{k0}$ and $\hat{E}_{k0} = \hat{E}_k \cup \{e^k_{i,i_1^k}\}$ is the set of edges in $\hat{\mathcal{B}}_{k0}$. Thus, since u_i = OR and $\hat{\mathcal{B}}_k$ is $\mathcal{U}$-feasible for i_1^k, by Definition 16, $\hat{\mathcal{B}}_{k0}$ must be $\mathcal{U}$-feasible for i. Agent i is therefore $\mathcal{U}$-reachable from S_0 by selection l. This proves the "if" part of the statement.

If no child i_1^k is $\mathcal{U}$-reachable from S_0 by selection l, then there exists no nonempty $\hat{\mathcal{B}}_k$ in $T^l_{i_k}$ that is $\mathcal{U}$-feasible for i_1^k. This implies there is no nonempty $\hat{\mathcal{B}}_{k0}$ in T_i^l that is $\mathcal{U}$-feasible for i. Therefore, agent i cannot be $\mathcal{U}$-reachable from S_0 by selection l. This proves the "only if" part of the statement.

2) Let u_i = AND and suppose i_1^k is $\mathcal{U}$-reachable from S_0 by selection l, for every $k = 1, \ldots, m$. By Definition 16, for every k, there exists a subset $\hat{\mathcal{B}}_k \subseteq \mathcal{B}^l_{i_1^k}$ in $T^l_{i_1^k}$ that is $\mathcal{U}$-feasible for i_1^k. For every k let $\hat{\mathcal{B}}_{k0} \subseteq \mathcal{B}_i^l$ be the set of all branches in T_i^l that correspond to branches in $\hat{\mathcal{B}}_k$ as defined in the proof of 1). Let $\hat{\mathcal{B}}_0 = \cup_{k=1}^m \hat{\mathcal{B}}_{k0} \subseteq \mathcal{B}_i^l$ with agent set $\hat{V}_0$ and edge set $\hat{E}_0$. By construction, $i \in \hat{V}_0$ and all of agent i's live edges belong to $\hat{E}_0$. It follows that $\hat{\mathcal{B}}_0$ is $\mathcal{U}$-feasible for i and thus agent i is $\mathcal{U}$-reachable from S_0 by selection l. This proves the "if" part of the statement.

If there is one child i_1^k that is not $\mathcal{U}$-reachable from S_0 by selection l, then there exists no set $\hat{\mathcal{B}}_k$ in $T^l_{i_k}$ that is $\mathcal{U}$-feasible for i_1^k. Suppose there is a set $\hat{\mathcal{B}}_0 \subseteq \mathcal{B}_i^l$ in T_i^l that is $\mathcal{U}$-feasible for i. Since u_i = AND, by Definition 16, it follows that edge $e^k_{i,i_1^k} \in \hat{E}_0$. For every branch in $\hat{\mathcal{B}}_0$ that starts with i and edge e^k_{i,i_1^k}, i.e., $B_r^{k0} = (i, e^k_{i,i_1^k}, i_1^k, e^{k'}_{i_1^k,i_2} i_2, \ldots, i_s)$, we denote the set of all corresponding branches $B_r^k = (i_1^k, e^{k'}_{i_1,i_2}, i_2, \ldots, i_s)$ as $\hat{\mathcal{B}}_k \subseteq \mathcal{B}^l_{i_1^k}$. It follows that $\hat{\mathcal{B}}_k$ is $\mathcal{U}$-feasible for i_1^k in $T^l_{i_1^k}$ and i_1^k is reachable from S_0. This is a contradiction. Thus, there is no set $\hat{\mathcal{B}}_0$ that is $\mathcal{U}$-feasible for i and agent i cannot be $\mathcal{U}$-reachable from S_0 by selection l. This proves the "only if" part of the statement. □

While Definition 16 uses the LEM to define $\mathcal{U}$-reachability in a static way, the LEM can also be used to reveal the $\mathcal{U}$-reachability of agents as an iterative process over time [45] as follows. First,

using Lemma 7, determine the $\mathcal{U}$-reachability of the agents with at least one edge coming from initial set $S'_0 = S_0$. If an agent is determined to be $\mathcal{U}$-reachable from S'_0, add it to S'_0 to get a new reachable set S'_1. In the next iteration, follow the same procedure and get a sequence of reachable sets $S'_0, S'_1, S'_2, \ldots$. The process ends at iteration t if $S'_t = S'_{t-1}$, where S'_t is the set of agents that are $\mathcal{U}$-reachable from S_0. The mapping between static and temporal determinations of reachability for the LEM is the key to proving the equivalence of the multiplex LTM and multiplex LEM. We also make use of the following definitions.

Definition 17 (LTM-related events)**.** *We define $f_i^k(t)$ ($g_i^k(t)$) to be the event that the sum of weights of active out-neighbors of i in layer k does (does not) exceed μ_i^k at t: $f_i^k(t) = \{\mu_i^k < \sum_{j \in N_i^k \cap S_t} w_{i,j}^k\}$ and $g_i^k(t) = \{\mu_i^k \geq \sum_{j \in N_i^k \cap S_t} w_{i,j}^k\}$. Let*

$$X_k = f_i^k(t), \quad Y_k = g_i^k(t-1), \quad Z_k = f_i^k(t-1).$$

Definition 18 (LTM-related probabilities)**.** *For agent i that is inactive at t, we define the probability that i becomes active at $t+1$ as $\mathbb{P}^{t+1}_{i(\mathrm{OR})}$ if i uses Protocol OR, and as $\mathbb{P}^{t+1}_{i(\mathrm{AND})}$ if i uses Protocol AND.*

Definition 19 (LEM-related events)**.** *We define $f_i'^k(t)$ ($g_i'^k(t)$) to be the event that agent i's live edge in layer k does (does not) connect to the reachable set S'_t at t. Let*

$$X'_k = f_i'^k(t), \quad Y'_k = g_i'^k(t-1), \quad Z'_k = f_i'^k(t-1).$$

Definition 20 (LEM-related probabilities)**.** *Consider the LEM as an iterative process. If agent $i \notin S'_t$, then we define the probability that $i \in S'_{t+1}$ as $\mathbb{P}'^{t+1}_{i(\mathrm{OR})}$ if i uses Protocol OR, and as $\mathbb{P}'^{t+1}_{i(\mathrm{AND})}$ if i uses Protocol AND.*

We state the equivalence of multiplex LTM and multiplex LEM in the following theorem.

Theorem 6. *For a multiplex network $\mathcal{G}$ with seed set S_0, multiplex selection of live edges $l \in L$ and sequence of protocols $\mathcal{U}$, the probabilities of the following two events regarding an arbitrary agent $i \notin S_0$ are the same:*

1. *i is active at steady state for the multiplex LTM under $\mathcal{U}$ with random thresholds and initial active set S_0;*
2. *i is $\mathcal{U}$-reachable from the set S_0 under random selection of live edges in the multiplex LEM.*

Proof. We prove by mathematical induction.

We use $\mathbb{P}(X_k|Y_k) = \mathbb{P}(X'_k|Y'_k)$, which we have from [45]. We show i) $\mathbb{P}^{t+1}_{i(\text{OR})} = \mathbb{P}'^{t+1}_{i(\text{OR})}$ and ii) $\mathbb{P}^{t+1}_{i(\text{AND})} = \mathbb{P}'^{t+1}_{i(\text{AND})}$, so by induction over the iterations, the probabilities of the two events in the statement of the theorem are the same.

i) *Proving* $\mathbb{P}^{t+1}_{i(\text{OR})} = \mathbb{P}'^{t+1}_{i(\text{OR})}$ *(when i uses Protocol OR)*

In the LTM, i being inactive at t means none of i's thresholds is exceeded at $t-1$ and i being active at $t+1$ means i's thresholds in at least one layer is exceeded at t. In this case, the probability that μ_i^k is exceeded at t is $\mathbb{P}_k = \mathbb{P}(X_k|Y_1, Y_2, ..., Y_m) = \mathbb{P}(X_k|Y_k)$. The last equality holds because random variables $\mu_i^1, \mu_i^2, ..., \mu_i^m$ are independent. Then $\mathbb{P}^{t+1}_{i(\text{OR})}$ is the complement of the probability that its threshold in none of the layers are exceeded, i.e., $\mathbb{P}^{t+1}_{i(\text{OR})} = 1 - \prod_{k=1}^{m}(1 - \mathbb{P}_k)$

In the LEM, by Lemma 7, $i \notin S'_t$ means all of i's live edges are not connected to S'_{t-1} and $i \in S'_{t+1}$ means at least one live edge of i is connected to S'_t. In this case, the probability that i's live edge in layer k connects to S'_t is $\mathbb{P}'_k = \mathbb{P}(X'_k|Y'_1, Y'_2, ..., Y'_k) = \mathbb{P}(X'_k|Y'_k)$. Then $\mathbb{P}'^{t+1}_{i(\text{OR})}$ is the complement of the probability that none of i's live edges connects to S'_t, i.e., $\mathbb{P}'^{t+1}_{i(\text{OR})} = 1 - \prod_{k=1}^{m}(1 - \mathbb{P}'_k)$.

From $\mathbb{P}(X_k|Y_k) = \mathbb{P}(X'_k|Y'_k)$ we have $\mathbb{P}_k = \mathbb{P}'_k, k = 1, 2, ..., m$. So that we conclude that $\mathbb{P}^{t+1}_{i(\text{OR})} = \mathbb{P}'^{t+1}_{i(\text{OR})}$.

ii) *Proving* $\mathbb{P}^{t+1}_{i(\text{AND})} = \mathbb{P}'^{t+1}_{i(\text{AND})}$ *(when i uses Protocol AND)*

In the LTM, i being inactive at t means at least one of its thresholds is not exceeded at $t-1$ and being active at $t+1$ means all of i's thresholds are exceeded at t. In this case, there are $2^m - 1$ possible events, with probabilities denoted as $\mathbb{P}_{m+1}, \ldots, \mathbb{P}_{2^m+m-1}$. We have that $\mathbb{P}_{m+1} = \mathbb{P}(X_1, X_2, ..., X_m|Y_1, Y_2, ..., Y_m)$. The other probabilities have a similar form but with one or more, but not all, of the Y_k replaced by Z_k. Since Y_k and Z_k are mutually exclusive for all k, we have $\mathbb{P}^{t+1}_{i(\text{AND})} = \sum_{l=m+1}^{2^m+m-1} \mathbb{P}_l$.

In the LEM, by Lemma 7, $i \notin S'_t$ means at least one of i's live edges are not connected to S'_{t-1} and $i \in S'_{t+1}$ means all of i's live edges are connected to S'_t. In this case, there are $2^m - 1$ possible events, with probabilities denoted as $\mathbb{P}'_{m+1}, \ldots, \mathbb{P}'_{2^m+m-1}$. We have that $\mathbb{P}'_{m+1} = \mathbb{P}(X'_1, X'_2, ..., X'_m|Y'_1, Y'_2, ..., Y'_m)$. The other probabilities have similar form but with one or more, but not all, of the Y'_k replaced by Z'_k. Since Y'_k and Z'_k are mutual exclusive for all k, we have $\mathbb{P}'^{t+1}_{i(\text{AND})} = \sum_{l=m+1}^{2^m+m-1} \mathbb{P}'_l$.

Since the thresholds are independent of one another, $\mathbb{P}_l$ can be separated into the product of m terms, each of which involves events associated to one layer only: $\mathbb{P}_l = \prod_{k=1}^{m} \mathbb{P}_{0k}$, where $\mathbb{P}_{0k}$ is either $\mathbb{P}(X_k, Y_k)/\mathbb{P}(Y_k) = \mathbb{P}(X_k|Y_k)$ or $\mathbb{P}(X_k, Z_k)/\mathbb{P}(Z_k) = 1$. Since the live edges that each agent uses are independent of one another, similar analysis applies to $\mathbb{P}'_l$: $\mathbb{P}'_l = \prod_{k=1}^{m} \mathbb{P}'_{0k}$, where $\mathbb{P}'_{0k}$ is either $\mathbb{P}(X'_k, Y'_k)/\mathbb{P}(Y'_k) = \mathbb{P}(X'_k|Y'_k)$ or $\mathbb{P}(X'_k, Z'_k)/\mathbb{P}(Z'_k) = 1$. It follows from $\mathbb{P}(X_k|Y_k) = \mathbb{P}(X'_k|Y'_k)$ that

$\mathbb{P}_l = \mathbb{P}'_l, l = m+1, m+2, ..., 2^m + m - 1$ and $\mathbb{P}^{t+1}_{i(\text{AND})} = \mathbb{P}'^{t+1}_{i(\text{AND})}$. □

Theorem 6 generalizes the equivalence of LTM and LEM to multiplex networks. Leveraging Theorem 6, we can calculate an agent's influence, in terms of spreading information through the network, without needing to simulate the multiplex LTM.

9.6 Computing Multiplex Influence Spread

In this section we define multiplex influence spread and multiplex cascade centrality for the LTM. We then derive and prove the validity of an algorithm to compute them.

9.6.1 Monoplex Influence Spread and Cascade Centrality

The *monoplex influence spread of agents in* S_0, denoted $\sigma^G_{S_0}$, is defined as the expected number of active agents at steady state for the monoplex LTM given the network G and initial active set S_0 [45]. The *monoplex cascade centrality of agent* j, denoted C^G_j, is the influence spread of agent j defined in [55] as $C^G_j = \sigma^G_j$, the expected number of active agents at steady state for the monoplex LTM given G and $S_0 = \{j\}$.

9.6.2 Multiplex Influence Spread and Cascade Centrality

Influence spread and cascade centrality are naturally generalized to the multiplex setting as follows.

Definition 21 (Multiplex influence spread)**.** *The* multiplex influence spread of agents in S_0, *denoted* $\sigma^{\mathcal{G},\mathcal{U}}_{S_0}$, *is defined as the expected number of active agents at steady state for the multiplex LTM given the network* $\mathcal{G}$, *sequence of protocols* $\mathcal{U}$, *and initial active set* S_0. *Let* $\mathbb{E}^{\mathcal{G},\mathcal{U}}_{S_0}$ *and* $\mathbb{P}^{\mathcal{G},\mathcal{U}}_{S_0}$ *be expected value and probability, respectively, conditioned on* $\mathcal{G}, \mathcal{U}, S_0$. *Then*

$$\sigma^{\mathcal{G},\mathcal{U}}_{S_0} = \mathbb{E}^{\mathcal{G},\mathcal{U}}_{S_0}\Big(\sum_{i=1}^{n} \bar{x}_i\Big) = \sum_{i=1}^{n} \mathbb{P}^{\mathcal{G},\mathcal{U}}_{S_0}(\bar{x}_i = 1). \tag{9.7}$$

Definition 22 (Multiplex cascade centrality)**.** *The* multiplex cascade centrality of agent j, *denoted* $C^{\mathcal{G},\mathcal{U}}_j$, *is defined as*

$$C^{\mathcal{G},\mathcal{U}}_j = \sigma^{\mathcal{G},\mathcal{U}}_j. \tag{9.8}$$

When $\mathcal{U} = (u, ..., u)$, we replace $\mathcal{U}$ with u in the superscript. For example, when $u = \text{OR}$, we write $\mathbb{P}^{\mathcal{G},\text{OR}}_{S_0}(\bar{x}_i = 1)$ and $C^{\mathcal{G},\text{OR}}_j$. When $\mathcal{G}$ is understood we drop it from the superscript.

9.6.3 Computing Multiplex Influence Spread and Centrality

We can directly compute multiplex influence spread and multiplex cascade centrality by computing probabilities of $\mathcal{U}$-reachability for the LEM, which is much easier than computing probabilities of agents being active at steady state for the LTM. We summarize in a corollary to Theorem 6.

Corollary 4. *Given multiplex network $\mathcal{G}$ and sequence of protocols $\mathcal{U}$, multiplex influence spread of agents in S_0 and multiplex cascade centrality of agent j can be determined as*

$$\sigma_{S_0}^{\mathcal{U}} = \sum_{i=1}^{n} r_{iS_0}^{\mathcal{U}}, \quad C_j^{\mathcal{U}} = \sum_{i=1}^{n} r_{ij}^{\mathcal{U}}. \tag{9.9}$$

Proof. By Definition 16, $r_{iS_0}^{\mathcal{U}}$ is the probability that i is $\mathcal{U}$-reachable from S_0 in the multiplex LEM of Definition 14. By Theorem 6, $r_{iS_0}^{\mathcal{U}} = \mathbb{P}_{S_0}^{\mathcal{U}}(\bar{x}_i = 1)$. Thus, by Definition 21, $\sum_{i=1}^{n} r_{iS_0}^{\mathcal{U}} = \sum_{i=1}^{n} \mathbb{P}_{S_0}^{\mathcal{U}}(\bar{x}_i = 1) = \sigma_{S_0}^{\mathcal{U}}$. In case $S_0 = \{j\}$, by Definition 22, $C_j^{\mathcal{U}} = \sigma_j^{\mathcal{U}} = \sum_{i=1}^{n} r_{ij}^{\mathcal{U}}$. □

Algorithm 1 uses (9.9) to compute multiplex influence spread and can be specialized to compute cascade centrality.

Algorithm 4 (Compute multiplex influence spread $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$)**.** *Given multiplex network $\mathcal{G}$ and sequence of protocols $\mathcal{U}$:*

1. *Find the set L of all possible selections of live edges for multiplex network $\mathcal{G}$ and initially active set S_0. Calculate the probability q_l of each $l \in L$.*
2. *For each agent i find $L_{iS_0}^{\mathcal{U}} \subseteq L$, the set of all $l \in L$ such that i is $\mathcal{U}$-reachable from S_0 by selection l.*
3. *Calculate $r_{iS_0}^{\mathcal{U}} = \sum_{l \in L_{iS_0}^{\mathcal{U}}} q_l$.*
4. *Calculate $\sigma_{S_0}^{\mathcal{G},\mathcal{U}} = \sum_{i=1}^{n} r_{iS_0}^{\mathcal{U}}$.*

Although the algorithm is not efficient, we can use it to accurately calculate multiplex influence spread for multiplex networks with a small number of agents. Next, we propose an efficient approach that sacrifices accuracy to calculate multiplex influence spread for large networks.

9.7 A Bayesian Network Approach

In this section, we map the problem of computing multiplex influence spread into a problem of probabilistic inference in Bayesian networks (BN). This means we can compute multiplex influence

spread by using an appropriate algorithm for inference in BNs, such as the loopy belief propagation algorithm. We first recall the definition of a BN.

Definition 23 (Bayesian network)**.** *Let $G = (V, E)$, where $V = 1, 2, ..., n$ and $E \subset V \times V$, be a directed acyclic graph (DAG). Each node $i \in V$ is associated with a random variable $x'_i \in \mathcal{X}'_i$. Denote the set of out-neighbors of $i \in V$ as N'_i. Let $\mathbb{P}(x'_i|x'_{N'_i})$ be the probability of x'_i conditioned on the states of nodes in N'_i. Then G is a* Bayesian network *if the joint distribution of the random variables is factorized into conditional probabilities: $\mathbb{P}(x'_1, x'_2, ..., x'_n) = \prod_{i=1}^{n} \mathbb{P}(x'_i|x'_{N'_i})$.*

Probabilistic inference in a Bayesian network refers to calculating the marginal probability of the state of each unobserved node, conditioned on the states of the observed nodes. The belief propagation (BP) algorithm was first proposed by Pearl [68] to solve probabilistic inference in Bayesian networks. Pearl [68] showed that the algorithm is exact on DAGs without loops, i.e., trees and polytrees [68]. It is not guaranteed to converge when applied to DAGs with loops. However, Murphy et al. [58] showed that loopy belief propagation (LBP) - the application of Pearl's algorithm to DAGs with loops - provides a good approximation when it converges. The junction tree algorithm [51] was proposed to perform exact inference on a general graph. The general graph is first modified with additional edges to make it a junction tree, and then belief propagation is performed on the modified network.

In the following algorithm, we show how the joint distribution of the probability of activation of agents in a class of LTM admits the graphical structure of Bayesian network.

Algorithm 5 (Bayesian network from multiplex LTM)**.** *Given multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG and sequence of protocols $\mathcal{U}$:*

1. *Let $G =$ proj($\mathcal{G}$) be the underlying DAG for the Bayesian network. Then $N'_i = N_i = \cup_{k=1}^{m} N_i^k$.*
2. *Let the random variable x'_i of node i in the Bayesian network be $\bar{x}_i$, the steady-state value of agent i for the multiplex LTM on $\mathcal{G}$. Then $x'_i \in \mathcal{X}'_i = \{0, 1\}$.*
3. *Construct the conditional probabilities for the Bayesian network in terms of the conditional probabilities for the multiplex LTM: $\mathbb{P}(x'_i|x'_{N_i}) = \mathbb{P}^{\mathcal{G},u_i}(\bar{x}_i|\bar{x}_{N_i})$.*

Since all random variables are discrete, the conditional probability $\mathbb{P}(x'_i|x'_{N_i}) = \mathbb{P}^{\mathcal{G},u_i}(\bar{x}_i|\bar{x}_{N_i})$ can be fully described with a conditional probability table (CPT). We show how to construct a CPT for $\mathbb{P}^{\mathcal{G},u_i}(\bar{x}_i|\bar{x}_{N_i})$ for the Fig. 9.4 example. The CPT of i has $2^{|N_i|}$ rows. For agent 6, $N_6 = \{3, 4, 5\}$, $\bar{x}_{N_6} = \{\bar{x}_3, \bar{x}_4, \bar{x}_5\}$ and its CPT has $2^{|N_6|} = 8$ rows. The CPT provides $\mathbb{P}^{\mathcal{G},u_6}(\bar{x}_6 = 0|\bar{x}_3, \bar{x}_4, \bar{x}_5)$ and

$\mathbb{P}^{\mathcal{G},u_6}(\bar{x}_6 = 1|\bar{x}_3, \bar{x}_4, \bar{x}_5)$. Table 9.1 and Table 9.2 are the CPTs for agent 6 when it uses Protocol OR and Protocol AND, respectively.

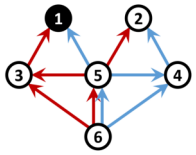

Figure 9.4: Multiplex network $\mathcal{G}_s$ with two unweighted layers and six agents. Red (blue) arrows represent edges in layer 1 (layer 2). $S_0 = \{1\}$.

Table 9.1: CPT of agent 6 with $u_6 = \text{OR}$ for $\mathcal{G}_s$ of Fig. 9.4

$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$	$\mathbb{P}^{\text{OR}}(\bar{x}_6=0\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$	$\mathbb{P}^{\text{OR}}(\bar{x}_6=1\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$
0	0	0	1.00	0.00
0	0	1	0.25	0.75
0	1	0	0.50	0.50
0	1	1	0.00	1.00
1	0	0	0.50	0.50
1	0	1	0.00	1.00
1	1	0	0.25	0.75
1	1	1	0.00	1.00

Table 9.2: CPT of agent 6 with $u_6 = \text{AND}$ for $\mathcal{G}_s$ of Fig. 9.4

$\bar{x}_3$	$\bar{x}_4$	$\bar{x}_5$	$\mathbb{P}^{\text{AND}}(\bar{x}_6=0\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$	$\mathbb{P}^{\text{AND}}(\bar{x}_6=1\|\bar{x}_3,\bar{x}_4,\bar{x}_5)$
0	0	0	1.00	0.00
0	0	1	0.75	0.25
0	1	0	1.00	0.00
0	1	1	0.50	0.50
1	0	0	1.00	0.00
1	0	1	0.50	0.50
1	1	0	0.75	0.25
1	1	1	0.00	1.00

We focus on the case when $\text{proj}(\mathcal{G})$ is a DAG. The case when $\text{proj}(\mathcal{G})$ is not a DAG can be handled by by combining the junction tree algorithm and belief propagation. However, a subsequent marginalization within the appropriate junction node maybe required to obtain the desired probability.

Theorem 7. *Given a multiplex network $\mathcal{G}$ for which $\text{proj}(\mathcal{G})$ is a DAG, with seed set S_0 and sequence of protocols $\mathcal{U}$, the following two probabilities are the same:*

1. $\mathbb{P}_{S_0}^{\mathcal{G},\mathcal{U}}(\bar{x}_i = 1)$, *the probability that agent i is active at steady state for the multiplex LTM.*

2. $\mathbb{P}^{\mathcal{G},\mathcal{U}}(\bar{x}_i = 1 | \bar{x}_j = 1, \bar{x}_l = 0, j \in S_0, l \notin S_0, N_l = \varnothing)$, *the marginal probability of node i in the corresponding Bayesian network of Algorithm 5, conditioned on observed nodes in the seed set and those not in the seed set that have no out-neighbors.*

Proof. Since the event that node i is activated is conditioned on the state of node i's out-neighbors in proj($\mathcal{G}$), it is independent of the state of all other nodes. Thus, the joint probability of activation of each node i factors into components based on the graphical structure of the Bayesian network constructed in Algorithm 5. So, the probability that agent i is active at steady state for the multiplex LTM (probability in the first statement) can be computed by probabilistic inference on the corresponding Bayesian network (the marginal probability in the second statement). The values of the observed nodes in the marginal probability are obtained as follows. Each $j \in S_0$ is always active, so we observe $\bar{x}_j = 1$. The state x_l of l that has no out-neighbors does not change over time. If also $l \notin S_0$, then l is always inactive, and we observe $\bar{x}_l = 0$. □

The equivalence in Theorem 7 implies that we can compute multiplex influence spread using probabilistic inference in BNs, which can be solved with BP algorithms.

Corollary 5. *Given a multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG, with seed set S_0 and sequence of protocols $\mathcal{U}$, multiplex influence spread $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$ can be computed as*

$$\sum_{i=1}^{n} \mathbb{P}^{\mathcal{G},\mathcal{U}}(\bar{x}_i = 1 | \bar{x}_j = 1, \bar{x}_l = 0, j \in S_0, l \notin S_0, N_l = \varnothing).$$

Proof. This follows from Theorem 7 and Definition 21. □

Nguyen and Zheng [64] showed that computing influence spread in a monoplex DAG with the Independent Cascade Model (ICM) is #P-complete. Here, we prove a similar result for computing influence spread for the multiplex LTM. The implication is that approximating influence spread for the multiplex LTM is the best we can do for large networks.

Theorem 8. *Consider a multiplex network $\mathcal{G}$ for which* proj($\mathcal{G}$) *is a DAG, with seed set S_0 and sequence of protocols $\mathcal{U}$. Computing $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$ for the multiplex LTM is #P-complete.*

Proof. The multiplex LTM problem, i.e., computing $\sigma_{S_0}^{\mathcal{G},\mathcal{U}}$ for the multiplex LTM, is #P-complete if (i) it is #P-hard and (ii) it is in #P. We first prove (ii). By Corollary 5, every instance of the multiplex LTM problem can be reduced to a marginalization problem. Since the marginalization problem is #P-complete, the multiplex LTM problem is in #P.

We prove (i) by showing that the multiplex LTM problem is a reduction from the ICM problem, which has been shown to be #P-complete (Theorem 1 of [64]). The ICM problem refers to computing the influence spread $\sigma_{S_0}^{G,\text{ICM}}$ for the ICM on a monoplex DAG $G = (V, E)$ with seed set S_0 and probability $w_{j,i}$ assigned to each edge $e_{j,i} \in E$.

Let m be the largest number of out-neighbors over all nodes in V. Consider a multiplex network with m layers $G_1, ..., G_m$ where $G_k = (V, E^k)$. To define edge sets E^k, assign all edges in G and to the multiplex network such that for each node there is at most one outgoing edge in each layer k. Let $w_{j,i}$ be the weight of edge from i to j in the multiplex network. Define a set V' with a node $i' \in V'$ for each node $i \in V$. For each i and k, compute the sum of weights of i's outgoing edges in layer k. If the sum is not 1, create an edge $e_{i,i'}^k \in E'^k$ from i to i' and assign to the edge a weight that makes the sum equal 1. Let multiplex network $\mathcal{G}'$ have m layers $G'_1, ..., G'_m$ where $G'_k = (V \cup V', E^k \cup E'^k)$. Then proj($\mathcal{G}'$) is a DAG. Further, every node $i' \in V'$ has no out-neighbors and $i' \notin S_0$. So in the multiplex LTM $i' \in V'$ remains inactive. Let $\mathcal{U} = \{\text{OR}, ...\text{OR}\}$. By construction, $\sigma_{S_0}^{\mathcal{G}',\mathcal{U}} = \sigma_{S_0}^{G,\text{ICM}}$. □

9.8 Analytical Expressions of Influence Spread

We derive analytical expressions for multiplex cascade centrality for two illustrative classes of the LTM with a two-layer (duplex) network and N agents. In Section 9.8.1, each of the layers is the same path network. This means that each agent has the same neighbors for each sensing modality; for example, each agent can see and hear its neighbors. Our results reveal how the cascade is affected when agents distinguish between signals rather than project them. If we view the path network as a cycle network with one link missing, then the duplex permutation network we study in Section 9.8.2 has one layer missing the link between agents 1 and N and the other layer missing the link between agents $N - 1$ and N.

9.8.1 Duplex Repeated Path Network

Let $\mathcal{G}_R$ be the duplex repeated path network of Fig. 9.5, which has the monoplex path network G_{Pa} = proj($\mathcal{G}_R$) on each of its two layers. For any N and any agent j, monoplex cascade centrality $C_j^{G_{Pa}}$ and multiplex cascade centralities $C_j^{\mathcal{G}_R,\text{OR}}$ and $C_j^{\mathcal{G}_R,\text{AND}}$ can be expressed analytically as follows.

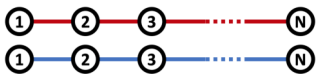

Figure 9.5: Duplex repeated path network $\mathcal{G}_R$ has path graph G_{Pa} as each layer.

Proposition 8 (Multiplex cascade centrality for $\mathcal{G}_R$)**.** *Consider the monoplex path network G_{Pa} and duplex repeated path network $\mathcal{G}_R$ for N agents with $u_i = u \in \{OR, AND\}$. Then*

$$C_j^{G_{Pa}} = h_j(.5), \;\; C_j^{\mathcal{G}_R,\text{OR}} = h_j(.75), \;\; C_j^{\mathcal{G}_R,\text{AND}} = h_j(.25),$$

$$h_j(p_0) = \begin{cases} \sum_{l=0}^{N-2} p_0^l + p_0^{N-2}, & j \in \{1, N\} \\ 1 + \sum_{l=0}^{N-3} p_0^l + p_0^{N-3}, & j \in \{2, N-1\} \\ \sum_{l=0}^{j-1} p_0^l + p_0^{j-1} + \sum_{l=1}^{N-j-1} p_0^l + p_0^{N-j-1}, & \text{o.w.} \end{cases}$$

Moreover,

$$C_j^{\mathcal{G}_R,\text{OR}} > C_j^{G_{Pa}} > C_j^{\mathcal{G}_R,\text{AND}}.$$

Proof. The result can be derived from Algorithm 1. Here, we provide a perspective from probabilistic inference in BNs. Table 9.3 shows the CPT of agent $i \in \{2, \ldots, N-1\}$ for G_{Pa} and for $\mathcal{G}_R$ with $u = \text{OR}$ and $u = \text{AND}$. With one initially active agent j, the activity can only spread from an agent

Table 9.3: CPT of agent $i \in \{2, 3, ..., N-1\}$. For G_{Pa}, $p_0 = .5$; for $\mathcal{G}_R$ with $u = \text{OR}$, $p_0 = .75$; for $\mathcal{G}_R$ and $u = \text{AND}$, $p_0 = .25$.

x_{i-1}	x_{i+1}	$\mathbb{P}^{\mathcal{G},u}(x_i = 0 \mid x_{i-1}, x_{i+1})$	$\mathbb{P}^{\mathcal{G},u}(x_i = 1 \mid x_{i-1}, x_{i+1})$
0	0	1	0
0	1	$1 - p_0$	p_0
1	0	$1 - p_0$	p_0
1	1	0	1

closer to j to an agent farther from j. Assume $1 < j < i < N$, for all three networks, the probability that agent i is active at steady state for the LTM can be factorized as follows:

$$\begin{aligned} \mathbb{P}_j^{\mathcal{G},u}(\bar{x}_i = 1) = \mathbb{P}^{\mathcal{G},u}(\bar{x}_i = 1 \mid \bar{x}_{i-1} = 1, x_{i+1} = 0) \\ \times \mathbb{P}^{\mathcal{G},u}(\bar{x}_{i-1} = 1 \mid \bar{x}_{i-2} = 1, x_i = 0) \times \cdots \\ \times \mathbb{P}^{\mathcal{G},u}(\bar{x}_{j+1} = 1 \mid \bar{x}_j = 1, x_{j+2} = 0) = p_0^{i-j}. \end{aligned} \tag{9.10}$$

The last equality holds since each agent uses the same protocol and so each conditional probability is p_0. The cascade centralities follow by Definitions 21 and 22. Cases $i, j \in \{1, N\}$ are calculated similarly. The inequality follows since $\mathbb{P}_j^{\mathcal{G}_R,\text{OR}}(\bar{x}_i = 1) > \mathbb{P}_j^{G_{Pa}}(\bar{x}_i = 1) > \mathbb{P}_j^{\mathcal{G}_R,\text{AND}}(\bar{x}_i = 1)$. □

Proposition 8 provides a systematic way to evaluate spread for any number of agents N in the multiplex LTM on $\mathcal{G}_R$. The inequality is consistent with the intuition in Remark 1: when agents can

distinguish signals from different sensing modalities and use Protocol OR (AND), they are more (less) easily activated, and the cascade is enhanced (diminished) relative to when agents cannot distinguish signals.

9.8.2 Duplex Permutation Networks

Let $\mathcal{G}_P$ be the duplex permutation network of Fig. 9.6. Then proj$(\mathcal{G}_P) = G_C$, the cycle network. We derive analytical expressions for $\mathbb{P}_j^{\mathcal{G}_P,u}(\bar{x}_i = 1)$, $u \in \{\text{OR}, \text{AND}\}$, as follows.

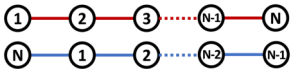

Figure 9.6: Duplex permutation network $\mathcal{G}_P$.

Proposition 9 (Probabilities for multiplex cascade centrality for $\mathcal{G}_P$)**.** *Consider the duplex permutation network $\mathcal{G}_P$ for N agents with $u_i = u \in \{OR, AND\}$ and the cyclic network $G_C = proj(\mathcal{G}_P)$. Then*

$$\begin{aligned}
\mathbb{P}_j^{\mathcal{G}_P,\text{OR}}(\bar{x}_i = 1) &= (.75)^{|i-j|} + .5(.75)^{N-|i-j|-3} - .5(.75)^{N-5} \\
\mathbb{P}_j^{\mathcal{G}_P,\text{AND}}(\bar{x}_i = 1) &= (.25)^{|i-j|} \\
\mathbb{P}_j^{G_C}(\bar{x}_i = 1) &= (.5)^{|i-j|} + (.5)^{N-|i-j|}
\end{aligned} \tag{9.11}$$

where $3 \le i \le N-3$ and $j = 2, \ldots, i-2, i+2, \ldots, N-2$. Moreover,

$$C_j^{\mathcal{G}_P,\text{OR}} > C_j^{G_C} > C_j^{\mathcal{G}_P,\text{AND}}$$

$$\mathbb{P}_j^{\mathcal{G}_P,\text{AND}}(\bar{x}_i = 1) = \mathbb{P}_j^{\mathcal{G}_R,\text{AND}}(\bar{x}_i = 1) = (0.25)^{|i-j|}$$

$$\mathbb{P}_j^{\mathcal{G}_P,\text{OR}}(\bar{x}_i = 1) > \mathbb{P}_j^{\mathcal{G}_R,\text{OR}}(\bar{x}_i = 1).$$

Proof. The probabilities derive from Algorithm 1. The first inequality follows since $\mathbb{P}_j^{\mathcal{G}_P,\text{OR}}(\bar{x}_i = 1) > \mathbb{P}_j^{G_C}(\bar{x}_i = 1) > \mathbb{P}_j^{\mathcal{G}_P,\text{AND}}(\bar{x}_i = 1)$. The rest follows from (9.10) and (9.11). □

When $u_N = \text{AND}$, the activity can only spread in $\mathcal{G}_P$ from j to i along the path between j and i on G_C that does not contain N. This explains the equality of probabilities for $\mathcal{G}_P$ and $\mathcal{G}_R$. When $u_i = u = \text{OR}$, the activity can spread in $\mathcal{G}_P$ from j to i along either path between j and i on G_C. This explains the last inequality, i.e., that the cascade is greater in $\mathcal{G}_P$ than in $\mathcal{G}_R$. As in Proposition 8, the first inequality in Proposition 9 is consistent with the intuition in Remark 1.

9.9 Heterogeneity in Protocol

9.9.1 Small Heterogeneous Multiplex Networks

We compute multiplex cascade centrality to evaluate for the LTM the role of heterogeneity in the tradeoff between sensitivity of the cascade to a real input and robustness of the cascade to a spurious signal. Knowing that agents that use Protocol OR enhance the cascade and agents that use AND diminish the cascade, we examine how to leverage heterogeneity in protocol to advantage. Parametrizing the tradeoff by c, we solve as a function of c for the optimal heterogeneous distribution of agents using OR and agents using AND.

We investigate with the duplex network of Fig. 9.7, which is small enough that we can compute cascade centrality with Algorithm 1. There are six agents (nodes 1 to 6) and a seventh node that represents an external signal. When node 7 appears in both layers, as in Fig. 9.7, we interpret it as real. When node 7 appears in only one layer, we interpret it as spurious. We assume that only one agent (e.g., agent 1 in Fig. 9.7) senses node 7, whether or not it is real or spurious, and that it is equally likely to be any of the six agents that sense it. We assume the edge pointing to the signal has a weight of 1.

The graph in layer 1 (red edges) in Fig. 9.7 represents a *directed sensing* modality, e.g., a team of robots with front and side facing cameras or a school of fish that see poorly to their rear. The graph in layer 2 (blue edges) represents a *proximity sensing* modality, e.g., robots that receive local broadcasts or fish that detect local movement with their lateral line.

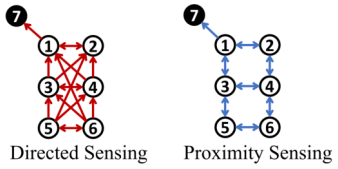

Figure 9.7: Duplex network with agents 1 to 6, layer 1 (red), and layer 2 (blue). Node 7, the external signal, is real since it appears in both layers.

As each agent can use either protocol, there are $2^6 = 64$ different possible sequences of protocols $\mathcal{U}$ in total. We define the utility function Q as a function of $\mathcal{U}$ and $c \geq 0$ to measure the benefit of cascades that result from real signals less the cost of cascades that result from spurious signals:

$$Q(\mathcal{U}, c) = \frac{1}{6} \sum_{l=1}^{6} \left(C_7^{\mathcal{G}^l_{\text{real}}, \mathcal{U}} - c \frac{1}{2} (C_7^{\mathcal{G}^l_{\text{spur1}}, \mathcal{U}} + C_7^{\mathcal{G}^l_{\text{spur2}}, \mathcal{U}}) \right). \tag{9.12}$$

Superscript l indexes the agent sensing node 7. Subscripts “real", “spur1" and “spur2" index the networks where node 7 appears in both layers, layer 1 only, and layer 2 only, respectively. Increasing c increases cost of response to spurious signals relative to benefit of response to real signals. Given c, the optimal sequence of protocols is $\mathcal{U}^c = \text{argmax}_{\mathcal{U}} Q(\mathcal{U}, c)$.

Fig. 9.8 illustrates $\mathcal{U}^c$ (with symmetry implied) on a plot of the optimal fraction of agents using AND as a function of c. A white (gray) circle represents an agent using OR (AND). When c is 0 or small, responding to real signals dominates and all agents use OR. When c is increases towards 3 and beyond, avoiding spurious signals dominates and all agents use AND. For c in between, the optimal solution is heterogeneous with more agents using AND as c increases: first agent 1 or 2, then agents 1 and 2, then {1,2,5} or {1,2,6}, then {1,2,3,6} or {1,2,4,5}, and then {1,2,3,5,6} or {1,2,4,5,6}.

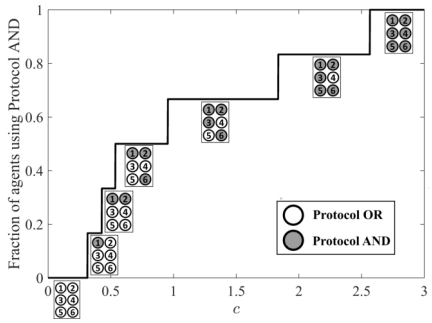

Figure 9.8: The optimal fraction of agents using Protocol AND with illustration of optimal solution $\mathcal{U}^c$ as c varies from 0 to 3. Symmetry is implied.

9.9.2 Large Heterogeneous Multiplex Networks

We apply Corollary 5 to study multiplex cascade centrality for a random multiplex network with 20 agents and homogeneous and heterogeneous protocols. We randomly generate duplex networks, for which the projection networks are DAGs, by fixing a topological order of nodes and assigning edges randomly with probability p_e. A higher probability p_e means agents sense a greater number of the other agents. We consider homogeneous groups, where u = OR and u = AND. We also consider heterogeneous groups in which each agent randomly chooses OR or AND with equal likelihood.

We let the root node be the initial active agent and study how the cascade centrality of the root changes as we vary p_e from 0 to 1. For every value of p_e, we randomly generate 400 networks. Fig. 9.9 shows how the cascade centrality, averaged over the random networks, changes as a function of p_e.

Regardless of the protocol, as $p_e \to 0$, the DAG become disconnected and the cascade centrality goes to 1 (only the root node is active at steady state). As $p_e \to 1$, the root node activates every other

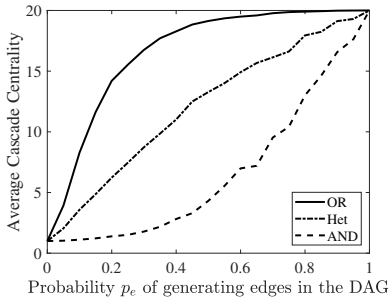

Figure 9.9: Multiplex cascade centrality of root node, averaged over 400 networks, as a function of probability p_e of edges in the DAG.

agent and the cascade centrality goes to 20. For p_e in between, Fig. 9.9 shows that the homogeneous groups with u = OR are most readily activated and cascade size is sensitive in the range $p_e \in (0, .5)$. Homogeneous groups with u = AND are least readily activated and cascade size is sensitive in the range $p_e \in (.5, 1)$.

9.10 Conclusion

We have extended the LTM to multiplex networks where agents use different protocols that distinguish signals from multiple sensing modalities. We have derived algorithms to compute influence spread accurately using the multiplex LEM and approximately using probabilistic inference. We have shown how multiple sensing modalities affect spread and how heterogeneity trades off sensitivity and robustness of spread.

Chapter 10

A Continuous Threshold Model of Cascade Dynamics

Yaofeng Desmond Zhong, Naomi Ehrich Leonard

Appears as Zhong and Leonard [98] in *2019 IEEE 58th Conference on Decision and Control*

We present a continuous threshold model (CTM) of cascade dynamics for a network of agents with real-valued activity levels that change continuously in time. The model generalizes the linear threshold model (LTM) from the literature, where an agent becomes active (adopts an innovation) if the fraction of its neighbors that are active is above a threshold. With the CTM we study the influence on cascades of heterogeneity in thresholds for a network comprised of a chain of three clusters of agents, each distinguished by a different threshold. The system is most sensitive to change as the dynamics pass through a bifurcation point: if the bifurcation is supercritical the response will be contained, while if the bifurcation is subcritical the response will be a cascade. We show that there is a subcritical bifurcation, thus a cascade, in response to an innovation if there is a large enough disparity between the thresholds of sufficiently large clusters on either end of the chain; otherwise the response will be contained.

10.1 Introduction

Cascade dynamics refer to the spread of an activity or innovation among a group of agents. They have been modelled as discrete-time, discrete-valued state dynamics in which an agent accepts

or rejects an innovation at each time step after comparing the fraction of its neighbors who have accepted the innovation to a threshold between 0 and 1. This model is referred to as the linear threshold model (LTM).

The LTM was first introduced in [31, 80]. Kempe et al. [45] and Lim et al. [55] studied the LTM with uniformly drawn thresholds. Zhong et al. [100] generalized the LTM to duplex networks where there exist two different types of interactions among the agents. All of these results leverage thresholds drawn from a uniform distribution. Acemoglu et al. [1] studied cascade dynamics using the LTM with deterministic thresholds. In this case, the analysis becomes challenging. Yang et al. [95] studied the influence minimization problem for the deterministic LTM by formulating the problem as a linear integer programming problem. Fardad and Kearney [24] studied the optimal seeding problem of cascade failure using a relaxation of the deterministic LTM and formulated the problem as a convex program. Pinheiro et al. [69] introduced nonlinearity to the fraction of neighbors and investigated cascades on the all-to-all network. However, the analysis of cascades remains challenging for more general network graphs, heterogeneous agents, and deterministic thresholds.

We propose a continuous threshold model (CTM) that introduces continuous-time, real-valued state dynamics with nonlinearity. The CTM is adapted from nonlinear consensus dynamics [32], which exhibit very rapid transitions in system state when the system passes through a bifurcation point. Using the proposed model, we can thus prove conditions for sudden cascades using methods from nonlinear dynamics.

We show how the CTM generalizes the LTM. We then use the CTM to study the influence on cascades of heterogeneity in (deterministic) thresholds among the agents. We consider networks comprised of three clusters of agents, each cluster associated with a different threshold and thus a different level of responsiveness to the state of neighbors. A lower threshold implies a higher responsiveness. Let $\epsilon > 0$. Cluster 1 is the "high response" cluster where agents use a threshold $\mu = 1/2 - \epsilon$. Cluster 2 is the "low response" cluster where agents use a threshold $\mu = 1/2 + \epsilon$. Cluster 3 is the "neutral response" cluster where agents use a threshold $\mu = 1/2$.

For the networks considered, we show how the size n of clusters 1 and 2 and the strength of the disparity 2ϵ between their thresholds determines whether or not there is a cascade in response to the introduction of an innovation. When the size and disparity are small the group exhibits a contained response, whereas when the size and disparity are sufficiently large the group exhibits a rapid increase in state, indicating a cascade. This is an interesting and even surprising result.

In the analysis, the contained response corresponds to a supercritical pitchfork bifurcation in

the dynamics and the cascade corresponds to a subcritical pitchfork bifurcation. Gray. et al. [32] observed the transition from supercritical to subcritical pitchfork in nonlinear consensus dynamics; however, they did not prove conditions under which this transition exists. The transition is also exhibited in the replicator-mutator dynamics studied by Dey et al. [23]. They showed the existence of the transition for the system with two strategies. We derive rigorous conditions for existence of the transition in the CTM dynamics of a network of N agents comprised of three clusters distinguished by their thresholds.

Our contribution is twofold. First, we present a new model of cascade dynamics that generalizes the LTM. Second, we provide new results on how cascades are influenced by heterogeneity in thresholds of agents, i.e., how ready an agent is to change its behavior in response to its neighbors. For a network of three clusters, we derive a necessary condition for there to be a cascade that depends on the sizes of the clusters with disparate thresholds and N. We show that there is a corresponding critical value $\epsilon^* > 0$ such that we can expect a cascade when $\epsilon > \epsilon^*$ but not when $\epsilon < \epsilon^*$.

Section 10.2 describes the CTM dynamics and equivalence to the LTM. In Section 10.3, we specialize the dynamics to a family of network graphs with three clusters. In Section 10.4, we define a cascade for the CTM and prove conditions under which a cascade occurs. Section 10.5 provides an example.

10.2 Continuous Threshold Model

The proposed model generalizes the discrete linear threshold model (LTM), and it is inspired by the Hopfield network dynamics [40] and adapted from nonlinear consensus dynamics [32]. To describe complex contagion within a group of N agents, let $x_i \in \mathbb{R}$ be the state of agent i, representing the activity level of agent i. Agent i is said to be active (inactive) if $x_i > 0$ ($x_i < 0$). A greater absolute value of the state $|x_i|$ means that agent i is more active (more inactive).

Interactions among agents, i.e., who can sense or communicate with whom, are encoded in graph $G = (V, E)$, where $V = \{1, \ldots, N\}$ is the set of N agents and $E \subset V \times V$ is the edge set representing interactions. An edge $e_{ij} \in E$ implies that j is a neighbor of i. We assume there are no self-loops, i.e., $e_{ii} \notin E$. The graph adjacency matrix $A \in \mathbb{R}^{N \times N}$ is a matrix with elements of 0 and 1, where $a_{ij} = 1$ if and only if $e_{ij} \in E$. The degree matrix $D \in \mathbb{R}^{N \times N}$ is a diagonal matrix with diagonal entries $d_i = \sum_{j=1}^{N} a_{ij}$.

The *continuous threshold model* (CTM) defines the change in activity level of each agent over time

as a function of the agent's current state and the state of its neighbors:

$$\dot{x}_i = -d_i x_i + \sum_{j=1}^{N} a_{ij} u S(v x_j) + d_i(1 - 2\mu_i). \tag{10.1}$$

μ_i can be interpreted as the threshold of agent i and $u, v > 0$ as control parameters. $S : \mathbb{R} \to [-1, 1]$ is a smooth, odd sigmoidal function that satisfies the following conditions: $S'(x) > 0, \forall x \in \mathbb{R}$; $S'(0) = 1$; and $\text{sgn}(S''(x)) = -\text{sgn}(x)$, where $(\cdot)'$ denotes the derivative and sgn is the sign function. Sigmoids are ubiquitous in models of biological and physical systems; here they serve to saturate influence from neighbors.

The control parameter u can be interpreted as the strength of the "social sensitivity" since a larger u means a greater attention to social cues. The control parameter v can be interpreted as the strength of the "social effort" since a larger v means a stronger signal sent by neighbors. The CTM generalizes the LTM in the following way. Let $u = 1$ and $v \to +\infty$, then the dynamics (10.1) become

$$\frac{1}{d_i}\dot{x}_i = -x_i + 2\Big(\sum_{j=1}^{N} \frac{a_{ij}}{d_i} \frac{S(v x_j) + 1}{2} - \mu_i\Big). \tag{10.2}$$

Since $v \to +\infty$, the sigmoidal function approaches the sign function, which maps a negative state to -1 and a positive state to +1. The fraction $(S(v x_j) + 1)/2$ maps a negative state to 0 and a positive state to 1. So the summation gives the fraction of active neighbors. Thus, the difference between the summation and μ_i is the comparison of fraction of active neighbors of agent i to the threshold of agent i.

Now consider the equilibrium points of (10.2). If agent i's fraction of active neighbors is greater than its threshold, the steady-state value of x_i is positive and agent i is active; otherwise, the steady-state value of x_i is negative and agent i is inactive. Let unseeded agents be defined by a negative initial state and seeded agents by a large positive initial state. Then it follows that the dynamics (10.2) behave like the LTM with deterministic thresholds μ_i, $i = 1, \ldots, N$ in the sense that the ordering of unseeded agents switching from inactive to active is the same and hence the steady states are the same.

We study the CTM for $v = 1$ and u a feedback control that depends on the *slow* filtered average state $\bar{x}_s$. We let $u = u_0 S(\kappa|\bar{x}_s|)$ and $\dot{\bar{x}}_s = \kappa_s(\bar{x} - \bar{x}_s)$ with $\bar{x} = \sum_{i=1}^{N} x_i/N$ and $u_0, \kappa, \kappa_s > 0$.

10.3 Networks with Three Clusters

Consider a class of networks with N agents in three clusters as in Section 10.1: every agent i in the high response cluster 1 has $\mu_i = 1/2 - \epsilon$, every agent j in the low response cluster 2 has $\mu_j = 1/2 + \epsilon$ and every agent k in the neutral response cluster 3 has $\mu_k = 1/2$. Let there be n agents in cluster 1, n in cluster 2, and $N - 2n$ in cluster 3.

Let all edges in the network be undirected, i.e., $a_{ij} = a_{ji}$. Within each cluster, the graph is all-to-all. Each agent in cluster 3 is connected to each agent in clusters 1 and 2, and there are no connections between agents in cluster 1 and agents in cluster 2. The network is motivated by environments in which a feature that influences threshold adoption is distributed and agents with biased thresholds interact with agents with similarly biased or unbiased thresholds. One example is a population clustered by age bracket, where the youngest are most likely and the eldest least likely to purchase a new technology when their friends do. Another is a spatially distributed group in which agents at one end measure smoke and adopt a low threshold to flee a fire, and agents on the other end miss the smoke and adopt a high threshold. Fig. 10.1 shows a network with $N = 11$ and $n = 4$.

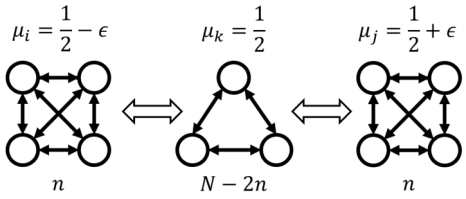

Figure 10.1: Network with three clusters: $N = 11$ and $n = 4$. Cluster 1 (high response) is on the left, cluster 2 (low response) is on the right, and cluster 3 (neutral response) is in the middle. White arrows indicate all-to-all, undirected connections between nodes in clusters.

With an approach similar to that in Theorem 4 of [32], it can be shown that the trajectories of (10.1) converge exponentially to the three-dimensional manifold where all the states in the same

cluster are the same. Let y_k be the average state of cluster $k = 1, 2, 3$. The reduced dynamics are

$$\begin{aligned}
\dot{y}_1 =& -(N-n-1)y_1 + (n-1)uS(y_1) \\
&+ (N-2n)uS(y_3) + 2(N-n-1)\epsilon \qquad (10.3)\\
\dot{y}_2 =& -(N-n-1)y_2 + (n-1)uS(y_2) \\
&+ (N-2n)uS(y_3) - 2(N-n-1)\epsilon \qquad (10.4)\\
\dot{y}_3 =& -(N-1)y_3 + (N-2n-1)uS(y_3) \\
&+ nuS(y_1) + nuS(y_2). \qquad (10.5)
\end{aligned}$$

Let $\mathbf{y} = [y_1, y_2, y_3]^T$ and $F(\mathbf{y}, u, \epsilon)$ the RHS of (10.3)-(10.5).

F commutes with the action of the nontrivial element of Z_2, the cyclic group of order 2, represented by the matrix:

$$\gamma = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix}, \quad \text{i.e., } F(\gamma\mathbf{y}, u, \epsilon) = \gamma F(\mathbf{y}, u, \epsilon).$$

This implies a Z_2-symmetric singularity. We show in the next section that F possesses a pitchfork bifurcation, and we prove a necessary condition for the transition from a supercritical to a subcritical pitchfork.

The bifurcations are illustrated in Fig. 10.2 where the horizontal axis represents the bifurcation parameter u and the vertical axis the average state $\bar{y} = (ny_1 + ny_2 + (N-2n)y_3)/N$. Blue curves represent stable solutions and red curves unstable solutions to (10.3)-(10.5). The neutrally active average state $\bar{y} = 0$ is always a solution, and it is stable for $u < u^c$ and unstable for $u > u^c$, where $u = u^c$ is the bifurcation point.

Due to the feedback, the social sensitivity parameter u will slowly increase when an innovation has been introduced and cross the bifurcation point where the system is highly sensitive to change (see [32] for generalizations to heterogeneous u_i). For initial conditions corresponding to one or more active agents such that $\bar{y}(0) > 0$, the solution will increase as shown by the green curves in Fig. 10.2. The trajectory in the supercritical pitchfork slowly follows the positive branch of the pitchfork as u is increased just above the critical value u^c_{sup}. We define this slow increase in $\bar{y}$ as a *contained response*. The trajectory in the supercritical pitchfork jumps up to the positive branch as u is increased just above the critical value u^c_{sub}. We define the jump as a *cascade* since it implies a

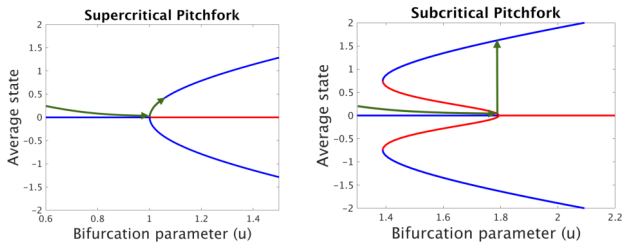

Figure 10.2: Pitchfork bifurcation diagrams: supercritical (left) and subcritical (right). Blue (red) curves are stable (unstable) solutions. Green curves are trajectories as u slowly increases.

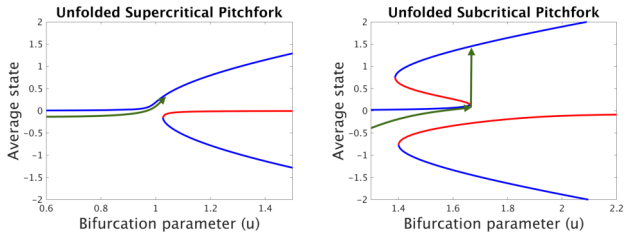

Figure 10.3: Unfolded pitchfork bifurcation diagrams: supercritical (left) and subcritical (right). Colors are as in Fig. 10.2.

rapid spread of the innovation.

Fig. 10.3 shows what happens to the bifurcation diagrams in the presence of a small positive input to the dynamics (10.3), corresponding to the introduction of an innovation as an external cue rather than as seeded positive initial conditions. The resulting "unfolded" supercritical pitchfork still exhibits the contained response and the "unfolded" subcritical pitchfork still exhibits the cascade. Here, even with an initial condition corresponding to an average initial state $\bar{y}(0) < 0$, the innovation can still trigger a cascade.

10.4 Conditions for Cascade

Although the transition from supercritical pitchfork to subcritical pitchfork has been observed in [32], it is unclear for what parameter values the transition exists in the CTM. In this section, we first show conditions for existence of a pitchfork bifurcation and then for existence of the transition.

By Z_2-symmetry, $\mathbf{y}^\star = [y^\star, -y^\star, 0]^T$ is always an equilibrium of (10.3)-(10.5), where for a given u and ϵ, $y^\star$ satisfies

$$-(N-n-1)y^\star + (n-1)uS(y^\star) + 2(N-n-1)\epsilon = 0. \tag{10.6}$$

Consider a perturbation to the trivial solution $\mathbf{y}^\star$ and denote the perturbed solution as $\mathbf{y}^\star + \Delta\mathbf{y} =$

$[y^\star + \Delta y_1, -y^\star + \Delta y_2, \Delta y_3]^T$. We ask the question, could there be a nontrivial equilibrium point where $\Delta \mathbf{y} \neq \mathbf{0}$? The change from no nontrivial equilibria to the existence of nontrivial equilibria corresponds to the bifurcation point. The following perturbation analysis allows us to reduce the dynamics near the trivial equilibrium to one-dimensional dynamics that match the normal form of a pitchfork bifurcation. We can then evaluate if the pitchfork bifurcation is supercritical or subcritical by examining the sign of coefficients in the reduction.

We use the Taylor series expansion to third order:

$$\begin{aligned} S(y^\star+\Delta y_1) =& S(y^\star) + S'(y^\star)\Delta y_1 + \frac{1}{2}S''(y^\star)(\Delta y_1)^2 \\ &+ \frac{1}{6}S'''(y^\star)(\Delta y_1)^3 + o((\Delta y_1)^3) \qquad (10.7) \\ S(-y^\star+\Delta y_2) =& S(-y^\star)+S'(-y^\star)\Delta y_2+\frac{1}{2}S''(-y^\star)(\Delta y_2)^2 \\ &+ \frac{1}{6}S'''(-y^\star)(\Delta y_2)^3 + o((\Delta y_2)^3) \qquad (10.8) \\ S(\Delta y_3) =& S'(0)\Delta y_3 + \frac{1}{2}S''(0)(\Delta y_3)^2 \\ &+ \frac{1}{6}S'''(0)(\Delta y_3)^3 + o((\Delta y_3)^3). \qquad (10.9) \end{aligned}$$

Since the sigmoidal is an odd function, we have that $S(y^\star) = -S(-y^\star)$, $S'(y^\star) = S'(-y^\star)$, $S''(y^\star) = -S''(-y^\star)$ and $S'''(y^\star) = S'''(-y^\star)$. Without loss of generality, we use the hyperbolic tangent as the sigmoidal function from now on. For other types of sigmoidal functions, the analysis follows similarly. Now Eqn. (10.9) has the form of

$$\tanh(\Delta y_3) = \Delta y_3 - \frac{1}{3}\Delta y_3^3 + o((\Delta y_3)^3). \qquad (10.10)$$

Lemma 8 and Proposition 10 will be used to derive the conditions for transition from supercritical to subcritical bifurcation.

Lemma 8. *Assume $\Delta x \in \mathbb{R}$ and $\Delta y \in \mathbb{R}$ have small magnitudes and satisfy*

$$a(\Delta x)^3 + b(\Delta x)^2 + c\Delta x = -\Delta y + \frac{1}{3}(\Delta y^3) + o((\Delta y)^3). \qquad (10.11)$$

Then we have

$$\Delta x = -\frac{1}{c}\Delta y - \frac{b}{c^3}(\Delta y)^2 + \left(\frac{1}{3c} - \frac{2b^2}{c^5} + \frac{a}{c^4}\right)(\Delta y)^3 + o((\Delta y^3)).$$

Proof. Assume $\Delta x = \alpha_1 \Delta y + \alpha_2 (\Delta y)^2 + \alpha_3 (\Delta y)^3 + o((\Delta y)^3)$, then we have

$$
\begin{aligned}
(\Delta x)^2 &= \alpha_1^2 (\Delta y)^2 + 2\alpha_1 \alpha_2 (\Delta y)^3 + o((\Delta y)^3) \\
(\Delta x)^3 &= \alpha_1^3 (\Delta y)^3 + o((\Delta y)^3).
\end{aligned}
$$

Substitute the above equations into Eqn. (10.11) and equate the coefficients in front of Δy, $(\Delta y)^2$, $(\Delta y)^3$ in the LHS and RHS, respectively. We get three equations. Then we solve for α_1, α_2 and α_3 and get the result. □

Proposition 10. *Consider dynamics (10.3)-(10.5) with $S(\cdot) = \tanh(\cdot)$. The conditions for equilibria of the perturbed dynamics of (10.3)-(10.5) around $y^\star$ can be reduced to the following single condition:*

$$
\frac{\mathrm{d}}{\mathrm{d}t}(\Delta y_3) = \lambda_1 \Delta y_3 + \lambda_3 (\Delta y_3)^3 + o((\Delta y_3)^3) = 0, \tag{10.12}
$$

where λ_1 and λ_3 depend on $y^\star$, u, N, n as follows:

$$
\begin{aligned}
\lambda_1(y^\star, u, N, n) =& -(N-1) - \Big(1 + \frac{n+1}{n-1}(N-2n)\Big)u \\
& - \frac{n(N-n-1)}{n-1}\frac{2}{c}
\end{aligned} \tag{10.13}
$$

$$
\begin{aligned}
\lambda_3(y^\star, u, N, n) =& \frac{1}{3}\Big(1 + \frac{n+1}{n-1}(N-2n)\Big)u \\
& + \frac{n(N-n-1)}{n-1}\Big(\frac{2}{3c} - \frac{4b^2}{c^5} + \frac{2a}{c^4}\Big)
\end{aligned} \tag{10.14}
$$

with

$$
a(y^\star, N, n) = \frac{n-1}{N-2n}\frac{1}{6}\tanh'''(y^\star) \tag{10.15}
$$

$$
b(y^\star, N, n) = \frac{n-1}{N-2n}\frac{1}{2}\tanh''(y^\star) \tag{10.16}
$$

$$
c(y^\star, u, N, n) = \frac{n-1}{N-2n}\tanh'(y^\star) - \frac{N-n-1}{(N-2n)u}. \tag{10.17}
$$

Proof. First we substitute $\mathbf{y}^\star + \Delta\mathbf{y}$ into the RHS of (10.3), (10.4) and set them equal to zero. With

(10.7), (10.8) and (10.10), we get

$$\begin{aligned}
&-(N-n-1)\Delta y_1 + (n-1)u\Big(S'(y^\star)\Delta y_1 \\
&\quad + \frac{1}{2}S''(y^\star)(\Delta y_1)^2 + \frac{1}{6}S'''(y^\star)(\Delta y_1)^3\Big) \\
&\quad + (N-2n)u(\Delta y_3 - \frac{1}{3}(\Delta y_3)^3) + o((\Delta y_3)^3) = 0 \qquad (10.18)\\
&-(N-n-1)\Delta y_2 + (n-1)u\Big(S'(-y^\star)\Delta y_2 \\
&\quad + \frac{1}{2}S''(-y^\star)(\Delta y_2)^2 + \frac{1}{6}S'''(-y^\star)(\Delta y_2)^3\Big) \\
&\quad + (N-2n)u(\Delta y_3 - \frac{1}{3}(\Delta y_3)^3) + o((\Delta y_3)^3) = 0. \qquad (10.19)
\end{aligned}$$

Eqns. (10.18) and (10.19) can be written as follows:

$$\begin{aligned}
a(\Delta y_1)^3 + b(\Delta y_1)^2 + c\Delta y_1 &= -\Delta y_3 + \frac{1}{3}(\Delta y_3)^3 + o((\Delta y_3)^3)\\
a(\Delta y_2)^3 - b(\Delta y_2)^2 + c\Delta y_2 &= -\Delta y_3 + \frac{1}{3}(\Delta y_3)^3 + o((\Delta y_3)^3)
\end{aligned}$$

with a, b, and c given by (10.15), (10.16), and (10.17), respectively.

By Lemma 8, we have

$$\begin{aligned}
\Delta y_1 = &-\frac{1}{c}\Delta y_3 - \frac{b}{c^3}(\Delta y_3)^2 + \Big(\frac{1}{3c} - \frac{2b^2}{c^5} + \frac{a}{c^4}\Big)(\Delta y_3)^3 \\
&+ o((\Delta y_3)^3) \qquad (10.20)\\
\Delta y_2 = &-\frac{1}{c}\Delta y_3 + \frac{b}{c^3}(\Delta y_3)^2 + \Big(\frac{1}{3c} - \frac{2b^2}{c^5} + \frac{a}{c^4}\Big)(\Delta y_3)^3 \\
&+ o((\Delta y_3)^3). \qquad (10.21)
\end{aligned}$$

We substitute $\mathbf{y}^\star + \Delta\mathbf{y}$ into the RHS of (10.5) and set it equal to zero. We leverage (10.6), (10.18) and (10.19) to get

$$\begin{aligned}
\frac{\mathrm{d}\Delta y_3}{\mathrm{d}t} = &-(N-1)\Delta y_3 + (N-2n-1)u(\Delta y_3 - \frac{1}{3}\Delta y_3^3) \\
&+ \frac{n(N-n-1)}{n-1}(\Delta y_1 + \Delta y_2) + o((\Delta y_3)^3) = 0. \qquad (10.22)
\end{aligned}$$

As we are able to express Δy_1 and Δy_2 in terms of Δy_3 from (10.20) and (10.21), we can substitute them into (10.22). This gives a reduction of the conditions for equilibria of (10.3)-(10.5) from three equations to a single equation in terms of Δy_3. We can see clearly the terms with $(\Delta y_3)^2$ cancel out,

which is consistent with the Z_2-symmetry. We then get our main equation (10.12) with λ_1 and λ_3 given by (10.13) and (10.14), respectively.

□

We examine (10.12) from Proposition 10. If $\lambda_3 < 0$, and λ_1 crosses zero from negative to positive, Δy_3 undergoes a supercritical pitchfork bifurcation. For $\lambda_1 < 0$ and $|\lambda_1|$ sufficiently small, there is a single stable solution at $\Delta y_3 = 0$, which implies $\Delta y_1 = \Delta y_2 = 0$. In this case $\mathbf{y}^\star$ is a stable equilibrium of (10.3)-(10.5) and there are no other solutions nearby. For $\lambda_1 > 0$ and $|\lambda_1|$ sufficiently small, $\Delta y_3 = 0$ is unstable and two stable equilibria $\Delta y_3 = \pm\sqrt{-\lambda_1/\lambda_3}$ appear.

If $\lambda_3 > 0$, and λ_1 crosses zero from negative to positive, Δy_3 undergoes a subcritical pitchfork bifurcation. For $\lambda_1 < 0$ and $|\lambda_1|$ sufficiently small, there are two unstable equilibria $\Delta y_3 = \pm\sqrt{-\lambda_1/\lambda_3}$ and one stable equilibrium $\Delta y_3 = 0$. For $\lambda_1 > 0$ and $|\lambda_1|$ sufficiently small, the three equilibria collapse into one unstable equilibrium $\Delta y_3 = 0$.

The following proposition gives the condition for existence of the transition from supercritical to subcritical pitchfork.

Proposition 11. *The transition from a supercritical pitchfork bifurcation to a subcritical pitchfork bifurcation of dynamics (10.3)-(10.5) with $S(\cdot) = \tanh(\cdot)$ occurs when λ_3 crosses zero from negative to positive. The condition for the transition is*

$$\lambda_3(y^\star, N, n) = 0, \tag{10.23}$$

where

$$\begin{aligned}\lambda_3(y^\star, N, n) = &-\frac{1}{3}(N-1) + \frac{n(N-n-1)}{n-1} \\ &\times \frac{2a(y^\star, N, n)c(y^\star, u(y^\star, N, n), N, n) - 4b^2(y^\star, N, n)}{c^5(y^\star, u(y^\star, N, n), N, n)},\end{aligned} \tag{10.24}$$

and

$$u(y^\star, N, n) = \frac{-c_1 + \sqrt{c_1^2 - 4c_2c_0}}{2c_2} = \frac{-2c_0}{\sqrt{c_1^2 - 4c_2c_0} + c_1}. \tag{10.25}$$

Here, a, b, c are given by (10.15), (10.16), (10.17) and

$$c_2(y^\star, N, n) = \Big(n + 1 + \frac{n-1}{N-2n}\Big)\tanh'(y^\star) \tag{10.26}$$

$$c_1(y^\star, N, n) = \frac{(N-2n-1)(N-n-1)}{(N-2n)} + \frac{(N-1)(n-1)}{N-2n}\tanh'(y^\star) \tag{10.27}$$

$$c_0(y^\star, N, n) = -\frac{(N-n-1)(N-1)}{N-2n}. \tag{10.28}$$

The value of $y^\star$ at the transition is the solution of (10.23). The value of u at the transition is a function of $y^\star$, N, and n (10.25). The value of ϵ at the transition is also a function of $y^\star$, N, and n:

$$\epsilon(y^\star, N, n) = \frac{1}{2}y^\star - \frac{(n-1)}{2(N-n-1)}u(y^\star, N, n)\tanh(y^\star). \tag{10.29}$$

Proof. From previous discussions, the transition from a supercritical bifurcation to a subcritical bifurcation occurs when λ_3 crosses zero from negative to positive. The bifurcation corresponds to $\lambda_1 = 0$. So at the transition, the following equations should be satisfied:

$$\begin{cases} g(y^\star, u, \epsilon, N, n) = 0 & (10.30) \\ \lambda_1(y^\star, u, \epsilon, N, n) = 0 & (10.31) \\ \lambda_3(y^\star, u, \epsilon, N, n) = 0. & (10.32) \end{cases}$$

Here $g(\cdot)$ denotes the LHS of Eqn. (10.6). The dependence of g, λ_1 and λ_3 on variables and parameters is indicated. Thus, given N and n, which specify the network graph structure in the family of networks with three clusters, we can solve for $y^\star$, u and ϵ from Eqn. (10.30)-(10.32).

Eqns. (10.31) and (10.32) do not depend on ϵ explicitly. Eqn. (10.31) can be rearranged as the following quadratic equation:

$$c_2(y^\star, N, n)u^2 + c_1(y^\star, N, n)u + c_0(y^\star, N, n) = 0 \tag{10.33}$$

with c_2, c_1, c_0 given by (10.26), (10.27), (10.28).

Since $\tanh'(y^\star) \in (0, 1]$, we get that $c_2 > 0$, $c_1 > 0$ and $c_0 < 0$. Thus, the quadratic equation (10.33) has one positive and one negative solution. We are only interested in a positive u, so we can write u as a function of $y^\star$, N and n as in (10.25).

As $y^\star$ increases, $\tanh'(y^\star)$ decreases. Then c_1 and c_2 decrease. Thus, the denominator of the

RHS of Eqn. (10.25) decreases. As the numerator is a positive constant, we see that $u(y^\star, N, n)$ is a strictly increasing function of $y^\star$ with $u(0, N, n) = 1$ and $u(+\infty, N, n) = (N-1)/(N-2n-1)$. From Eqn. (10.30), we can express ϵ as a function of $y^\star$, N, and n as given by (10.29).

In Eqns. (10.13) and (10.14), the terms in the big parenthesis in front of u are the same. Thus, setting $\lambda_1 = 0$ in Eqn. (10.31), we can simplify the expression for λ_3 to get

$$\lambda_3 = -\frac{1}{3}(N-1) + \frac{n(N-n-1)}{n-1}\Big(\frac{2a}{c^4} - \frac{4b^2}{c^5}\Big). \tag{10.34}$$

From (10.34), we see that λ_3 depends on N, n, a, b and c. From (10.15)-(10.17) and the fact that we can express u as a function of $y^\star$, we can then express λ_3 as $\lambda_3(y^\star, N, n)$ and get (10.24). □

Our main theorem gives the condition for the existence of a transition from supercritical pitchfork to subcritical pitchfork in dynamics (10.3)-(10.5). The existence only depends on the network structure, i.e., N and n.

Theorem 9. *Given N and n, if there exists a $y_+^\star > 0$ such that $\lambda_3(y_+^\star, N, n) > 0$, then there exists $y_0^\star \in (0, y_+^\star)$ and $y_1^\star \in (y_+^\star, +\infty)$ such that $\lambda_3(y_0^\star, N, n) = \lambda_3(y_1^\star, N, n) = 0$. In particular, the existence of $y_0^\star$ indicates a transition from supercritical pitchfork bifurcation to subcritical pitchfork bifurcation at $\epsilon(y_0^\star)$ and $u(y_0^\star)$. This implies a cascade in the network with three clusters. If there does not exist such a $y_+^\star$, then there is no such transition and thus no cascade.*

Proof. From the proof of Proposition 11, we know that $u(y^\star)$ is a continuous function of $y^\star$ and $u(y^\star) \in [1, (N-1)/(N-2n-1))$. Then from (10.17), c as a function of $y^\star$ does not blow up and is continuous in $y^\star$. Thus, from (10.17), (10.25), we have

$$\begin{aligned} c(y^\star) &= \frac{n-1}{N-2n}\tanh'(y^\star) - \frac{N-n-1}{(N-2n)}\frac{\sqrt{c_1^2 - 4c_2c_0} + c_1}{-2c_0} \\ &\le \frac{n-1}{N-2n}\tanh'(y^\star) - \frac{N-n-1}{(N-2n)}\frac{c_1 + c_1}{-2c_0} \\ &= -\frac{(N-n-1)(N-2n-1)}{(N-2n)(N-1)} < 0. \end{aligned}$$

Thus, from (10.24) it follows that $\lambda_3(y^\star, N, n)$ does not blow up and is continuous in $y^\star$. Moreover, we have

$$\begin{aligned} \lambda_3(0, N, n) &= -\frac{1}{3}(N-1) - \frac{2}{3}\frac{n(N-n-1)}{(N-2n)} < 0 \\ \lambda_3(\infty, N, n) &= -\frac{1}{3}(N-1) < 0. \end{aligned}$$

If there exists a $y_+^\star > 0$ such that $\lambda_3(y_+^\star, N, n) > 0$, then from the continuity of $\lambda_3(y^\star, N, n)$, we know there exists a $y_0^\star \in (0, y_+^\star)$ and $y_1^\star \in (y_+^\star, +\infty)$ such that $\lambda_3(y_0^\star, N, n) = \lambda_3(y_1^\star, N, n) = 0$. Thus, $\lambda_3(y_0^\star, N, n)$ crosses zero from negative to positive, and from Proposition 11, there exists a transition from supercritical pitchfork bifurcation to subcritical pitchfork bifurcation in dynamics (10.3)-(10.5). The value of ϵ and u at which this transition happens can be calculated by $\epsilon(y_0^\star, N, n)$ and $u(y_0^\star, N, n)$ from Eqns. (10.29) and (10.25), respectively. □

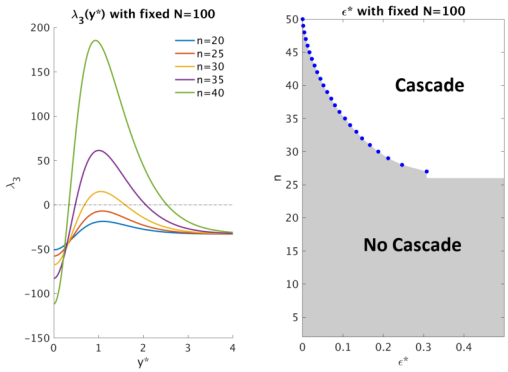

Figure 10.4: The curves of $\lambda_3(y^\star)$ for different values of n and fixed N (left). For lower n, $\lambda_3(y^\star)$ remains negative. For higher n, $\lambda_3(y^\star) = 0$ has two solutions, and thus at the smaller solution $y_0^\star$, there is a transition from supercritical to subcritical pitchfork, and the possibility of a cascade. Critical disparity ϵ^* for different values of n and fixed N (right). For $n \geq 27$, $\epsilon > \epsilon^*$ leads to a cascade.

Remark 3. *Fig. 10.4 illustrates how the existence of a $y_+^\star > 0$, and thus a cascade, depends on network structure parameters N, n, and ϵ. For a fixed N, a large enough n, i.e., a large enough number of agents with disparity in thresholds, is necessary for the cascade. For n large enough that $y_0^\star$ exists, we can expect that for $\epsilon \in [0, \epsilon(y_0^\star, N, n)]$, the bifurcation is supercritical, since it is for $\epsilon = 0$ [32]. As ϵ increases to greater than the critical value $\epsilon^\star = \epsilon(y_0^\star, N, n)$, we expect to see the transition from no cascade to cascade. For $N = 100$, a cascade is possible if $n \geq 27$. The minimum disparity $\epsilon^\star$ that guarantees a cascade decreases as n increases.*

10.5 An example

We present a simulation of the CTM with the network structure shown in Fig. 10.1 and $\epsilon = 0.2$. The initial conditions of the 11 agents are picked randomly. Here, the average initial state is negative. We let $u_0 = 3$, $\kappa = 10$, and $\kappa_s = 0.05$. Then $u = 3\tanh(10|\bar{x}_s|)$, where $\dot{\bar{x}}_s = 0.05(\bar{x} - \bar{x}_s)$, $\bar{x} = \sum_{i=1}^{11} x_i/11$. Fig. 10.5 shows how the states evolve. Agents in clusters 1, 2, and 3 are plotted in red, green, and blue, respectively. A perturbation $\beta = 1$ is added to the dynamics (10.1) of an agent in the red

cluster; its trajectory takes the largest value after the transient period. Except for the perturbed agent, states of all agents in each cluster quickly converge to a common value. So, we can interpret the results in terms of a perturbation of the reduced dynamics (10.3)-(10.5). The solution converges to a perturbation of $\mathbf{y}^\star = [y^\star, -y^\star, 0]^T$. Because of the perturbation, $\bar{x}_s$ slowly increases, which leads to a slow increase in u. At a certain time, u crosses the bifurcation point, which leads to a cascade.

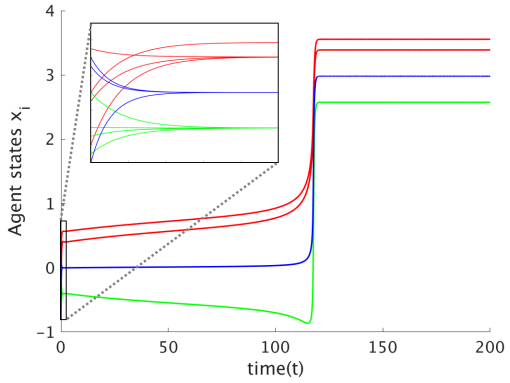

Figure 10.5: Agent state trajectories of the CTM in a network with three clusters, $N = 11$, $n = 4$. There is a cascade corresponding to the unfolded subcritical pitchfork as can be expected since $0.2 = \epsilon > \epsilon(y_0^\star, N, n) = 0.11$.

In this example, the graph structure $N = 11$ and $n = 4$ ensure the existence of $y_0^\star$ such that $\lambda_3(y_0^\star, N, n) = 0$. Thus from Theorem 9, there exists a transition from supercritical pitchfork to subcritical pitchfork in the symmetric system dynamics. Here $\epsilon(y_0^\star, N, n) = 0.11$. With a small ϵ (e.g., 0.1) the system exhibits a supercritical pitchfork; with a large epsilon (e.g., 0.2), the system exhibits a subcritical pitchfork. The introduction of an additive perturbation $\beta = 1$ to the dynamics of a node in the high responsive group breaks the symmetry and lets the subcritical pitchfork unfold as shown in Fig. 10.3 on the right. Thus, as we can see from the simulation, a cascade can be triggered even with a negative initial average state.

Chapter 11

Symplectic ODE-Net: Learning Hamiltonian Dynamics with Control

Yaofeng Desmond Zhong, Biswadip Dey, Amit Chakraborty

Appears as Zhong et al. [101] in *the 8th International Conference in Learning Representations (ICLR 2020)*

In this paper, we introduce Symplectic[1] ODE-Net (SymODEN), a deep learning framework which can infer the dynamics of a physical system, given by an ordinary differential equation (ODE), from observed state trajectories. To achieve better generalization with fewer training samples, SymODEN incorporates appropriate inductive bias by designing the associated computation graph in a physics-informed manner. In particular, we enforce Hamiltonian dynamics with control to learn the underlying dynamics in a transparent way, which can then be leveraged to draw insight about relevant physical aspects of the system, such as mass and potential energy. In addition, we propose a parametrization which can enforce this Hamiltonian formalism even when the generalized coordinate data is embedded in a high-dimensional space or we can only access velocity data instead of generalized momentum. This framework, by offering interpretable, physically-consistent models for physical systems, opens up new possibilities for synthesizing model-based control strategies.

[1]We use the word *Symplectic* to emphasize that the learned dynamics endows a *symplectic* structure [4] on the underlying space.

11.1 Introduction

In recent years, deep neural networks [29] have become very accurate and widely used in many application domains, such as image recognition [38], language comprehension [22], and sequential decision making [82]. To learn underlying patterns from data and enable generalization beyond the training set, the learning approach incorporates appropriate inductive bias [37, 7] by promoting representations which are *simple* in some sense. It typically manifests itself via a set of assumptions, which in turn can guide a learning algorithm to pick one hypothesis over another. The success in predicting an outcome for previously unseen data then depends on how well the inductive bias captures the ground reality. Inductive bias can be introduced as the prior in a Bayesian model, or via the choice of computation graphs in a neural network.

In a variety of settings, especially in physical systems, wherein laws of physics are primarily responsible for shaping the outcome, generalization in neural networks can be improved by leveraging underlying physics for designing the computation graphs. Here, by leveraging a generalization of the Hamiltonian dynamics, we develop a learning framework which exploits the underlying physics in the associated computation graph. Our results show that incorporation of such physics-based inductive bias offers insight about relevant physical properties of the system, such as inertia, potential energy, total conserved energy. These insights, in turn, enable a more accurate prediction of future behavior and improvement in out-of-sample behavior. Furthermore, learning a physically-consistent model of the underlying dynamics can subsequently enable usage of model-based controllers which can provide performance guarantees for complex, nonlinear systems. In particular, insight about kinetic and potential energy of a physical system can be leveraged to synthesize appropriate control strategies, such as the method of controlled Lagrangian [11] and interconnection & damping assignment [66], which can reshape the closed-loop energy landscape to achieve a broad range of control objectives (regulation, tracking, etc.).

Related Work

Physics-based Priors for Learning in Dynamical Systems: The last few years have witnessed a significant interest in incorporating physics-based priors into deep learning frameworks. Such approaches, in contrast to more rigid parametric system identification techniques [84], use neural networks to approximate the state-transition dynamics and therefore are more expressive. [77], by representing the causal relationships in a physical system as a directed graph, use a recurrent

graph network to infer latent space dynamics of robotic systems. [56] and [34] leverage Lagrangian mechanics to learn the dynamics of kinematic structures from time-series data of position, velocity, and acceleration. A more recent (concurrent) work by [33] uses Hamiltonian mechanics to learn the dynamics of autonomous, energy-conserved mechanical systems from time-series data of position, momentum, and their derivatives. A key difference between these approaches and the proposed one is that our framework does not require any information about higher-order derivatives (e.g., acceleration) and can incorporate external control into the Hamiltonian formalism.

Neural Networks for Dynamics and Control: Inferring underlying dynamics from time-series data plays a critical role in controlling closed-loop response of dynamical systems, such as robotic manipulators [54] and building HVAC systems [93]. Although the use of neural networks towards identification and control of dynamical systems dates back to more than three decades ago [63], recent advances in deep neural networks have led to renewed interest in this domain. [89] learn dynamics with control from high-dimensional observations (raw image sequences) using a variational approach and synthesize an iterative LQR controller to control physical systems by imposing a locally linear constraint. [43] and [49] adopt a variational approach and use recurrent architectures to learn state-space models from noisy observation. SE3-Nets [14] learn $SE(3)$ transformation of rigid bodies from point cloud data. [5] use partial information about the system state to learn a nonlinear state-space model. However, this body of work, while attempting to learn state-space models, does not take physics-based priors into consideration.

Contribution

The main contribution of this work is two-fold. First, we introduce a learning framework called Symplectic ODE-Net (SymODEN) which encodes a generalization of the Hamiltonian dynamics. This generalization, by adding an external control term to the standard Hamiltonian dynamics, allows us to learn the system dynamics which conforms to Hamiltonian dynamics with control. With the learned structured dynamics, we are able to synthesize controllers to control the system to track a reference configuration. Moreover, by encoding the structure, we can achieve better predictions with smaller network sizes. Second, we take one step forward in combining the physics-based prior and the data-driven approach. Previous approaches [56, 33] require data in the form of generalized coordinates and their derivatives up to the second order. However, a large number of physical systems accommodate generalized coordinates which are non-Euclidean (e.g., angles), and such angle data is often obtained in the embedded form, i.e., $(\cos q, \sin q)$ instead of the coordinate

(q) itself. The underlying reason is that an angular coordinate lies on $\mathbb{S}^1$ instead of $\mathbb{R}^1$. In contrast to previous approaches which do not address this aspect, SymODEN has been designed to work with angle data in the embedded form. Additionally, we leverage differentiable ODE solvers to avoid the need for estimating second-order derivatives of generalized coordinates. Code for the SymODEN framework and experiments is available at https://github.com/d-biswa/Symplectic-ODENet.

11.2 Preliminary Concepts

11.2.1 Hamiltonian Dynamics

Lagrangian dynamics and Hamiltonian dynamics are both reformulations of Newtonian dynamics. They provide novel insights into the laws of mechanics. In these formulations, the configuration of a system is described by its generalized coordinates. Over time, the configuration point of the system moves in the configuration space, tracing out a trajectory. Lagrangian dynamics describes the evolution of this trajectory, i.e., the equations of motion, in the configuration space. Hamiltonian dynamics, however, tracks the change of system states in the phase space, i.e. the product space of generalized coordinates $\mathbf{q} = (q_1, q_2, ..., q_n)$ and generalized momenta $\mathbf{p} = (p_1, p_2, ..., p_n)$. In other words, Hamiltonian dynamics treats $\mathbf{q}$ and $\mathbf{p}$ on an equal footing. This not only provides symmetric equations of motion but also leads to a whole new approach to classical mechanics [27]. Hamiltonian dynamics is also widely used in statistical and quantum mechanics.

In Hamiltonian dynamics, the time-evolution of a system is described by the Hamiltonian $H(\mathbf{q}, \mathbf{p})$, a scalar function of generalized coordinates and momenta. Moreover, in almost all physical systems, the Hamiltonian is the same as the total energy and hence can be expressed as

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}(\mathbf{q})\mathbf{p} + V(\mathbf{q}), \tag{11.1}$$

where the mass matrix $\mathbf{M}(\mathbf{q})$ is symmetric positive definite and $V(\mathbf{q})$ represents the potential energy of the system. Correspondingly, the time-evolution of the system is governed by

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \qquad \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}}, \tag{11.2}$$

where we have dropped explicit dependence on $\mathbf{q}$ and $\mathbf{p}$ for brevity of notation. Moreover, since

$$\dot{H} = \left(\frac{\partial H}{\partial \mathbf{q}}\right)^T \dot{\mathbf{q}} + \left(\frac{\partial H}{\partial \mathbf{p}}\right)^T \dot{\mathbf{p}} = 0, \tag{11.3}$$

the total energy is conserved along a trajectory of the system. The RHS of Equation (11.2) is called the symplectic gradient [73] of H, and Equation (11.3) shows that moving along the symplectic gradient keeps the Hamiltonian constant.

In this work, we consider a generalization of the Hamiltonian dynamics which provides a means to incorporate external control ($\mathbf{u}$), such as force and torque. As external control is usually affine and only influences changes in the generalized momenta, we can express this generalization as

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \frac{\partial H}{\partial \mathbf{p}} \\ -\frac{\partial H}{\partial \mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{g}(\mathbf{q}) \end{bmatrix} \mathbf{u}, \tag{11.4}$$

where the input matrix $\mathbf{g}(\mathbf{q})$ is typically assumed to have full column rank. For $\mathbf{u} = \mathbf{0}$, the generalized dynamics reduces to the classical Hamiltonian dynamics (11.2) and the total energy is conserved; however, when $\mathbf{u} \neq \mathbf{0}$, the system has a dissipation-free energy exchange with the environment.

11.2.2 Control via Energy Shaping

Once we have learned the dynamics of a system, the learned model can be used to synthesize a controller for driving the system to a reference configuration $\mathbf{q}^\star$. As the proposed approach offers insight about the energy associated with a system, it is a natural choice to exploit this information for synthesizing controllers via *energy shaping* [65]. As energy is a fundamental aspect of physical systems, reshaping the associated energy landscape enables us to specify a broad range of control objectives and synthesize nonlinear controllers with provable performance guarantees.

If $\text{rank}(\mathbf{g}(\mathbf{q})) = \text{rank}(\mathbf{q})$, the system is fully-actuated and we have control over any dimension of "acceleration" in $\dot{\mathbf{p}}$. For such fully-actuated systems, a controller $\mathbf{u}(\mathbf{q}, \mathbf{p}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\mathbf{p})$ can be synthesized via *potential energy shaping* $\boldsymbol{\beta}(\mathbf{q})$ and *damping injection* $\mathbf{v}(\mathbf{p})$. For completeness, we restate this procedure [65] using our notation. As the name suggests, the goal of potential energy shaping is to synthesize $\boldsymbol{\beta}(\mathbf{q})$ such that the closed-loop system behaves as if its time-evolution is governed by a desired Hamiltonian H_d. With this, we have

$$\begin{bmatrix} \dot{\mathbf{q}} \\ \dot{\mathbf{p}} \end{bmatrix} = \begin{bmatrix} \frac{\partial H}{\partial \mathbf{p}} \\ -\frac{\partial H}{\partial \mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{g}(\mathbf{q}) \end{bmatrix} \boldsymbol{\beta}(\mathbf{q}) = \begin{bmatrix} \frac{\partial H_d}{\partial \mathbf{p}} \\ -\frac{\partial H_d}{\partial \mathbf{q}} \end{bmatrix}, \tag{11.5}$$

where the difference between the desired Hamiltonian and the original one lies in their potential energy term, i.e.

$$H_d(\mathbf{q}, \mathbf{p}) = \frac{1}{2} \mathbf{p}^T \mathbf{M}^{-1}(\mathbf{q}) \mathbf{p} + V_d(\mathbf{q}). \tag{11.6}$$

In other words, $\boldsymbol{\beta}(\mathbf{q})$ *shape* the potential energy such that the desired Hamiltonian $H_d(\mathbf{q}, \mathbf{p})$ has a minimum at $(\mathbf{q}^\star, \mathbf{0})$. Then, by substituting Equation (11.1) and Equation (11.6) into Equation (11.5), we get

$$\boldsymbol{\beta}(\mathbf{q}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \frac{\partial V_d}{\partial \mathbf{q}}\Big). \tag{11.7}$$

Thus, with potential energy shaping, we ensure that the system has the lowest energy at the desired reference configuration. Furthermore, to ensure that trajectories actually converge to this configuration, we add an additional damping term[2] given by

$$\mathbf{v}(\mathbf{p}) = -\mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}(\mathbf{K}_d\mathbf{p}). \tag{11.8}$$

However, for underactuated systems, potential energy shaping alone cannot[3] drive the system to a desired configuration. We also need kinetic energy shaping for this purpose [16].

Remark If the desired potential energy is chosen to be a quadratic of the form

$$V_d(\mathbf{q}) = \frac{1}{2}(\mathbf{q} - \mathbf{q}^\star)^T\mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star), \tag{11.9}$$

the external forcing term can be expressed as

$$\mathbf{u} = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\left(\frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star) - \mathbf{K}_d\mathbf{p}\right). \tag{11.10}$$

This can be interpreted as a PD controller with an additional energy compensation term.We [4]

11.3 Symplectic ODE-Net

In this section, we introduce the network architecture of Symplectic ODE-Net. In Subsection 11.3.1, we show how to learn an ordinary differential equation with a constant control term. In Subsection 11.3.2, we assume we have access to generalized coordinate and momentum data and derive the network architecture. In Subsection 11.3.3, we take one step further to propose a data-driven approach to deal with data of embedded angle coordinates. In Subsection 11.3.4, we put together the line of reasoning introduced in the previous two subsections to propose SymODEN for learning dynamics on the hybrid space $\mathbb{R}^n \times \mathbb{T}^m$.

[2] If we have access to $\mathbf{q}$ instead of $\mathbf{p}$, we use $\mathbf{q}$ instead in Equation (11.8).
[3] As gg^T is not invertible, we cannot solve the matching condition given by Equation (11.7).
[4] Please refer to Appendix 11.6.2 for more details.

11.3.1 Training Neural ODE with Constant Forcing

Now we focus on the problem of learning the ordinary differential equation (ODE) from time series data. Consider an ODE: $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x})$. Assume we don't know the analytical expression of the right hand side (RHS) and we approximate it with a neural network. If we have time series data $\mathbf{X} = (\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, ..., \mathbf{x}_{t_n})$, how could we learn $\mathbf{f}(\mathbf{x})$ from the data?

[17] introduced Neural ODE, differentiable ODE solvers with O(1)-memory backpropagation. With Neural ODE, we make predictions by approximating the RHS function using a neural network $\mathbf{f}_\theta$ and feed it into an ODE solver

$$\mathbf{x}_{t_1}, \mathbf{x}_{t_2}, ..., \mathbf{x}_{t_n} = \text{ODESolve}(\mathbf{x}_{t_0}, \mathbf{f}_\theta, t_1, t_2, ..., t_n)$$

We can then construct the loss function $L = \|\mathbf{X} - \mathbf{X}\|_2^2$ and update the weights θ by backpropagating through the ODE solver.

In theory, we can learn $\mathbf{f}_\theta$ in this way. In practice, however, the neural net is hard to train if n is large. If we have a bad initial estimate of the $\mathbf{f}_\theta$, the prediction error would in general be large. Although $|\mathbf{x}_{t_1} - \mathbf{x}_{t_1}|$ might be small, $\mathbf{x}_{t_N}$ would be far from $\mathbf{x}_{t_N}$ as error accumulates, which makes the neural network hard to train. In fact, the prediction error of $\mathbf{x}_{t_N}$ is not as important as $\mathbf{x}_{t_1}$. In other words, we should weight data points in a short time horizon more than the rest of the data points. In order to address this and better utilize the data, we introduce the time horizon τ as a hyperparameter and predict $\mathbf{x}_{t_{i+1}}, \mathbf{x}_{t_{i+2}}, ..., \mathbf{x}_{t_{i+\tau}}$ from initial condition $\mathbf{x}_{t_i}$, where $i = 0, ..., n - \tau$.

One challenge toward leveraging Neural ODE to learn state-space models is the incorporation of the control term into the dynamics. Equation (11.4) has the form $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$ with $\mathbf{x} = (\mathbf{q}, \mathbf{p})$. A function of this form cannot be directly fed into Neural ODE directly since the domain and range of $\mathbf{f}$ have different dimensions. In general, if our data consist of trajectories of $(\mathbf{x}, \mathbf{u})_{t_0,...,t_n}$ where $\mathbf{u}$ remains the same in a trajectory, we can leverage the augmented dynamics

$$\begin{bmatrix} \dot{\mathbf{x}} \\ \dot{\mathbf{u}} \end{bmatrix} = \begin{bmatrix} \mathbf{f}_\theta(\mathbf{x}, \mathbf{u}) \\ \mathbf{0} \end{bmatrix} = \tilde{\mathbf{f}}_\theta(\mathbf{x}, \mathbf{u}). \tag{11.11}$$

With Equation (11.11), we can match the input and output dimension of $\tilde{\mathbf{f}}_\theta$, which enables us to feed it into Neural ODE. The idea here is to use different constant external forcing to get the system responses and use those responses to train the model. With a trained model, we can apply a time-varying $\mathbf{u}$ to the dynamics $\dot{\mathbf{x}} = \mathbf{f}_\theta(\mathbf{x}, \mathbf{u})$ and generate estimated trajectories. When we synthesize

the controller, $\mathbf{u}$ remains constant in each integration step. As long as our model interpolates well among different values of constant $\mathbf{u}$, we could get good estimated trajectories with a time-varying $\mathbf{u}$. The problem is then how to design the network architecture of $\tilde{\mathbf{f}}_\theta$, or equivalently $\mathbf{f}_\theta$ such that we can learn the dynamics in an efficient way.

11.3.2 Learning from Generalized Coordinate and Momentum

Suppose we have trajectory data consisting of $(\mathbf{q}, \mathbf{p}, \mathbf{u})_{t_0,\dots,t_n}$, where $\mathbf{u}$ remains constant in a trajectory. If we have the prior knowledge that the unforced dynamics of $\mathbf{q}$ and $\mathbf{p}$ is governed by Hamiltonian dynamics, we can use three neural nets – $\mathbf{M}^{-1}_{\theta_1}(\mathbf{q})$, $V_{\theta_2}(\mathbf{q})$ and $\mathbf{g}_{\theta_3}(\mathbf{q})$ – as function approximators to represent the inverse of mass matrix, potential energy and the input matrix. Thus,

$$\mathbf{f}_\theta(\mathbf{q}, \mathbf{p}, \mathbf{u}) = \begin{bmatrix} \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \\ -\frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{0} \\ \mathbf{g}_{\theta_3}(\mathbf{q}) \end{bmatrix} \mathbf{u} \tag{11.12}$$

where

$$H_{\theta_1,\theta_2}(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \mathbf{M}^{-1}_{\theta_1}(\mathbf{q})\mathbf{p} + V_{\theta_2}(\mathbf{q}) \tag{11.13}$$

The partial derivative in the expression can be taken care of by automatic differentiation. by putting the designed $\mathbf{f}_\theta(\mathbf{q}, \mathbf{p}, \mathbf{u})$ into Neural ODE, we obtain a systematic way of adding the prior knowledge of Hamiltonian dynamics into end-to-end learning.

11.3.3 Learning from Embedded Angle Data

In the previous subsection, we assume $(\mathbf{q}, \mathbf{p}, \mathbf{u})_{t_0,\dots,t_n}$. In a lot of physical system models, the state variables involve angles which reside in the interval $[-\pi, \pi)$. In other words, each angle resides on the manifold $\mathbb{S}^1$. From a data-driven perspective, the data that respects the geometry is a 2 dimensional embedding $(\cos q, \sin q)$. Furthermore, the generalized momentum data is usually not available. Instead, the velocity is often available. For example, in OpenAI Gym [13] `Pendulum-v0` task, the observation is $(\cos q, \sin q, \dot{q})$.

From a theoretical perspective, however, the angle itself is often used, instead of the 2D embedding. The reason being both the Lagrangian and the Hamiltonian formulations are derived using generalized coordinates. Using an independent generalized coordinate system makes it easier to solve for the equations of motion.

In this subsection, we take the data-driven standpoint and develop an angle-aware method to

accommodate the underlying manifold structure. We assume all the generalized coordinates are angles and the data comes in the form of $(\mathbf{x}_1(\mathbf{q}), \mathbf{x}_2(\mathbf{q}), \mathbf{x}_3(\dot{\mathbf{q}}), \mathbf{u})_{t_0,\dots,t_n} = (\cos\mathbf{q}, \sin\mathbf{q}, \dot{\mathbf{q}}, \mathbf{u})_{t_0,\dots,t_n}$. We aim to incorporate our theoretical prior – Hamiltonian dynamics – into the data-driven approach. The goal is to learn the dynamics of $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. Noticing $\mathbf{p} = \mathbf{M}(\mathbf{x}_1, \mathbf{x}_2)\dot{\mathbf{q}}$, we can write down the derivative of $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$,

$$\begin{aligned}
\dot{\mathbf{x}}_1 &= -\sin\mathbf{q} \circ \dot{\mathbf{q}} = -\mathbf{x}_2 \circ \dot{\mathbf{q}} \\
\dot{\mathbf{x}}_2 &= \cos\mathbf{q} \circ \dot{\mathbf{q}} = \mathbf{x}_1 \circ \dot{\mathbf{q}} \\
\dot{\mathbf{x}}_3 &= \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{p}) = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}(\mathbf{x}_1, \mathbf{x}_2))\mathbf{p} + \mathbf{M}^{-1}(\mathbf{x}_1, \mathbf{x}_2)\dot{\mathbf{p}}
\end{aligned} \tag{11.14}$$

where "$\circ$" represents the elementwise product (i.e., Hadamard product). We assume $\mathbf{q}$ and $\mathbf{p}$ evolve with the generalized Hamiltonian dynamics Equation (11.4). Here the Hamiltonian $H(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p})$ is a function of $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{p}$ instead of $\mathbf{q}$ and $\mathbf{p}$.

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \tag{11.15}$$

$$\begin{aligned}
\dot{\mathbf{p}} &= -\frac{\partial H}{\partial \mathbf{q}} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u} = -\frac{\partial \mathbf{x}_1}{\partial \mathbf{q}}\frac{\partial H}{\partial \mathbf{x}_1} - \frac{\partial \mathbf{x}_2}{\partial \mathbf{q}}\frac{\partial H}{\partial \mathbf{x}_2} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u} \\
&= \sin\mathbf{q} \circ \frac{\partial H}{\partial \mathbf{x}_1} - \cos\mathbf{q} \circ \frac{\partial H}{\partial \mathbf{x}_2} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u} = \mathbf{x}_2 \circ \frac{\partial H}{\partial \mathbf{x}_1} - \mathbf{x}_1 \circ \frac{\partial H}{\partial \mathbf{x}_2} + \mathbf{g}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{u}
\end{aligned} \tag{11.16}$$

Then the right hand side of Equation (11.14) can be expressed as a function of state variables and control $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{u})$. Thus, it can be fed into the Neural ODE. We use three neural nets – $\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2)$, $V_{\theta_2}(\mathbf{x}_1, \mathbf{x}_2)$ and $\mathbf{g}_{\theta_3}(\mathbf{x}_1, \mathbf{x}_2)$ – as function approximators. Substitute Equation (11.15) and Equation (11.16) into Equation (11.14), then the RHS serves as $\mathbf{f}_\theta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{u})$.[5]

$$\mathbf{f}_\theta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{u}) = \begin{bmatrix} -\mathbf{x}_2 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \\ \mathbf{x}_1 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \\ \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1,\mathbf{x}_2))\mathbf{p} + \mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1,\mathbf{x}_2)\left(\mathbf{x}_2 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{x}_1} - \mathbf{x}_1 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{x}_2} + \mathbf{g}_{\theta_3}(\mathbf{x}_1,\mathbf{x}_2)\mathbf{u}\right) \end{bmatrix} \tag{11.17}$$

where

$$H_{\theta_1,\theta_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{p} + V_{\theta_2}(\mathbf{x}_1, \mathbf{x}_2) \tag{11.18}$$

$$\mathbf{p} = \mathbf{M}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2)\mathbf{x}_3 \tag{11.19}$$

[5] In Equation (11.17), the derivative of $\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2)$ can be expanded using chain rule and expressed as a function of the states.

11.3.4 Learning on Hybrid Spaces $\mathbb{R}^n \times \mathbb{T}^m$

In Subsection 11.3.2, we treated the generalized coordinates as translational coordinates. In Subsection 11.3.3, we developed an angle-aware method to better deal with embedded angle data. In most of physical systems, these two types of coordinates coexist. For example, robotics systems are usually modelled as interconnected rigid bodies. The positions of joints or center of mass are translational coordinates and the orientations of each rigid body are angular coordinates. In other words, the generalized coordinates lie on $\mathbb{R}^n \times \mathbb{T}^m$, where $\mathbb{T}^m$ denotes the m-torus, with $\mathbb{T}^1 = \mathbb{S}^1$ and $\mathbb{T}^2 = \mathbb{S}^1 \times \mathbb{S}^1$. In this subsection, we put together the architecture of the previous two subsections. We assume the generalized coordinates are $\mathbf{q} = (\mathbf{r}, \boldsymbol{\phi}) \in \mathbb{R}^n \times \mathbb{T}^m$ and the data comes in the form of $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{u})_{t_0,\dots,t_n} = (\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathbf{r}}, \dot{\boldsymbol{\phi}}, \mathbf{u})_{t_0,\dots,t_n}$. With similar line of reasoning, we use three neural nets – $\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, $V_{\theta_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and $\mathbf{g}_{\theta_3}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ – as function approximators. We have

$$\mathbf{p} = \mathbf{M}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2, , \mathbf{x}_3) \begin{bmatrix} \mathbf{x}_4 \\ \mathbf{x}_5 \end{bmatrix} \tag{11.20}$$

$$H_{\theta_1,\theta_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T \mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)\mathbf{p} + V_{\theta_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \tag{11.21}$$

with Hamiltonian dynamics, we have

$$\dot{\mathbf{q}} = \begin{bmatrix} \dot{\mathbf{r}} \\ \dot{\boldsymbol{\phi}} \end{bmatrix} = \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}} \tag{11.22}$$

$$\dot{\mathbf{p}} = \begin{bmatrix} -\frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{x}_1} \\ \mathbf{x}_3 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{x}_2} - \mathbf{x}_2 \circ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{x}_3} \end{bmatrix} + \mathbf{g}_{\theta_3}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)\mathbf{u} \tag{11.23}$$

Then

$$\begin{bmatrix} \dot{\mathbf{x}}_1 \\ \dot{\mathbf{x}}_2 \\ \dot{\mathbf{x}}_3 \\ \dot{\mathbf{x}}_4 \\ \dot{\mathbf{x}}_5 \end{bmatrix} = \begin{bmatrix} \dot{\mathbf{r}} \\ -\mathbf{x}_3\dot{\boldsymbol{\phi}} \\ \mathbf{x}_2\dot{\boldsymbol{\phi}} \\ \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3))\mathbf{p} + \mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)\dot{\mathbf{p}} \end{bmatrix} = \mathbf{f}_\theta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{u}) \tag{11.24}$$

where the $\dot{\mathbf{r}}$ and $\dot{\boldsymbol{\phi}}$ come from Equation (11.22). Now we obtain a $\mathbf{f}_\theta$ which can be fed into Neural ODE. Figure 11.1 shows the flow of the computation graph based on Equation (11.20)-(11.24).

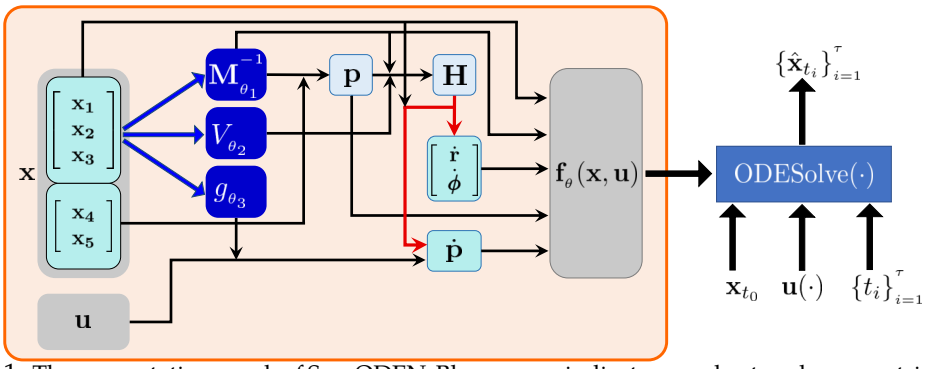

Figure 11.1: The computation graph of SymODEN. Blue arrows indicate neural network parametrization. Red arrows indicate automatic differentiation. For a given $(\mathbf{x}, \mathbf{u})$, the computation graph outputs a $\mathbf{f}_\theta(\mathbf{x}, \mathbf{u})$ which follows Hamiltonian dynamics with control. The function itself is an input to the Neural ODE to generate estimation of states at each time step. Since all the operations are differentiable, weights of the neural networks can be updated by backpropagation.

11.3.5 Positive Definiteness of the Mass matrix

In real physical systems, the mass matrix $\mathbf{M}$ is positive definite, which ensures a positive kinetic energy with a non-zero velocity. The positive definiteness of $\mathbf{M}$ implies the positive definiteness of $\mathbf{M}_{\theta_1}^{-1}$. Thus, we impose this constraint in the network architecture by $\mathbf{M}_{\theta_1}^{-1} = \mathbf{L}_{\theta_1}\mathbf{L}_{\theta_1}^T$, where $\mathbf{L}_{\theta_1}$ is a lower-triangular matrix. The positive definiteness is ensured if the diagonal elements of $\mathbf{M}_{\theta_1}^{-1}$ are positive. In practice, this can be done by adding a small constant ϵ to the diagonal elements of $\mathbf{M}_{\theta_1}^{-1}$. It not only makes $\mathbf{M}_{\theta_1}$ invertible, but also stabilizes the training.

11.4 Experiments

11.4.1 Experimental Setup

We use the following four tasks to evaluate the performance of Symplectic ODE-Net model - (i) Task 1: a pendulum with generalized coordinate and momentum data (learning on $\mathbb{R}^1$); (ii) Task 2: a pendulum with embedded angle data (learning on $\mathbb{S}^1$); (iii) Task 3: a CartPole system (learning on $\mathbb{R}^1 \times \mathbb{S}^1$); and (iv) Task 4: an Acrobot (learning on $\mathbb{T}^2$).

Model Variants. Besides the Symplectic ODE-Net model derived above, we consider a variant by approximating the Hamiltonian using a fully connected neural net H_{θ_1,θ_2}. We call it *Unstructured Symplectic ODE-Net (Unstructured SymODEN)* since this model does not exploit the structure of the Hamiltonian (11.1).

Baseline Models. In order to show that we can learn the dynamics better with less parameters by leveraging prior knowledge, we set up baseline models for all four experiments. For the pendulum with generalized coordinate and momentum data, the *naive baseline* model approximates

Equation (11.12) – $\mathbf{f}_\theta(\mathbf{x}, \mathbf{u})$ – by a fully connected neural net. For all the other experiments, which involves embedded angle data, we set up two different baseline models: *naive baseline* approximates $\mathbf{f}_\theta(\mathbf{x}, \mathbf{u})$ by a fully connected neural net. It doesn't respect the fact that the coordinate pair, $\cos\boldsymbol{\phi}$ and $\sin\boldsymbol{\phi}$, lie on $\mathbb{T}^m$. Thus, we set up the *geometric baseline* model which approximates $\dot{q}$ and $\dot{p}$ with a fully connected neural net. This ensures that the angle data evolves on $\mathbb{T}^m$. [6]

Data Generation. For all tasks, we randomly generated initial conditions of states and subsequently combined them with 5 different constant control inputs, i.e., $u = -2.0, -1.0, 0.0, 1.0, 2.0$ to produce the initial conditions and input required for simulation. The simulators integrate the corresponding dynamics for 20 time steps to generate trajectory data which is then used to construct the training set. The simulators for different tasks are different. For Task 1, we integrate the true generalized Hamiltonian dynamics with a time interval of 0.05 seconds to generate trajectories. All the other tasks deal with embedded angle data and velocity directly, so we use OpenAI Gym [13] simulators to generate trajectory data. One drawback of using OpenAI Gym is that not all environments use the Runge-Kutta method (RK4) to carry out the integration. OpenAI Gym favors other numerical schemes over RK4 because of speed, but it is harder to learn the dynamics with inaccurate data. For example, if we plot the total energy as a function of time from data generated by `Pendulum-v0` environment with zero action, we see that the total energy oscillates around a constant by a significant amount, even though the total energy should be conserved. Thus, for Task 2 and Task 3, we use `Pendulum-v0` and `CartPole-v1`, respectively, and replace the numerical integrator of the environments to RK4. For Task 4, we use the `Acrobot-v1` environment which is already using RK4. We also change the action space of `Pendulum-v0`, `CartPole-v1` and `Acrobot-v1` to a continuous space with a large enough bound.

Model training. In all the tasks, we train our model using Adam optimizer [46] with 1000 epochs. We set a time horizon $\tau = 3$, and choose "RK4" as the numerical integration scheme in Neural ODE. We vary the size of the training set by doubling from 16 initial state conditions to 1024 initial state conditions. Each initial state condition is combined with five constant control $u = -2.0, -1.0, 0.0, 1.0, 2.0$ to produce initial condition for simulation. Each trajectory is generated by integrating the dynamics 20 time steps forward. We set the size of mini-batches to be the number of initial state conditions. We logged the *train error* per trajectory and the *prediction error* per trajectory in each case for all the tasks. The train error per trajectory is the mean squared error (MSE) between the estimated trajectory and the ground truth over 20 time steps. To evaluate the performance of each model in terms of long time prediction, we construct the metric of prediction

[6] For more information on model details, please refer to Appendix 11.6.1.

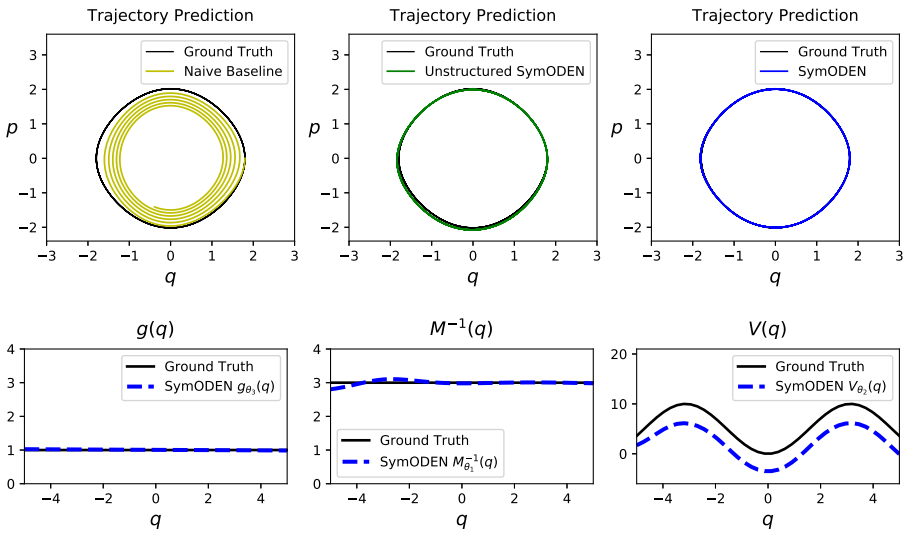

Figure 11.2: Sample trajectories and learned functions of Task 1.

error per trajectory by using the same initial state condition in the training set with a constant control of $u = 0.0$, integrating 40 time steps forward, and calculating the MSE over 40 time steps. The reason for using only the unforced trajectories is that a constant nonzero control might cause the velocity to keep increasing or decreasing over time, and large absolute values of velocity are of little interest for synthesizing controllers.

11.4.2 Task 1: Pendulum with Generalized Coordinate and Momentum Data

In this task, we use the model described in Section 11.3.2 and present the predicted trajectories of the learned models as well as the learned functions of SymODEN. We also point out the drawback of treating the angle data as a Cartesian coordinate. The dynamics of this task has the following form

$$\dot{q} = 3p, \qquad \dot{p} = -5\sin q + u \tag{11.25}$$

with Hamiltonian $H(q, p) = 1.5p^2 + 5(1 - \cos q)$. In other words $M(q) = 3$, $V(q) = 5(1 - \cos q)$ and $g(q) = 1$.

In Figure 11.2, The ground truth is an unforced trajectory which is energy-conserved. The prediction trajectory of the baseline model does not conserve energy, while both the SymODEN and its unstructured variant predict energy-conserved trajectories. For SymODEN, the learned $g_{\theta_3}(q)$ and $M^{-1}_{\theta_1}(q)$ matches the ground truth well. $V_{\theta_2}(q)$ differs from the ground truth with a constant.

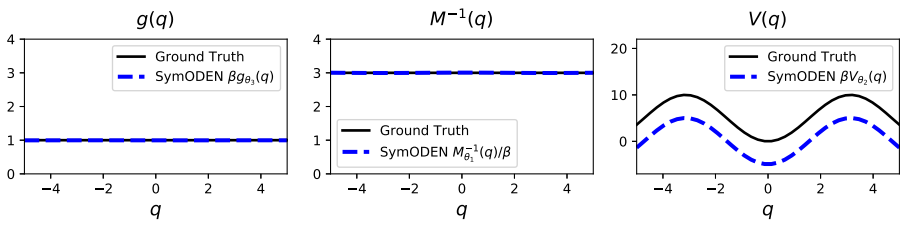

Figure 11.3: Without true generalized momentum data, the learned functions match the ground truth with a scaling. Here $\beta = 0.357$

This is acceptable since the potential energy is a relative notion. Only the derivative of $V_{\theta_2}(q)$ plays a role in the dynamics.

Here we treat q as a variable in $\mathbb{R}^1$ and our training set contains initial conditions of $q \in [-\pi, 3\pi]$. The learned functions do not extrapolate well outside this range, as we can see from the left part in the figures of $M^{-1}_{\theta_1}(q)$ and $V_{\theta_2}(q)$. We address this issue by working directly with embedded angle data, which leads us to the next subsection.

11.4.3 Task 2: Pendulum with Embedded Data

In this task, the dynamics is the same as Equation (11.25) but the training data are generated by the OpenAI Gym simulator, i.e. we use embedded angle data and assume we only have access to $\dot{q}$ instead of p. We use the model described in Section 11.3.3 and synthesize an energy-based controller (Section 11.2.2). Without true p data, the learned function matches the ground truth with a scaling β, as shown in Figure 11.3. To explain the scaling, let us look at the following dynamics

$$\dot{q} = p/\alpha, \qquad \dot{p} = -15\alpha \sin q + 3\alpha u \tag{11.26}$$

with Hamiltonian $H = p^2/(2\alpha) + 15\alpha(1 - \cos q)$. If we only look at the dynamics of q, we have $\ddot{q} = -15 \sin q + 3u$, which is independent of α. If we don't have access to the generalized momentum p, our trained neural network may converge to a Hamiltonian with a α_e which is different from the true value, $\alpha_t = 1/3$, in this task. By a scaling $\beta = \alpha_t/\alpha_e = 0.357$, the learned functions match the ground truth. Even we are not learning the true α_t, we can still perform prediction and control since we are learning the dynamics of q correctly. We let $V_d = -V_{\theta_2}(q)$, then the desired Hamiltonian has minimum energy when the pendulum rests at the upward position. For the damping injection, we

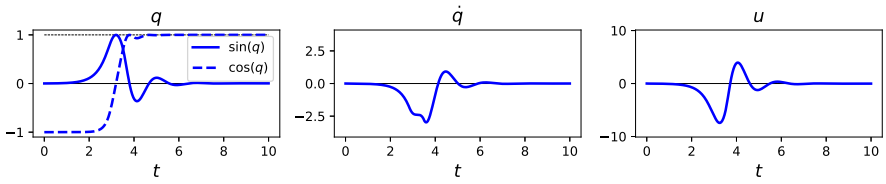

Figure 11.4: Time-evolution of the state variables $(\cos q, \sin q, \dot{q})$ when the closed-loop control input $u(\cos q, \sin q, \dot{q})$ is governed by Equation (11.27). The thin black lines show the expected results.

let $K_d = 3$. Then from Equation (11.7) and (11.8), the controller we synthesize is

$$u(\cos q, \sin q, \dot{q}) = g_{\theta_3}^{-1}(\cos q, \sin q)\Big(2\big(-\frac{\partial V_{\theta_2}}{\partial \cos q}\sin q + \frac{\partial V_{\theta_2}}{\partial \sin q}\cos q\big) - 3\dot{q}\Big) \tag{11.27}$$

Only SymODEN out of all models we consider provides the learned potential energy which is required to synthesize the controller. Figure 11.4 shows how the states evolve when the controller is fed into the OpenAI Gym simulator. We can successfully control the pendulum into the inverted position using the controller based on the learned model even though the absolute maximum control u, 7.5, is more than three times larger than the absolute maximum u in the training set, which is 2.0. This shows SymODEN extrapolates well.

11.4.4 Task 3: CartPole System

The CartPole system is an underactuated system and to synthesize a controller to balance the pole from arbitrary initial condition requires trajectory optimization or kinetic energy shaping. We show that we can learn its dynamics and perform prediction in Section 11.4.6. We also train SymODEN in a fully-actuated version of the CartPole system (see Section 11.6.5). The corresponding energy-based controller can bring the pole to the inverted position while driving the cart to the origin.

11.4.5 Task 4: Acrobot

The Acrobot is an underactuated double pendulum. As this system exhibits chaotic motion, it is not possible to predict its long-term behavior. However, Figure 11.6 shows that SymODEN can provide reasonably good short-term prediction. We also train SymODEN in a fully-actuated version of the Acrobot and show that we can control this system to reach the inverted position (see Appendix 11.6.5).

11.4.6 Results

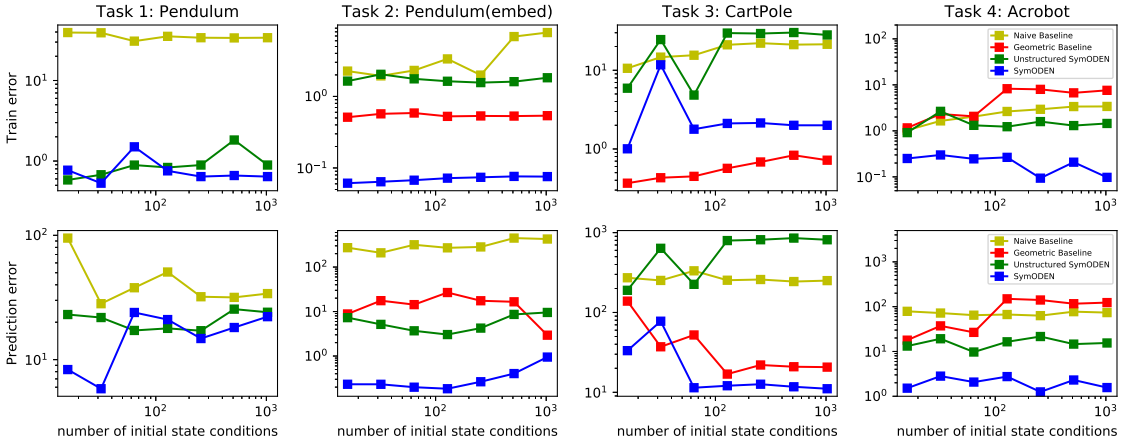

Figure 11.5: Train error per trajectory and prediction error per trajectory for all 4 tasks with different number of training trajectories. Horizontal axis shows number of initial state conditions (16, 32, 64, 128, 256, 512, 1024) in the training set. Both the horizontal axis and vertical axis are in log scale.

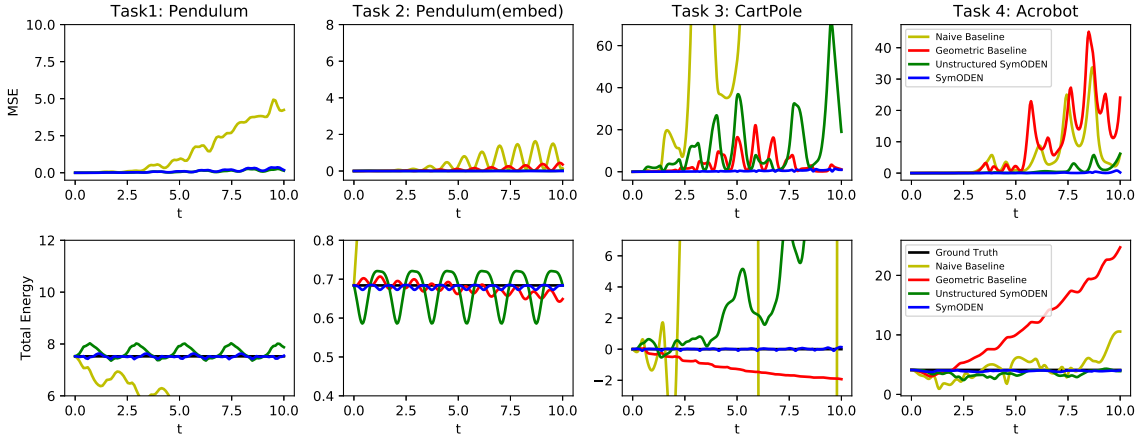

Figure 11.6: Mean square error and total energy of test trajectories. SymODEN works the best in terms of both MSE and total energy. Since SymODEN has learned the Hamiltonian and discovered the conservation from data the predicted trajectories match the ground truth. The ground truth of energy in all four tasks stay constant.

In this subsection, we show the train error, prediction error, as well as the MSE and total energy of a sample test trajectory for all the tasks. Figure 11.5 shows the variation in train error and prediction error with changes in the number of initial state conditions in the training set. We can see that SymODEN yields better generalization in every task. In Task 3, although the Geometric Baseline Model yields lower train error in comparison to the other models, SymODEN generates more accurate predictions, indicating overfitting in the Geometric Baseline Model. By incorporating the physics-based prior of Hamiltonian dynamics, SymODEN learns dynamics that obeys physical laws and thus provides better predictions. In most cases, SymODEN trained with a smaller training dataset performs better than other models in terms of the train and prediction error, indicating that

better generalization can be achieved even with fewer training samples.

Figure 11.6 shows the evolution of MSE and total energy along a trajectory with a previously unseen initial condition. For all the tasks, MSE of the baseline models diverges faster than SymODEN. Unstructured SymODEN performs well in all tasks except Task 3. As for the total energy, in Task 1 and Task 2, SymODEN and Unstructured SymODEN conserve total energy by oscillating around a constant value. In these models, the Hamiltonian itself is learned and the prediction of the future states stay around a level set of the Hamiltonian. Baseline models, however, fail to find the conservation and the estimation of future states drift away from the initial Hamiltonian level set.

11.5 Conclusion

Here we have introduced Symplectic ODE-Net which provides a systematic way to incorporate prior knowledge of Hamiltonian dynamics with control into a deep learning framework. We show that SymODEN achieves better prediction with fewer training samples by learning an interpretable, physically-consistent state-space model. Future works will incorporate a broader class of physics-based prior, such as the port-Hamiltonian system formulation, to learn dynamics of a larger class of physical systems. SymODEN can work with embedded angle data or when we only have access to velocity instead of generalized momentum. Future works would explore other types of embedding, such as embedded 3D orientations. Another interesting direction could be to combine energy shaping control (potential as well as kinetic energy shaping) with interpretable end-to-end learning frameworks.

Acknowledgments

This research was inspired by the ideas and plans articulated by N. E. Leonard and A. Majumdar, Princeton University, in their ONR grant #N00014-18-1-2873. The research was primarily carried out during Y. D. Zhong's internship at Siemens Corporation, Corporate Technology. Pre- and post-internship, Y. D. Zhong's work was supported by ONR grant #N00014-18-1-2873.

11.6 Appendix

11.6.1 Experiment Implementation Details

The architectures used for our experiments are shown below. For all the tasks, SymODEN has the lowest number of total parameters. To ensure that the learned function is smooth, we use Tanh activation function instead of ReLu. As we have differentiation in the computation graph, non-smooth activation functions would lead to discontinuities in the derivatives. This, in turn, would result in an ODE with a discontinuous RHS which is not desirable. All the architectures shown below are fully-connected neural networks. The first number indicates the dimension of the input layer. The last number indicates the dimension of output layer. The dimension of hidden layers is shown in the middle along with the activation functions.

Task 1: Pendulum

- Input: 2 state dimensions, 1 action dimension
- Baseline Model (0.36M parameters): 2 - 600Tanh - 600Tanh - 2Linear
- Unstructured SymODEN (0.20M parameters):
  - H_{θ_1,θ_2}: 2 - 400Tanh - 400Tanh - 1Linear
  - g_{θ_3}: 1 - 200Tanh - 200Tanh - 1Linear
- SymODEN (0.13M parameters):
  - $M^{-1}_{\theta_1}$: 1 - 300Tanh - 300Tanh - 1Linear
  - V_{θ_2}: 1 - 50Tanh - 50Tanh - 1Linear
  - g_{θ_3}: 1 - 200Tanh - 200Tanh - 1Linear

Task 2: Pendulum with embedded data

- Input: 3 state dimensions, 1 action dimension
- Naive Baseline Model (0.65M parameters): 4 - 800Tanh - 800Tanh - 3Linear
- Geometric Baseline Model (0.46M parameters):
  - $M^{-1}_{\theta_1} = L_{\theta_1} L^T_{\theta_1}$, where L_{θ_1}: 2 - 300Tanh - 300Tanh - 300Tanh - 1Linear
  - approximate $(\dot{q}, \dot{p})$: 4 - 600Tanh - 600Tanh - 2Linear

- Unstructured SymODEN (0.39M parameters):
  - $M^{-1}_{\theta_1} = L_{\theta_1} L^T_{\theta_1}$, where L_{θ_1}: 2 - 300Tanh - 300Tanh - 300Tanh - 1Linear
  - H_{θ_2}: 3 - 500Tanh - 500Tanh - 1Linear
  - g_{θ_3}: 2 - 200Tanh - 200Tanh - 1Linear
- SymODEN (0.14M parameters):
  - $M^{-1}_{\theta_1} = L_{\theta_1} L^T_{\theta_1}$, where L_{θ_1}: 2 - 300Tanh - 300Tanh - 300Tanh - 1Linear
  - V_{θ_2}: 2 - 50Tanh - 50Tanh - 1Linear
  - g_{θ_3}: 2 - 200Tanh - 200Tanh - 1Linear

Task 3: CartPole

- Input: 5 state dimensions, 1 action dimension
- Naive Baseline Model (1.01M parameters): 6 - 1000Tanh - 1000Tanh - 5Linear
- Geometric Baseline Model (0.82M parameters):
  - $M^{-1}_{\theta_1} = L_{\theta_1} L^T_{\theta_1}$, where L_{θ_1}: 3 - 400Tanh - 400Tanh - 400Tanh - 3Linear
  - approximate $(\dot{\mathbf{q}}, \dot{\mathbf{p}})$: 6 - 700Tanh - 700Tanh - 4Linear
- Unstructured SymODEN (0.67M parameters):
  - $M^{-1}_{\theta_1} = L_{\theta_1} L^T_{\theta_1}$, where L_{θ_1}: 3 - 400Tanh - 400Tanh - 400Tanh - 3Linear
  - H_{θ_2}: 5 - 500Tanh - 500Tanh - 1Linear
  - g_{θ_3}: 3 - 300Tanh - 300Tanh - 2Linear
- SymODEN (0.51M parameters):
  - $M^{-1}_{\theta_1} = L_{\theta_1} L^T_{\theta_1}$, where L_{θ_1}: 3 - 400Tanh - 400Tanh - 400Tanh - 3Linear
  - V_{θ_2}: 3 - 300Tanh - 300Tanh - 1Linear
  - g_{θ_3}: 3 - 300Tanh - 300Tanh - 2Linear

Task 4:Acrobot

- Input: 6 state dimensions, 1 action dimension
- Naive Baseline Model (1.46M parameters): 7 - 1200Tanh - 1200Tanh - 6Linear

- Geometric Baseline Model (0.97M parameters):
  - $M_{\theta_1}^{-1} = L_{\theta_1} L_{\theta_1}^T$, where L_{θ_1}: 4 - 400Tanh - 400Tanh - 400Tanh - 3Linear
  - approximate $(\dot{\mathbf{q}}, \dot{\mathbf{p}})$: 7 - 800Tanh - 800Tanh - 4Linear
- Unstructured SymODEN (0.78M parameters):
  - $M_{\theta_1}^{-1} = L_{\theta_1} L_{\theta_1}^T$, where L_{θ_1}: 4 - 400Tanh - 400Tanh - 400Tanh - 3Linear
  - H_{θ_2}: 6 - 600Tanh - 600Tanh - 1Linear
  - g_{θ_3}: 4 - 300Tanh - 300Tanh - 2Linear
- SymODEN (0.51M parameters):
  - $M_{\theta_1}^{-1} = L_{\theta_1} L_{\theta_1}^T$, where L_{θ_1}: 4 - 400Tanh - 400Tanh - 400Tanh - 3Linear
  - V_{θ_2}: 4 - 300Tanh - 300Tanh - 1Linear
  - g_{θ_3}: 4 - 300Tanh - 300Tanh - 2Linear

11.6.2 Special Case of Energy-based Controller - PD Controller with Energy Compensation

The energy-based controller has the form $\mathbf{u}(\mathbf{q}, \mathbf{p}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\mathbf{p})$, where the potential energy shaping term $\boldsymbol{\beta}(\mathbf{q})$ and the damping injection term $\mathbf{v}(\mathbf{p})$ are given by Equation (11.7) and Equation (11.8), respectively.

If the desired potential energy $V_q(\mathbf{q})$ is given by a quadratic, as in Equation (11.9), then

$$\begin{aligned}\boldsymbol{\beta}(\mathbf{q}) &= \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \frac{\partial V_d}{\partial \mathbf{q}}\Big) \\ &= \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star)\Big),\end{aligned} \tag{11.28}$$

and the controller can be expressed as

$$\mathbf{u}(\mathbf{q}, \mathbf{p}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\mathbf{p}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star) - \mathbf{K}_d\mathbf{p}\Big). \tag{11.29}$$

The corresponding external forcing term is then given by

$$\mathbf{g}(\mathbf{q})\mathbf{u} = \frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p(\mathbf{q} - \mathbf{q}^\star) - \mathbf{K}_d\mathbf{p}, \tag{11.30}$$

which is same as Equation (11.10) in the main body of the paper. The first term in this external forcing provides an energy compensation, whereas the second term and the last term are proportional and derivative control terms, respectively. Thus, this control can be perceived as a PD controller with an additional energy compensation.

11.6.3 Ablation Study of Differentiable ODE Solver

In Hamiltonian Neural Networks (HNN), [33] incorporate the Hamiltonian structure into learning by minimizing the difference between the symplectic gradients and the true gradients. When the true gradient is not available, which is often the case, the authors suggested using finite difference approximations. In SymODEN, true gradients or gradient approximations are not necessary since we integrate the estimated gradient using differentiable ODE solvers and set up the loss function with the integrated values. Here we perform an ablation study of the differentiable ODE Solver.

Both HNN and the Unstructured SymODEN approximate the Hamiltonian by a neural network and the main difference is the differentiable ODE solver, so we compare the performance of HNN and the Unstructured SymODEN. We set the time horizon $\tau = 1$ since it naturally corresponds to the finite difference estimate of the gradient. A larger τ would correspond to higher-order estimates of gradients. Since there is no angle-aware design in HNN, we use Task 1 to compare the performance of these two models.

We generate 25 training trajectories, each of which contains 45 time steps. This is consistent with the HNN paper. In the HNN paper [33], the initial conditions of the trajectories are generated randomly in an annulus, whereas in this paper, we generate the initial state conditions uniformly in a reasonable range in each state dimension. We guess the reason that the authors of HNN choose the annulus data generation is that they do not have an angle-aware design. Take the pendulum for example; all the training and test trajectories they generate do not pass the inverted position. If they make prediction on a trajectory with a large enough initial speed, the angle would go over $\pm 2\pi$, $\pm 4\pi$, etc. in the long run. Since these are away from the region where the model gets trained, we can expect the prediction would be poor. In fact, this motivates us to design the angle-aware SymODEN in Section 11.3.3. In this ablation study, we generate the training data in both ways.

Table 11.1 shows the train error and the prediction error per trajectory of the two models. We can see Unstructured SymODEN performs better than HNN. This is an expected result. To see why this is the case, let us assume the training loss per time step of HNN is similar to that of Unstructured SymODEN. Since the training loss is on the symplectic gradient, the error would accumulate while

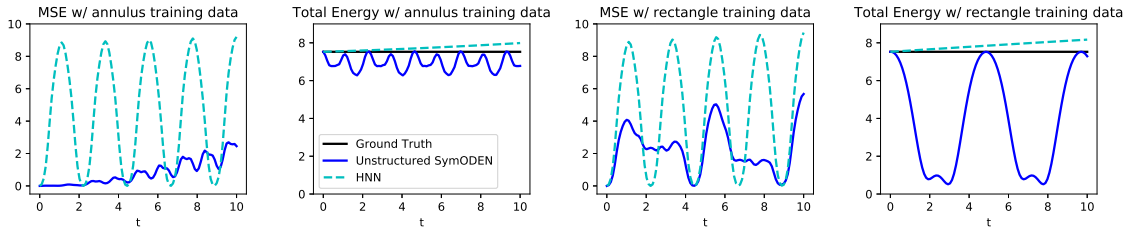

Figure 11.7: MSE and Total energy of a sample test trajectory. Left two figures: the training data for the models are randomly generated in an annulus, the same as in HNN. Right two figures: the training data for the models are randomly generated in a rectangle - the same way that we use in SymODEN.

integrating the symplectic gradient to get the estimated state values, and MSE of the state values would likely be one order of magnitude greater than that of Unstructured SymODEN. Figure 11.7 shows the MSE and total energy of a particular trajectory. It is clear that the MSE of the Unstructured SymODEN is lower than that of HNN. The MSE of HNN periodically touches zero does not mean it has a good prediction at that time step. Since the trajectories in the phase space are closed circles, those zeros mean the predicted trajectory of HNN lags behind (or runs ahead of) the true trajectory by one or more circles. Also, the energy of the HNN trajectory drifts instead of staying constant, probably because the finite difference approximation is not accurate enough.

Table 11.1: Train error and prediction error per trajectory of Unstructured SymODEN and HNN. The train error per trajectory is the sum of MSE of all the 45 timesteps averaged over the 25 training trajectories. The prediction error per trajectory is the sum of MSE of 90 timesteps in a trajectory.

Models	annulus training data		rectangle training data	
	train error	prediction error	train error	prediction error
Unstructured SymODEN	56.59	440.78	502.60	4363.87
HNN	290.67	564.16	5457.80	26209.17

11.6.4 Effects of the time horizon τ

Incorporating the differential ODE solver also introduces two hyperparameters: solver types and time horizon τ. For the solver types, the Euler solver is not accurate enough for our tasks. The adaptive solver “dopri5" lead to similar train error, test error and prediction error as the RK4 solver, but requires more time during training. Thus, in our experiments, we choose RK4.

Time horizon τ is the number of points we use to construct our loss function. Table 11.2 shows the train error, test error and prediction error per trajectory in Task 2 when τ is varied from 1 to 5. We can see that longer time horizons lead to better models. This is expected since long time horizons penalize worse long term predictions. We also observe in our experiments that longer

time horizons require more time to train the models.

Table 11.2: Train error, test error and prediction error per trajectory of Task 2

Time Horizon	$\tau = 1$	$\tau = 2$	$\tau = 3$	$\tau = 4$	$\tau = 5$
Train Error	0.744	0.136	0.068	0.033	0.017
Test Error	0.579	0.098	0.052	0.024	0.012
Prediction Error	3.138	0.502	0.199	0.095	0.048

11.6.5 Fully-actuated Cartpole and Acrobot

CartPole and Acrobot are underactuated systems. Incorporating the control of underactuated systems into the end-to-end learning framework is our future work. Here we trained SymODEN on fully-actuated versions of Cartpole and Acrobot and synthesized controllers based on the learned model.

For the fully-actuated CartPole, Figure 11.8 shows the snapshots of the system of a controlled trajectory with an initial condition where the pole is below the horizon. Figure 11.9 shows the time series of state variables and control inputs. We can successfully learn the dynamics and control the pole to the inverted position and the cart to the origin.

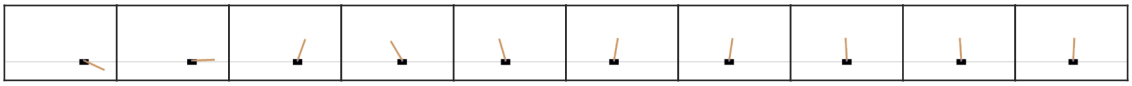

Figure 11.8: Snapshots of a controlled trajectory of the fully-actuated CartPole system with a 0.3s time interval.

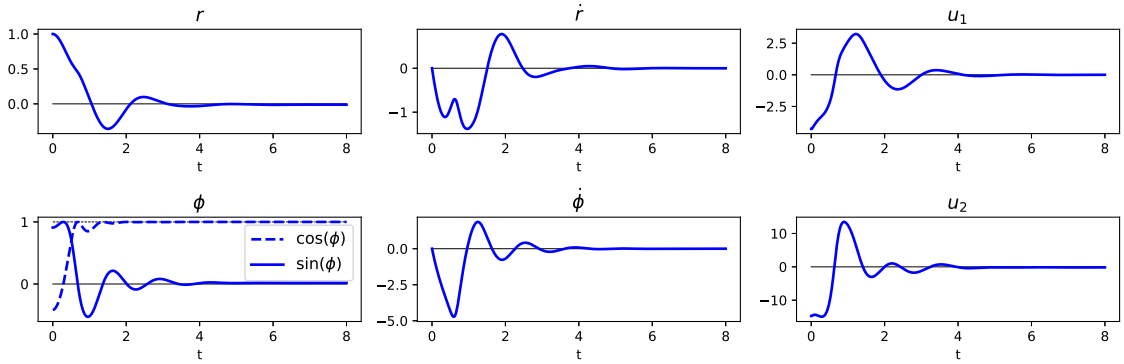

Figure 11.9: Time series of state variables and control inputs of a controlled trajectory shown in Figure 11.8. Black reference lines indicate expected value in the end.

For the fully-actuated Acrobot, Figure 11.10 shows the snapshots of a controlled trajectory. Figure 11.11 shows the time series of state variables and control inputs. We can successfully control the Acrobot from the downward position to the upward position, though the final value of q_2 is a little

away from zero. Taking into account that the dynamics has been learned with only 64 different initial state conditions, it is most likely that the upward position did not show up in the training data.

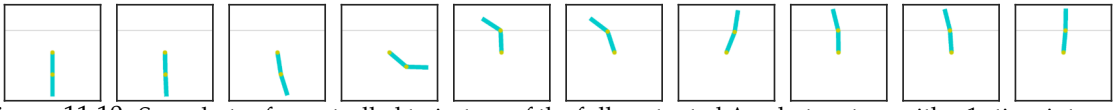

Figure 11.10: Snapshots of a controlled trajectory of the fully-actuated Acrobot system with a 1s time interval.

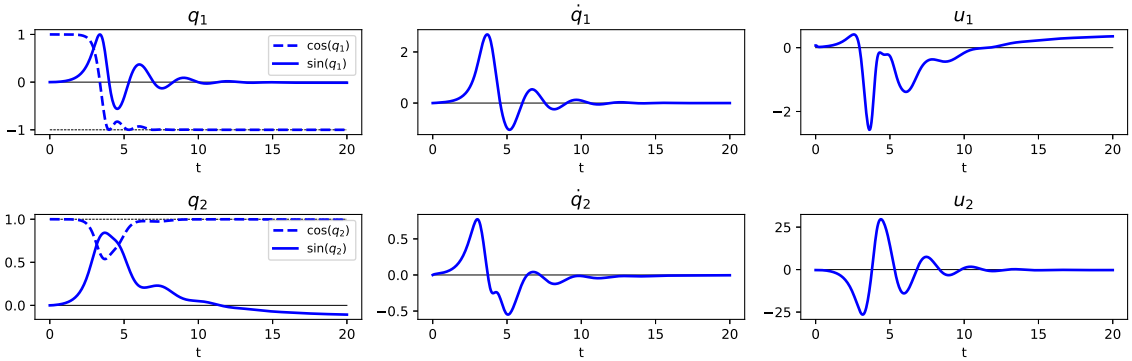

Figure 11.11: Time series of state variables and control inputs of a controlled trajectory shown in Figure 11.10. Black reference lines indicate expected value in the end.

11.6.6 Test Errors of the Tasks

Here we show statistics of train, test, and prediction per trajectory in all four tasks. The train errors are based on 64 initial state conditions and 5 constant inputs. The test errors are based on 64 previously unseen initial state conditions and the same 5 constant inputs. Each trajectory in the train and test set contains 20 steps. The prediction error is based on the same 64 initial state conditions (during training) and zero inputs.

Table 11.3: Train, Test and Prediction errors of the Four Tasks

	Naive Baseline	Geometric Baseline	Unstructured Symplectic-ODE	Symplectic-ODE
Task 1: Pendulum				
Model Parameter	0.36M	N/A	0.20M	0.13M
Train error	30.82 ± 43.45	N/A	0.89 ± 2.76	1.50 ± 4.17
Test error	40.99 ± 56.28	N/A	2.74 ± 9.94	2.34 ± 5.79
Prediction error	37.87 ± 117.02	N/A	17.17 ± 71.48	23.95 ± 66.61
Task 2: Pendulum (embed)				
Model Parameter	0.65M	0.46M	0.39M	0.14M
Train error	2.31 ± 3.72	0.59 ± 1.634	1.76 ± 3.69	0.067 ± 0.276
Test error	2.18 ± 3.59	0.49 ± 1.762	1.41 ± 2.82	0.052 ± 0.241
Prediction error	317.21 ± 521.46	14.31 ± 29.54	3.69 ± 7.72	0.20 ± 0.49
Task3: CartPole				
Model Parameter	1.01M	0.82M	0.67M	0.51M
Train error	15.53 ± 22.52	0.45 ± 0.37	4.84 ± 4.42	1.78 ± 1.81
Test error	25.42 ± 38.49	1.20 ± 2.67	6.90 ± 8.66	1.89 ± 1.81
Prediction error	332.44 ± 245.24	52.26 ± 73.25	225.22 ± 194.24	11.41 ± 16.06
Task 4: Acrobot				
Model Parameter	1.46M	0.97M	0.78M	0.51M
Train error	2.04 ± 2.90	2.07 ± 3.72	1.32 ± 2.08	0.25 ± 0.39
Test error	5.62 ± 9.29	5.12 ± 7.25	3.33 ± 6.00	0.28 ± 0.48
Prediction error	64.61 ± 145.20	26.68 ± 34.90	9.72 ± 16.58	2.07 ± 5.26

Chapter 12

Dissipative SymODEN: Encoding Hamiltonian Dynamics with Dissipation and Control into Deep Learning

Yaofeng Desmond Zhong, Biswadip Dey, Amit Chakraborty

Appears as Zhong et al. [101] in *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*

In this work, we introduce *Dissipative SymODEN*, a deep learning architecture which can infer the dynamics of a physical system with dissipation from observed state trajectories. To improve prediction accuracy while reducing network size, Dissipative SymODEN encodes the port-Hamiltonian dynamics with energy dissipation and external input into the design of its computation graph and learns the dynamics in a structured way. The learned model, by revealing key aspects of the system, such as the inertia, dissipation, and potential energy, paves the way for energy-based controllers.

12.1 Introduction

Inferring systems dynamics from observed trajectories plays a critical role in identification and control of complex, physical systems, such as robotic manipulators [54] and HVAC systems [93]. Although the use of neural networks in this context has a rich history of more than three decades [63], recent advances in deep learning [29] have led to renewed interest in this topic [89, 43, 49, 14, 5]. Deep neural networks learn underlying patterns from data and enable generalization beyond the training set by incorporating appropriate inductive bias into the learning approach. To promote representations that are *simple* in some sense, inductive bias [37, 7] often manifests itself via a set of assumptions and guides a learning algorithm to pick one hypothesis over another. The success in predicting an outcome for previously unseen data depends on how well the inductive bias captures the ground reality. Inductive bias can be introduced as the prior in a Bayesian model, or via the choice of computation graphs in a neural network.

Incorporation of physics-based priors into deep learning has been a key focus in the recent times. As these approaches use neural networks to approximate system dynamics, they are more expressive than traditional system identification techniques [84]. By using a directed graph to capture the causal relationships in a physical system, [77] introduces a recurrent graph network to infer latent space dynamics in robotic systems. [56] and [34] leveraged Lagrangian mechanics to learn the dynamics of kinematic structures from discrete observations. On the other hand, [33] and [101] have utilized Hamiltonian mechanics for learning dynamics from data. However, strict enforcement of the Hamiltonian prior is restrictive for real-life systems which often loses energy in a structured way (e.g. frictional losses in robotic arms, resistive losses in power grids, etc.).

To explicitly encode dissipation as a prior into end-to-end learning, we expand the scope of the Symplectic ODE-Net (SymODEN) architecture [101] and propose *Dissipative SymODEN*. The underlying dynamics is motivated by the port-Hamiltonian formulation [66], which has a correction term accounting for the prior of dissipation. With this term, Dissipative SymODEN can accommodate the energy losses from various sources of dissipation present in real-life systems. Our results show that inclusion of dissipation into the physics-informed SymODEN architecture improves its prediction accuracy and out-of-sample behavior, while offering insight about relevant physical properties of the system (such as inertia matrix, potential energy, energy dissipation etc.). These insights, in turn, can enable the use of energy-based controllers, such as the method of controlled Lagrangian [11] and interconnection & damping assignment [66], which offer performance guarantees for complex,

nonlinear systems.

Contribution: The main contribution of this work is the introduction of a physics-informed learning architecture called Dissipative SymODEN which encodes a non-conservative physics, i.e. Hamiltonian dynamics with energy dissipation, into deep learning. This provides a means to uncover the dynamics of real-life physical systems whose Hamiltonian aspects have been adapted to external input and energy dissipation. By ensuring that the computation graph is aligned with the underlying physics, we achieve transparency, better predictions with smaller networks, and improved generalization. The architecture of Dissipative SymODEN has also been designed to accommodate angle data in the embedded form. Additionally, we use differentiable ODE solvers to avoid the need for derivative estimation.

12.2 The Port-Hamiltonian Dynamics

Hamiltonian dynamics is often used to systematically describe the dynamics of a physical system in the phase space $(\mathbf{q}, \mathbf{p})$, where $\mathbf{q} = (q_1, q_2, ..., q_n)$ is the generalized coordinate and $\mathbf{p} = (p_1, p_2, ..., p_n)$ is the generalized momentum. In this approach, the key to the dynamics is a scalar function $H(\mathbf{q}, \mathbf{p})$, which is referred to as the Hamiltonian. In almost all physical systems, the Hamiltonian represents the total energy which can be expressed as

$$H(\mathbf{q}, \mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}(\mathbf{q})\mathbf{p} + V(\mathbf{q}), \tag{12.1}$$

where $\mathbf{M}(\mathbf{q})$ is the symmetric positive definite mass/inertia matrix and $V(\mathbf{q})$ represents the potential energy of the system. The equations of motion are governed by the symplectic gradient [73] of the Hamiltonian, i.e.,

$$\dot{\mathbf{q}} = \frac{\partial H}{\partial \mathbf{p}} \qquad \dot{\mathbf{p}} = -\frac{\partial H}{\partial \mathbf{q}}. \tag{12.2}$$

Moreover, since $\dot{H} = (\frac{\partial H}{\partial \mathbf{q}})^T\dot{\mathbf{q}} + (\frac{\partial H}{\partial \mathbf{p}})^T\dot{\mathbf{p}} = 0$, moving along the symplectic gradient conserves the Hamiltonian (i.e. the total energy). However, although the classical Hamiltonian dynamics ensures energy conservation, it fails to model dissipation and external inputs, which often appear in real-life systems. The port-Hamiltonian dynamics generalizes the classical Hamiltonian dynamics by explicitly modelling the total energy, dissipation and external inputs. Motivated by this formulation,

we consider the following port-Hamiltonian dynamics in this work:

$$\begin{bmatrix}\dot{\mathbf{q}}\\ \dot{\mathbf{p}}\end{bmatrix} = \left(\begin{bmatrix}\mathbf{0} & \mathbf{I}\\ -\mathbf{I} & \mathbf{0}\end{bmatrix} - \mathbf{D}(\mathbf{q})\right)\begin{bmatrix}\frac{\partial H}{\partial \mathbf{q}}\\ \frac{\partial H}{\partial \mathbf{p}}\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{g}(\mathbf{q})\end{bmatrix}\mathbf{u}, \tag{12.3}$$

where the dissipation matrix $\mathbf{D}(\mathbf{q})$ is symmetric positive semi-definite and represents energy dissipation. The external input $\mathbf{u}$ is usually affine and only affects the generalized momenta. The input matrix $\mathbf{g}(\mathbf{q})$ is assumed to have full column rank. As expected, with zero dissipation and zero input, (12.3) reduces to the classical Hamiltonian dynamics.

12.3 Dissipative Symplectic ODE-Net

12.3.1 Training Neural ODE with Constant Forcing

We focus on the problem of learning an ordinary differential equation (ODE) from observation data. Assume the analytical form of the right hand side (RHS) of an ODE "$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u})$" is unknown. An observation data $\mathbf{X} = ((\mathbf{x}_{t_0}, \mathbf{u}_c), ..., (\mathbf{x}_{t_n}, \mathbf{u}_c))$ with a constant input $\mathbf{u}_c$ allows us to approximate $\mathbf{f}(\mathbf{x}, \mathbf{u})$ with a neural net by leveraging

$$\begin{bmatrix}\dot{\mathbf{x}}\\ \dot{\mathbf{u}}\end{bmatrix}\Bigg|_{\mathbf{X}} = \begin{bmatrix}\mathbf{f}_\theta(\mathbf{x}, \mathbf{u})\\ \mathbf{0}\end{bmatrix} = \tilde{\mathbf{f}}_\theta(\mathbf{x}, \mathbf{u}). \tag{12.4}$$

Equation (12.4), by matching the input and output dimensions, enables us to feed it into Neural ODE [17]. With Neural ODE, we make predictions by approximating the RHS of (12.4) using a neural network and feed it into an ODE solver

$$(\mathbf{x}, \mathbf{u}_c)_{t_1, t_2, ..., t_n} = \text{ODESolve}((\mathbf{x}, \mathbf{u}_c)_{t_0}, \tilde{\mathbf{f}}_\theta, t_1, t_2, ..., t_n).$$

We can then construct the loss function $L = \|\mathbf{X} - \mathbf{X}\|_2^2$. In practice, we introduce the time horizon τ as a hyperparameter and predict $\mathbf{x}_{t_{i+1}}, \mathbf{x}_{t_{i+2}}, ..., \mathbf{x}_{t_{i+\tau}}$ from initial condition $\mathbf{x}_{t_i}$, where $i = 0, ..., n - \tau$. The problem is then how to design the network architecture of $\tilde{\mathbf{f}}_\theta$, or equivalently $\mathbf{f}_\theta$.

12.3.2 Learning from Generalized coordinate and Momentum

Suppose we have data consisting of $(\mathbf{q}, \mathbf{p}, \mathbf{u})_{t_0, ..., t_n}$, where $\mathbf{u}$ remains constant in each trajectory. We use four neural nets – $\mathbf{M}^{-1}_{\theta_1}(\mathbf{q})$, $V_{\theta_2}(\mathbf{q})$, $\mathbf{g}_{\theta_3}(\mathbf{q})$ and $\mathbf{D}_{\theta_4}(\mathbf{q})$ – as function approximators to represent the

inverse of mass matrix, potential energy, the input matrix and the dissipation matrix, respectively. Thus,

$$\mathbf{f}_\theta(\mathbf{q},\mathbf{p},\mathbf{u}) = \left(\begin{bmatrix}\mathbf{0} & \mathbf{I}\\ -\mathbf{I} & \mathbf{0}\end{bmatrix} - \mathbf{D}_{\theta_4}(\mathbf{q})\right)\begin{bmatrix}\frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{q}}\\ \frac{\partial H_{\theta_1,\theta_2}}{\partial \mathbf{p}}\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ \mathbf{g}_{\theta_3}(\mathbf{q})\end{bmatrix}\mathbf{u} \tag{12.5}$$

where

$$H_{\theta_1,\theta_2}(\mathbf{q},\mathbf{p}) = \frac{1}{2}\mathbf{p}^T\mathbf{M}^{-1}_{\theta_1}(\mathbf{q})\mathbf{p} + V_{\theta_2}(\mathbf{q}) \tag{12.6}$$

The partial derivative can be taken care of by automatic differentiation. By putting the designed $\mathbf{f}_\theta(\mathbf{q},\mathbf{p},\mathbf{u})$ into Neural ODE, we obtain a systematic way of adding the prior knowledge of a structured dynamics into end-to-end learning.

12.3.3 Learning from Embedded Angle Data

Often, especially in robotics, the state variables involve angles residing in the interval $[-\pi,\pi)$. In other words, each angle lies on the manifold $\mathbb{S}^1$. However, generalized coordinates are typically assumed to lie on $\mathbb{R}^n$. To bridge this gap, we use an angle-aware design [101] and assume that the generalized coordinates are angles available as $(\mathbf{x}_1(\mathbf{q}),\mathbf{x}_2(\mathbf{q}),\mathbf{x}_3(\dot{\mathbf{q}}),\mathbf{u})_{t_0,\dots,t_n} = (\cos\mathbf{q},\sin\mathbf{q},\dot{\mathbf{q}},\mathbf{u})_{t_0,\dots,t_n}$. Then, similar to [101], we aim to learn a structured dynamics (12.3) expressed in terms of $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. As $\mathbf{p} = \mathbf{M}(\mathbf{x}_1,\mathbf{x}_2)\dot{\mathbf{q}}$, we can express this dynamics as

$$\begin{aligned}
\dot{\mathbf{x}}_1 &= -\sin\mathbf{q}\circ\dot{\mathbf{q}} = -\mathbf{x}_2\circ\dot{\mathbf{q}}\\
\dot{\mathbf{x}}_2 &= \cos\mathbf{q}\circ\dot{\mathbf{q}} = \mathbf{x}_1\circ\dot{\mathbf{q}}\\
\dot{\mathbf{x}}_3 &= \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}(\mathbf{x}_1,\mathbf{x}_2)\mathbf{p}) = \frac{\mathrm{d}}{\mathrm{d}t}(\mathbf{M}^{-1}(\mathbf{x}_1,\mathbf{x}_2))\mathbf{p} + \mathbf{M}^{-1}(\mathbf{x}_1,\mathbf{x}_2)\dot{\mathbf{p}},
\end{aligned} \tag{12.7}$$

where "$\circ$" represents the element-wise product. We assume $\mathbf{q}$ and $\mathbf{p}$ evolve with the structured dynamics Equation (12.3) and substitute Equation (12.3) in to the RHS of Equation (12.7). Similar to our approach in Sec 12.3.2, we use four neural nets to express the RHS of Equation (12.7) as $\mathbf{f}_\theta(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{u})$. Thus, it can be fed into Equation 12.4 and the Neural ODE.

12.3.4 Learning on Hybrid Spaces $\mathbb{R}^n \times \mathbb{T}^m$

In most of physical systems, both translational coordinates and rotational coordinates coexist. In other words, the generalized coordinates lie on $\mathbb{R}^n \times \mathbb{T}^m$, where $\mathbb{T}^m$ denotes the m-torus. Here we put together the architecture of the previous two subsections. We assume the generalized coordinates are $\mathbf{q} = (\mathbf{r},\boldsymbol{\phi}) \in \mathbb{R}^n \times \mathbb{T}^m$ and the data comes in the form of $(\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3,\mathbf{x}_4,\mathbf{x}_5,\mathbf{u})_{t_0,\dots,t_n} =$

$(\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathbf{r}}, \dot{\boldsymbol{\phi}}, \mathbf{u})_{t_0,\ldots,t_n}$. We use neural nets – $\mathbf{M}^{-1}_{\theta_1}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, $V_{\theta_2}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$, $\mathbf{g}_{\theta_3}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ and $\mathbf{D}_{\theta_4}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ – as function approximators. Then the dynamics is given by

$$[\dot{\mathbf{x}}_1, \dot{\mathbf{x}}_2, \dot{\mathbf{x}}_3, \dot{\mathbf{x}}_4, \dot{\mathbf{x}}_5]^T = \mathbf{f}_\theta(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4, \mathbf{x}_5, \mathbf{u})$$

12.3.5 The Dissipation Matrix and the Mass matrix

As the dissipation matrix models energy dissipation such as friction and resistance, it is positive semi-definite. We impose this constraint in the network architecture by $\mathbf{D}_{\theta_4} = \mathbf{L}_{\theta_4}\mathbf{L}^T_{\theta_4}$, where $\mathbf{L}_{\theta_4}$ is a lower-triangular matrix. In real physical systems, both the mass matrix $\mathbf{M}$ and its inverse are positive definite. Similarly, semi-definiteness is constraint by $\mathbf{M}^{-1}_{\theta_1} = \mathbf{L}_{\theta_1}\mathbf{L}^T_{\theta_1}$, where $\mathbf{L}_{\theta_1}$ is a lower-triangular matrix. The positive definiteness is ensured by adding a small constant ϵ to the diagonal elements of $\mathbf{M}^{-1}_{\theta_1}$. It not only makes $\mathbf{M}_{\theta_1}$ invertible, but also stabilizes training.

12.4 Experiments

12.4.1 Experimental Setup

We use the following four tasks to evaluate the performance of Dissipative SymODEN architecture – (i) Task 1: a pendulum with generalized coordinate and momentum data; (ii) Task 2: a pendulum with embedded angle data; (iii) Task 3: a CartPole system; and (iv) Task 4: an Acrobot.

Model Variants: Besides the Dissipative SymODEN model derived above, we consider a variant, called *Unstructured (Unstr.) Dissipative SymODEN*, which approximates the Hamiltonian by a fully connected neural net H_{θ_1,θ_2}. We also consider the original *SymODEN* [101] as a model variant.

Baseline Models: We set up baseline models for all four experiments. For the pendulum with generalized coordinate and momentum data, the *naive baseline* model approximates (12.5) – $\mathbf{f}_\theta(\mathbf{x}, \mathbf{u})$ – by a fully connected neural net. For all the other experiments, which involves embedded angle data, we set up two different baseline models: *naive baseline* approximates $\mathbf{f}_\theta(\mathbf{x}, \mathbf{u})$ by a fully connected neural net. Also, we set up the *geometric baseline* model which approximates $\dot{\mathbf{q}}$ and $\dot{\mathbf{p}}$ with a fully connected neural net.

Data Generation: For all tasks, we randomly generated initial conditions of states and subsequently combined them with 5 different constant control inputs, i.e., $u = -2.0, -1.0, 0.0, 1.0, 2.0$, to produce the initial conditions and input required for simulation. The simulators integrate the corresponding dynamics for 20 time steps to generate trajectory data which is then used to construct

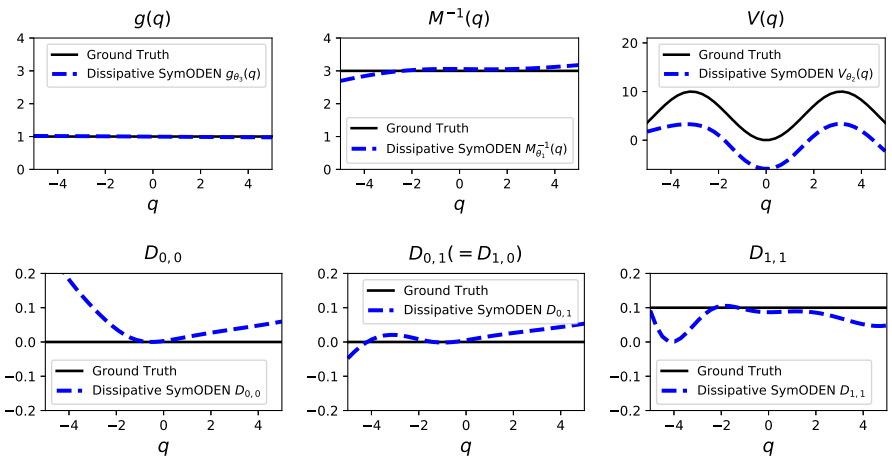

Figure 12.1: Learned functions in Task 1 (Pendulum).

the training set and test set.

Model training: In all the tasks, we train our model using Adam optimizer [46] with 1000 epochs. We set a time horizon $\tau = 3$, and choose "RK4" as the numerical integration scheme in Neural ODE. We logged the *train error*, *test error* and *prediction (pred.) error* per trajectory for all the tasks. Prediction error per trajectory is calculated by using the same initial state condition in the training set with a constant control of $u = 0.0$, integrating 40 time steps forward.

12.4.2 Task 1: Pendulum with Generalized Coordinate and Momentum Data

In this task, we use the model described in Section 12.3.2 and present the predicted trajectories of the learned models as well as the learned functions of Dissipative SymODEN. The underlying dynamics is given by

$$\dot{q} = 3p, \qquad \dot{p} = -5\sin q - 0.3p + u, \tag{12.8}$$

with the Hamiltonian $H(q,p) = 1.5p^2 + 5(1-\cos q)$. In other words $M^{-1}(q) = 3$, $V(q) = 5(1-\cos q)$, $g(q) = 1$ and $\mathbf{D}_{\theta_4}(q) = [0, 0; 0, 0.1]$. Figure 12.1 shows that the learned $g_{\theta_3}(q)$ and $M^{-1}_{\theta_1}(q)$ matches the ground truth pretty well. Also, $V_{\theta_2}(q)$ differs from the ground truth by an almost constant margin which is expected since only the derivative of $V_{\theta_2}(q)$ impacts the dynamics. The learned dissipation matrix $\mathbf{D}_{\theta_4}(q)$ does not match the ground truth. We address this issue in the next subsection. In Table 12.1, Naive Baseline's prediction error is the lowest because predicted trajectories reach the origin faster than the ground truth.

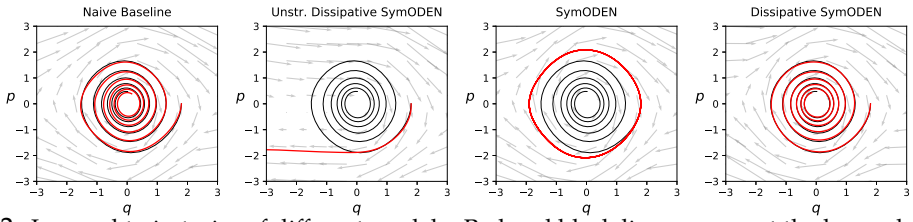

Figure 12.2: Learned trajectories of different models. Red and black lines represent the learned and ground truth trajectories, respectively and the gray arrows show the vector fields learned by each model. *Dissipative SymODEN* learns a more accurate vector field than the *naive baseline* model. Moreover, it appears that whereas *SymODEN* learns an energy-conserved vector field slightly different from the ground truth, *Unstructured Dissipative SymODEN* learns it completely wrong.

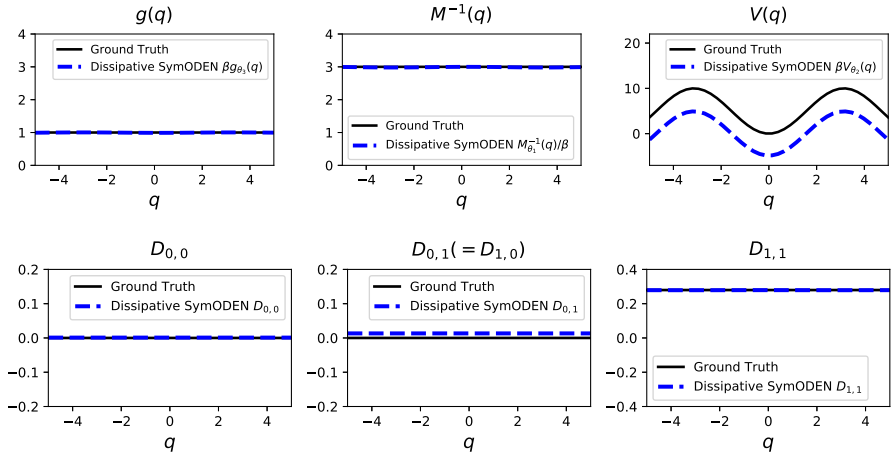

Figure 12.3: Learned functions in Task 2 (Pendulum with embedded data).

12.4.3 Task 2: Pendulum with Embedded Data

In this task, the dynamics is the same as Equation (12.8) but the training data are generated by the OpenAI Gym simulator, i.e. we use embedded angle data and assume we only have access to $\dot{q}$ instead of p. We use the model described in Section 12.3.3 to learn the structured dynamics. Without true p data, the learned function matches the ground truth with a scaling β, as shown in Figure 12.3. Please refer to [101] for explanation of the scaling. In this example, with the scaling $\beta = 0.357$, the learned functions match the ground truth. With the angle-aware design, we learned the dissipation matrix much better than the previous subsection.

12.4.4 Results

In Table 12.1, we show the train, test and prediction errors for all four tasks. Dissipative SymODEN performs the best in all three metrics. As SymODEN does not allow dissipation, it does not perform well in these tasks. Since Unstructured Dissipative SymODEN architecture has trouble learning a

good vector field, it performs the worst in all the tasks except Task 2. In conclusion, Dissipative SymODEN achieves higher accuracy with less model parameters. Moreover, the learned model reveals physical aspects of the system, which can be leveraged by energy-based controllers.

Table 12.1: Train, Test and Prediction Errors of Four Tasks

Task		Naive Baseline	Geometric Baseline	UnStr. Dissipative SymODEN	SymODEN	Dissipative SymODEN
1	*#Parameters*	0.36M	N/A	0.22M	0.13M	0.15M
	Train error	26.38 ± 38.00	N/A	34.80 ± 68.53	4.47 ± 6.40	0.88 ± 1.41
	Test error	35.03 ± 49.89	N/A	49.44 ± 81.31	7.52 ± 10.13	1.25 ± 1.81
	Pred. error	32.544 ± 36.203	N/A	219.36 ± 296.86	96.50 ± 99.56	34.03 ± 47.83
2	*#Parameters*	0.65M	0.46M	0.41M	0.14M	0.16M
	Train error	2.02 ± 4.41	0.42 ± 1.16	1.90 ± 3.85	2.37 ± 2.71	0.15 ± 0.27
	Test error	2.01 ± 4.99	0.33 ± 1.22	1.61 ± 3.36	2.67 ± 2.83	0.13 ± 0.25
	Pred. error	40.18 ± 78.10	0.81 ± 0.68	7.04 ± 13.65	72.78 ± 90.42	1.04 ± 1.3
3	*#Parameters*	1.01M	0.82M	0.69M	0.51M	0.53M
	Train error	12.92 ± 15.58	0.48 ± 0.50	12.09 ± 18.38	3.33 ± 3.85	0.88 ± 0.89
	Test error	20.07 ± 26.42	1.34 ± 3.19	19.87 ± 23.16	3.80 ± 3.71	1.37 ± 1.30
	Pred. error	268.24 ± 204.15	60.12 ± 96.18	366.38 ± 405.45	30.21 ± 34.33	8.32 ± 7.81
4	*#Parameters*	1.46M	0.97M	0.80M	0.51M	0.53M
	Train error	1.76 ± 2.26	1.90 ± 2.82	77.56 ± 111.50	2.92 ± 2.58	0.47 ± 0.64
	Test error	5.12 ± 9.14	4.87 ± 7.42	122.70 ± 190.90	5.27 ± 6.55	0.81 ± 1.10
	Pred. error	36.65 ± 77.16	44.26 ± 95.70	590.77 ± 807.88	68.26 ± 103.46	12.72 ± 32.12

Acknowledgments

This research was inspired by the ideas and plans articulated by N. E. Leonard and A. Majumdar, Princeton University, in their ONR grant #N00014-18-1-2873. The research was primarily carried out during Y. D. Zhong's internship at Siemens Corporation, Corporate Technology. Pre- and post-internship, Y. D. Zhong's work was supported by ONR grant #N00014-18-1-2873.

Chapter 13

Unsupervised Learning of Lagrangian Dynamics from Images for Prediction and Control

Yaofeng Desmond Zhong, Naomi Ehrich Leonard

To appear as Zhong and Leonard [99] in *the 34th Conference on Neural Information Processing Systems (NeurIPS 2020)*

Recent approaches for modelling dynamics of physical systems with neural networks enforce Lagrangian or Hamiltonian structure to improve prediction and generalization. However, these approaches fail to handle the case when coordinates are embedded in high-dimensional data such as images. We introduce a new unsupervised neural network model that learns Lagrangian dynamics from images, with interpretability that benefits prediction and control. The model infers Lagrangian dynamics on generalized coordinates that are simultaneously learned with a coordinate-aware variational autoencoder (VAE). The VAE is designed to account for the geometry of physical systems composed of multiple rigid bodies in the plane. By inferring interpretable Lagrangian dynamics, the model learns physical system properties, such as kinetic and potential energy, which enables long-term prediction of dynamics in the image space and synthesis of energy-based controllers.

13.1 Introduction

In the past decade, deep learning has achieved significant success in computer vision [38], natural language processing [85] and sequential decision making [82]. Recently, an increasing number of works have leveraged deep neural networks to model physical systems. Neural network models are able to find patterns from data and generalize those patterns beyond training data, partly because they incorporate appropriate priors through design of the neural network architecture. Since Lagrangian/Hamiltonian dynamics represent a broad class of physical systems, recent approaches have incorporated Lagrangian/Hamiltonian dynamics as physics priors [56, 20, 33, 101, 102], in physical system modeling, to improve prediction and generalization. These approaches, however, require coordinate data, which are not always available in real-world applications. Another class of approaches learn physical models from images, by either learning the map from images to coordinates with supervision on true coordinate data [90] or learning the coordinates in an unsupervised way but only with translational coordinates [57, 50]. The unsupervised learning of rotational coordinates such as angles of objects are under-explored in the literature.

In this work, we propose an unsupervised neural network model that learns coordinates and Lagrangian dynamics on those coordinates from images physical systems in motion in the plane. The latent dynamical model enforces Lagrangian dynamics, which benefits long term prediction of the system. As Lagrangian dynamics commonly involve rotational coordinates to describe the changing configurations of objects in the system, we propose a coordinate-aware variational autoencoder (VAE) that can infer interpretable rotational and translational coordinates from images without supervision. The interpretable coordinates together with the interpretable Lagrangian dynamics pave the way for introducing energy-based controllers of the learned dynamics.

13.1.1 Related work

Lagrangian/Hamiltonian prior in learning dynamics To improve prediction and generalization of physical system modelling, a class of approaches has incorporated the physics prior of Hamiltonian or Lagrangian dynamics into deep learning. Deep Lagrangian Network [56] and Lagrangian Neural Network [20] learn Lagrangian dynamics from position, velocity and acceleration data. Hamiltonian Neural Networks [33] learn Hamiltonian dynamics from position, velocity and acceleration data. By leveraging ODE integrators, Hamiltonian Graph Networks [78] and Symplectic ODE-Net [101] learn Hamiltonian dynamics from only position and velocity data. All of these works require direct

observation of low dimensional position and velocity data.

Unsupervised learning of dynamics With little position and velocity data, Belbute-Peres et al. [8] learn underlying dynamics. However, the authors observed that their model fails to learn meaningful dynamics when there is no supervision on position and velocity data at all. Without supervision, Watter et al. [89] and Levine et al. [53] learn locally linear dynamics and Jaques et al. [42] learns unknown parameters in latent dynamics with a given form. Kossen et al. [48] extracts position and velocity of each object from videos and learns the underlying dynamics. Watters et al. [91] adopts an object-oriented design to gain data efficiency and robustness. Battaglia et al. [6], Sanchez-Gonzalez et al. [76] and Watters et al. [90] learn dynamics with supervision by taking into account the prior of objects and their relations. These object-oriented designs focus little on rotational coordinates. Variational Integrator Network [74] considers rotational coordinates but cannot handle systems with multiple rotational coordinates.

13.2 Preliminary concepts

13.2.1 Lagrangian dynamics

Lagrangian dynamics are a reformulation of Newton's second law of motion. The configuration of a system in motion at time t is described by generalized coordinates $\mathbf{q}(t) = (q_1(t), q_2(t), ..., q_m(t))$, where m is the number of degrees of freedom (DOF) of the system. For planar rigid body systems with n rigid bodies and k holonomic constraints, the DOF is $m = 3n - k$. From D'Alembert's principle, the equations of motion of the system, also known as the Euler-Lagrange equation, are

$$\frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{\partial L}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{Q}^{nc}, \tag{13.1}$$

where the scalar function $L(\mathbf{q}, \dot{\mathbf{q}})$ is the Lagrangian, $\dot{\mathbf{q}} = \mathrm{d}\mathbf{q}/\mathrm{d}t$, and $\mathbf{Q}^{nc}$ is a vector of non-conservative generalized forces. The Lagrangian $L(\mathbf{q}, \dot{\mathbf{q}})$ is the difference between kinetic energy $T(\mathbf{q}, \dot{\mathbf{q}})$ and potential energy $V(\mathbf{q})$. For rigid body systems, the Lagrangian is

$$L(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V(\mathbf{q}), \tag{13.2}$$

where $\mathbf{M}(\mathbf{q})$ is the mass matrix. In this work, we assume that the control inputs are the only non-conservative generalized forces, i.e., $\mathbf{Q}^{nc} = \mathbf{g}(\mathbf{q})\mathbf{u}$, where $\mathbf{g}(\mathbf{q})$ is the input matrix and $\mathbf{u}$ is a vector

of control inputs such as forces or torques. Substituting $\mathbf{Q}^{nc} = \mathbf{g}(\mathbf{q})\mathbf{u}$ and $L(\mathbf{q}, \dot{\mathbf{q}})$ from (13.2) into (13.1), we get the equations of motion in the form of m second-order ordinary differential equations (ODE):

$$\ddot{\mathbf{q}} = \mathbf{M}^{-1}(\mathbf{q})\Big(-\frac{1}{2}\frac{\mathrm{d}\mathbf{M}(\mathbf{q})}{\mathrm{d}t}\dot{\mathbf{q}} - \frac{\mathrm{d}V(\mathbf{q})}{\mathrm{d}\mathbf{q}} + \mathbf{g}(\mathbf{q})\mathbf{u}\Big). \tag{13.3}$$

13.2.2 Control via energy shaping

Our goal is to control the system to a reference configuration $\mathbf{q}^\star$, inferred from a goal image $\mathbf{x}^\star$, based on the learned dynamics. As we are essentially learning the kinetic and potential energy associated with the system, we can leverage the learned energy for control by *energy shaping* [65, 10].

If $\text{rank}(\mathbf{g}(\mathbf{q})) = m$, we have control over every DOF and the system is fully actuated. For such systems, control to the reference configuration $\mathbf{q}^\star$ can be achieved with the control law $\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\dot{\mathbf{q}})$, where $\boldsymbol{\beta}(\mathbf{q})$ is the *potential energy shaping* and $\mathbf{v}(\dot{\mathbf{q}})$ is the *damping injection*. The goal of potential energy shaping is to let the system behave as if it is governed by a desired Lagrangian L_d with no non-conservative generalized forces.

$$\frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\partial L}{\partial \dot{\mathbf{q}}}\Big) - \frac{\partial L}{\partial \mathbf{q}} = \mathbf{g}(\mathbf{q})\boldsymbol{\beta}(\mathbf{q}) \quad \Longleftrightarrow \quad \frac{\mathrm{d}}{\mathrm{d}t}\Big(\frac{\partial L_d}{\partial \dot{\mathbf{q}}}\Big) - \frac{\partial L_d}{\partial \mathbf{q}} = 0, \tag{13.4}$$

where the desired Lagrangian has desired potential energy $V_d(\mathbf{q})$:

$$L_d(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) - V_d(\mathbf{q}) = \frac{1}{2}\dot{\mathbf{q}}^T\mathbf{M}(\mathbf{q})\dot{\mathbf{q}} - V_d(\mathbf{q}). \tag{13.5}$$

The difference between L_d and L is the difference between V and V_d, which explains the name potential energy shaping: $\boldsymbol{\beta}(\mathbf{q})$ shapes the potential energy V of the original system into a desired potential energy V_d. The potential energy V_d is designed to have a global minimum at $\mathbf{q}^\star$. By the equivalence (13.4), we get

$$\boldsymbol{\beta}(\mathbf{q}) = \mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}\Big(\frac{\partial V}{\partial \mathbf{q}} - \frac{\partial V_d}{\partial \mathbf{q}}\Big). \tag{13.6}$$

With only potential energy shaping, the system dynamics will oscillate around $\mathbf{q}^\star$.[1] The purpose of damping injection $\mathbf{v}(\dot{\mathbf{q}})$ is to impose convergence, exponentially in time, to $\mathbf{q}^\star$. The damping injection has the form

$$\mathbf{v}(\dot{\mathbf{q}}) = -\mathbf{g}^T(\mathbf{g}\mathbf{g}^T)^{-1}(\mathbf{K}_d\dot{\mathbf{q}}). \tag{13.7}$$

For underactuated systems, however, this controller design is not valid since $\mathbf{g}\mathbf{g}^T$ will not be invert-

[1] Please see Supplementary Materials for more details.

ible. In general, we also need kinetic energy shaping [10] to achieve a control goal.

Remark The design parameters here are V_d and $\mathbf{K}_d$. A quadratic desired potential energy

$$V_d(\mathbf{q}) = \frac{1}{2}(\mathbf{q} - \mathbf{q}^\star)^T \mathbf{K}_p (\mathbf{q} - \mathbf{q}^\star), \tag{13.8}$$

results in a controller design

$$\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \mathbf{g}^T (\mathbf{g}\mathbf{g}^T)^{-1} \left(\frac{\partial V}{\partial \mathbf{q}} - \mathbf{K}_p (\mathbf{q} - \mathbf{q}^\star) - \mathbf{K}_d \dot{\mathbf{q}} \right). \tag{13.9}$$

This can be interpreted as a proportional-derivative (PD) controller with energy compensation.

13.2.3 Training Neural ODE with constant control

The Lagrangian dynamics can be formulated as a set of first-order ODE

$$\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{u}), \tag{13.10}$$

where $\mathbf{s}$ is a state vector and unknown vector field $\mathbf{f}$, which is a vector-valued function, can be parameterized with a neural network $\mathbf{f}_\psi$. We leverage Neural ODE, proposed by Chen et al. [17], to learn the function $\mathbf{f}$ that explains the trajectory data of $\mathbf{s}$. The idea is to predict future states from an initial state by integrating the ODE with an ODE solver. As all the operations in the ODE solver are differentiable, $\mathbf{f}_\psi$ can be updated by back-propagating through the ODE solver and approximating the true $\mathbf{f}$. However, Neural ODE cannot be applied to (13.10) directly since the input dimension and the output dimension of $\mathbf{f}$ are not the same. Zhong et al. [101] showed that if the control remains constant for each trajectory in the training data, Neural ODE can be applied to the following augmented ODE:

$$\begin{pmatrix} \dot{\mathbf{s}} \\ \dot{\mathbf{u}} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_\psi(\mathbf{s}, \mathbf{u}) \\ \mathbf{0} \end{pmatrix} = \tilde{\mathbf{f}}_\psi(\mathbf{s}, \mathbf{u}). \tag{13.11}$$

With a learned $\mathbf{f}_\psi$, we can apply a controller design $\mathbf{u} = \mathbf{u}(\mathbf{s})$ that is not constant, e.g., an energy-based controller, by integrating the ODE $\dot{\mathbf{s}} = \mathbf{f}(\mathbf{s}, \mathbf{u}(\mathbf{s}))$.

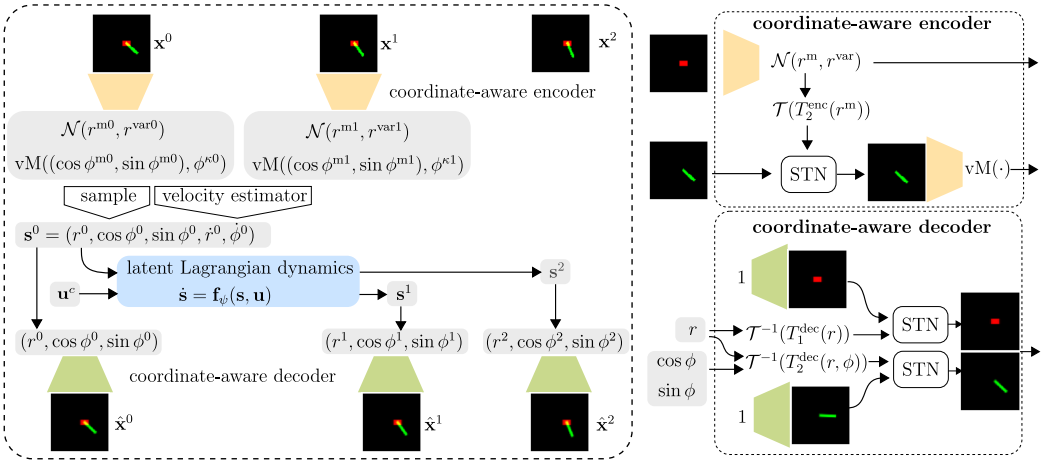

Figure 13.1: **Left**: Model architecture. (Using CartPole as an illustrative example.) The initial state $\mathbf{s}^0$ is constructed by sampling the distribution and a velocity estimator. The latent Lagrangian dynamics take $\mathbf{s}^0$ and the constant control $\mathbf{u}^c$ for that trajectory and predict future states up to T_{pred}. The diagram shows the $T_{\text{pred}} = 2$ case. **Top-right**: The coordinate-aware encoder estimates the distribution of generalized coordinates. **Bottom-right**: The initial and predicted generalized coordinates are decoded to the reconstruction images with the coordinate-aware decoder.

13.3 Model architecture

Let $\mathbf{X} = ((\mathbf{x}^0, \mathbf{u}^c), (\mathbf{x}^1, \mathbf{u}^c)), ..., (\mathbf{x}^{T_{\text{pred}}}, \mathbf{u}^c))$ be a given sequence of image and control pairs, where $\mathbf{x}^\tau$, $\tau = 0, 1, \ldots, T_{\text{pred}}$, is the image of the trajectory of a rigid-body system under constant control $\mathbf{u}^c$ at time $t = \tau \Delta t$. From $\mathbf{X}$ we want to learn a state-space model (13.10) that governs the time evolution of the rigid-body system dynamics. We assume the number of rigid bodies n is known and the segmentation of each object in the image is given. Each image can be written as $\mathbf{x}^\tau = (\mathbf{x}_1^\tau, ..., \mathbf{x}_n^\tau)$, where $\mathbf{x}_i^\tau \in \mathbb{R}^{n_x}$ contains visual information about the ith rigid body at $t = \tau \Delta t$ and n_x is the dimension of the image space.

In Section 13.3.1, we parameterize $\mathbf{f}(\mathbf{s}, \mathbf{u})$ with a neural network and design the architecture of the neural network such that (13.10) is constrained to follow Lagrangian dynamics, where the physical properties such as mass and potential energy are learned from data. Since we have no access to state data, we need to infer states $\mathbf{s}$, i.e., generalized coordinates and velocities from image data. Sections 13.3.2 and 13.3.4 introduce an inference model (encoder) and a generative model (decoder) pair. Together they make up a variational autoencoder (VAE) [47] to infer the generalized coordinates in an unsupervised way. Section 13.3.3 introduces a simple estimator of velocity from learned generalized coordinates. The VAE and the state-space model are trained together, as described in Section 13.3.5. The model architecture is shown in Figure 13.1.

13.3.1 Latent Lagrangian dynamics

The Lagrangian dynamics (13.3) yield a second-order ODE. From a model-based perspective, they can be re-written as a first-order ODE (13.10) by choosing the state as $\mathbf{s} = (\mathbf{q}, \dot{\mathbf{q}})$. However, from a data-driven perspective, this choice of state is problematic when the generalized coordinates involve angles. Consider the pendulum task in Figure 13.2 as an example where we want to infer the generalized coordinate, i.e., the angle of the pendulum ϕ, from an image of the pendulum. The map from the image to the angle ϕ should be bijective. However, if we choose the state as $\mathbf{s} = (\phi, \dot{\phi})$, the map is not bijective, since ϕ and $\phi + 2\pi$ map to the same image. If we restrict $\phi \in [-\pi, \pi)$, then the dynamics are not continuous when the pendulum moves around the inverted position. Inspired by Zhong et al. [101], we solve this issue by proposing the state as $\mathbf{s} = (\cos\phi, \sin\phi, \dot{\phi})$, such that the mapping from the pendulum image to $(\cos\phi, \sin\phi)$ is bijective.

In general, for a planar rigid-body system with $\mathbf{q} = (\mathbf{r}, \boldsymbol{\phi})$, where $\mathbf{r} \in \mathbb{R}^{m_R}$ are translational generalized coordinates and $\boldsymbol{\phi} \in \mathbb{T}^{m_T}$ are rotational generalized coordinates , the proposed state is $\mathbf{s} = (\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3, \mathbf{s}_4, \mathbf{s}_5) = (\mathbf{r}, \cos\boldsymbol{\phi}, \sin\boldsymbol{\phi}, \dot{\mathbf{r}}, \dot{\boldsymbol{\phi}})$, where cos and sin are applied element-wise to $\boldsymbol{\phi}$. To enforce Lagrangian dynamics in the state-space model, we take the derivative of $\mathbf{s}$ with respect to t and substitute in (13.3) to get

$$\dot{\mathbf{s}} = \begin{pmatrix} \mathbf{s}_4 \\ -\mathbf{s}_3 \circ \mathbf{s}_5 \\ \mathbf{s}_2 \circ \mathbf{s}_5 \\ \mathbf{M}^{-1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)\left(-\frac{1}{2}\frac{\mathrm{d}\mathbf{M}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)}{\mathrm{d}t}\begin{pmatrix}\mathbf{s}_4 \\ \mathbf{s}_5\end{pmatrix} + \begin{pmatrix} -\frac{\partial V(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)}{\partial \mathbf{s}_1} \\ \frac{\partial V(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)}{\partial \mathbf{s}_2}\mathbf{s}_3 - \frac{\partial V(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)}{\partial \mathbf{s}_3}\mathbf{s}_2 \end{pmatrix} + \mathbf{g}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)\mathbf{u}\right) \end{pmatrix} \tag{13.12}$$

where $\circ$ is the element-wise product. We use three neural networks, $\mathbf{M}_{\psi_1}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$,$V_{\psi_2}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, and $\mathbf{g}_{\psi_3}(\mathbf{s}_1, \mathbf{s}_2, \mathbf{s}_3)$, to approximate the mass matrix, the potential energy and the input matrix, respectively. Equation (13.12) is then a state-space model parameterized by a neural network $\dot{\mathbf{s}} = \mathbf{f}_\psi(\mathbf{s}, \mathbf{u})$. It can be trained as stated in Section 13.2.3 given the initial condition $\mathbf{s}^0$. Next, we present the means to infer $\mathbf{s}^0 = (\mathbf{r}^0, \cos\boldsymbol{\phi}^0, \sin\boldsymbol{\phi}^0, \dot{\mathbf{r}}^0, \dot{\boldsymbol{\phi}}^0)$ and $\mathbf{u}^c$ from the given images.

13.3.2 Coordinate-aware encoder

From a latent variable modelling perspective, an image $\mathbf{x}$ of a rigid-body system can be generated by first specifying the values of the generalized coordinates and then assigning values to pixels

based on the generalized coordinates with a generative model - the decoder. In order to infer those generalized coordinates from images, we need an inference model - the encoder. We perform variational inference with a coordinate-aware VAE.

The coordinate-aware encoder infers a distribution on the generalized coordinates. The Gaussian distribution is the default for modelling latent variables in VAE. This is appropriate for modelling a translational generalized coordinate r since r resides in $\mathbb{R}^1$. However, this is not appropriate for modelling a rotational generalized coordinate ϕ since a Gaussian distribution is not a distribution on $\mathbb{S}^1$. If we use a Gaussian distribution to model hyperspherical latent variables, the VAE performs worse than a traditional autoencoder [21]. Thus, to model ϕ, we use the von Mises (vM) distribution, a family of distributions on $\mathbb{S}^1$. Analogous to a Gaussian distribution, a Von Mises distribution is characterized by two parameters: $\mu \in \mathbb{R}^2$, $||\mu||^2 = 1$ is the mean, and $\kappa \in \mathbb{R}_{\geq 0}$ is the concentration around μ. The Von Mises distribution reduces to a uniform distribution when $\kappa = 0$.

In our model, for a rotational generalized coordinate ϕ, we assume a posterior distribution $Q(\phi|\mathbf{x}) = \text{vM}((\cos\phi^{\text{m}}, \sin\phi^{\text{m}}), \phi^{\kappa})$ with prior $P(\phi) = \text{vM}(\cdot, 0) = U(\mathbb{S}^1)$. For a translational generalized coordinate r, we assume a posterior distribution $Q(r|\mathbf{x}) = \mathcal{N}(r^{\text{m}}, r^{\text{var}})$ with prior $\mathcal{N}(0, 1)$. We denote the joint posterior distribution as $Q(\mathbf{q}|\mathbf{x})$ and joint prior distribution as $P(\mathbf{q})$. The encoder is a neural network that takes an image as input and provides the parameters of the distributions as output. A black-box neural network encoder would not be able to learn interpretable generalized coordinates for a system in motion described by Lagrangian dynamics. Instead, we propose a coordinate-aware encoder by designing the architecture of the neural network to account for the geometry of the system. This is the key to interpretable encoding of generalized coordinates.

Recall that each generalized coordinate q_j specifies the position/angle of a rigid body i_j in the system. In principle, the coordinate can be learned from the image segmentation of i_j. However, the reference frame of a generalized coordinate might depend on other generalized coordinates and change across images. Take the CartPole example in Figure 13.2 as motivation. The system has two DOF and natural choices of generalized coordinates are the horizontal position of the cart $q_1 = r$ and the angle of the pole $q_2 = \phi$. The origin of the reference frame of r is the center of the image, which is the same across all images. The origin of the reference frame of ϕ, however,

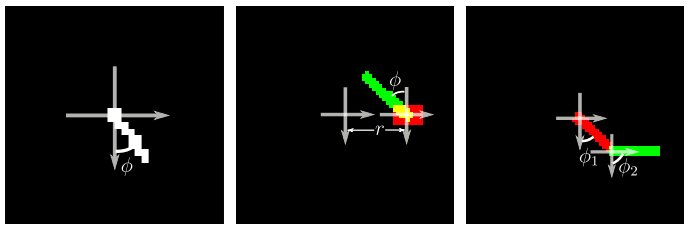

Figure 13.2: One choice of generalized coordinates and their corresponding reference frames in three example systems

is the center of the cart, which is not the same across all the images since the cart can move. In order to learn the angle of the pole, we can either use a translation invariant architecture such as Convolution Neural Networks (CNN) or place the center of the encoding attention window of the pole segmentation image at the center of the cart. The former approach does not work well in extracting generalized coordinates.[2] Thus, we adopt the latter approach, where we shift our encoding attention window horizontally with direction and magnitude given by generalized coordinate r, before feeding it into the encoder to learn ϕ. In this way we exploit the geometry of the system in the encoder.

The default attention window is the image grid and corresponds to the default reference frame, where the origin is at the center of the image with horizontal and vertical axes. The above encoding attention window mechanism for a general system can be formalized by considering the transformation from the default reference frame to the reference frame of each generalized coordinate. The transformation of a point (x_d, y_d) in the default reference frame to a point (x_t, y_t) in the target reference frame is captured by transformation $\mathcal{T}(x, y, \theta)$ corresponding to translation by (x, y) and rotation by θ as follows:

$$\begin{pmatrix} x_t \\ y_t \\ 1 \end{pmatrix} = \mathcal{T}(x, y, \theta) \begin{pmatrix} x_d \\ y_d \\ 1 \end{pmatrix}, \quad \text{where} \quad \mathcal{T}(x, y, \theta) = \begin{pmatrix} \cos\theta & \sin\theta & x \\ -\sin\theta & \cos\theta & y \\ 0 & 0 & 1 \end{pmatrix}. \tag{13.13}$$

So let $\mathcal{T}((x, y, \theta)_j^{\text{enc}})$ be the transformation from default frame to reference frame for the generalized coordinate q_j. This transformation depends on constant parameters $\mathbf{c}$ associated with the shape and size of the rigid bodies and generalized coordinates $\mathbf{q}_{-j}$, which denotes the vector of generalized coordinates with q_j removed. Let $(x, y, \theta)_j^{\text{enc}} = T_j^{\text{enc}}(\mathbf{q}_{-j}, \mathbf{c})$. Both $\mathbf{q}_{-j}$ and $\mathbf{c}$ are learned from images. However, the function T_j^{enc} is specified by leveraging the geometry of the system. In the CartPole example, $(q_1, q_2) = (r, \phi)$, and $T_1^{\text{enc}} \equiv (0, 0, 0)$ and $T_2^{\text{enc}}(q_1) = (q_1, 0, 0)$. In the Acrobot example, $(q_1, q_2) = (\phi_1, \phi_2)$, and $T_1^{\text{enc}} \equiv (0, 0, 0)$ and $T_2^{\text{enc}}(q_1, l_1) = (l_1 \sin q_1, l_1 \cos q_1, 0)$.

The shift of attention window can be implemented with a spatial transformer network (STN) [41], which generates a transformed image $\tilde{\mathbf{x}}_{i_j}$ from $\mathbf{x}_{i_j}$, i.e., $\tilde{\mathbf{x}}_{i_j} = \text{STN}(\mathbf{x}_{i_j}, \mathcal{T}(T_j^{\text{enc}}(\mathbf{q}_{-j}, \mathbf{c})))$. In general, to encode q_j, we use a multilayer perceptron (MLP) that takes $\tilde{\mathbf{x}}_{i_j}$ as input and provides the parameters of the q_j distribution as output. For a translational coordinate q_j, we have $(q_j^{\text{m}}, \log q_j^{\text{var}}) =$

[2]Here we expect to encode the angle of the pole from a pole image regardless of where it appears in the image. As the translation invariance of CNN is shown by Kauderer-Abrams [44] to be primarily dependent on data augmentation, the encoding of generalized coordinates might not generalize well to unseen trajectories. Also, in general we need both translation invariance and rotation invariance, a property that CNN do not have.

$\text{MLP}_j^{\text{enc}}(\tilde{\mathbf{x}}_{i_j})$. For a rotational coordinate q_j, we have $(\alpha_j, \beta_j, \log q_j^\kappa) = \text{MLP}_j^{\text{enc}}(\tilde{\mathbf{x}}_{i_j})$, where the mean of the von Mises distribution is computed as $(\cos q_j^m, \sin q_j^m) = (\alpha_j, \beta_j)/\sqrt{\alpha_j^2 + \beta_j^2}$. We then take a sample from the q_j distribution.[3] Doing this for every generalized coordinate q_j, we can get $(\mathbf{r}^\tau, \cos\boldsymbol{\phi}^\tau, \sin\boldsymbol{\phi}^\tau)$ from $\mathbf{x}^\tau$ for any τ.[4] We will use $(\mathbf{r}^0, \cos\boldsymbol{\phi}^0, \sin\boldsymbol{\phi}^0)$ and $(\mathbf{r}^1, \cos\boldsymbol{\phi}^1, \sin\boldsymbol{\phi}^1)$.

13.3.3 Velocity estimator

To integrate Equation (13.12), we also need to infer $(\dot{\mathbf{r}}^0, \dot{\boldsymbol{\phi}}^0)$, the initial velocity. We can estimate the initial velocity from the encoded generalized coordinates by finite difference. We use the following simple first-order finite difference estimator:

$$\dot{\mathbf{r}}^0 = (\mathbf{r}^{\text{m}1} - \mathbf{r}^{\text{m}0})/\Delta t, \tag{13.14}$$

$$\dot{\boldsymbol{\phi}}^0 = \big((\sin\boldsymbol{\phi}^{\text{m}1} - \sin\boldsymbol{\phi}^{\text{m}0}) \circ \cos\boldsymbol{\phi}^{\text{m}0} - (\cos\boldsymbol{\phi}^{\text{m}1} - \cos\boldsymbol{\phi}^{\text{m}0}) \circ \sin\boldsymbol{\phi}^{\text{m}0}\big)/\Delta t, \tag{13.15}$$

where $(\mathbf{r}^{\text{m}0}, \cos\boldsymbol{\phi}^{\text{m}0}, \sin\boldsymbol{\phi}^{\text{m}0})$ and $(\mathbf{r}^{\text{m}1}, \cos\boldsymbol{\phi}^{\text{m}1}, \sin\boldsymbol{\phi}^{\text{m}1})$ are the means of the generalized coordinates encoded from the image at time $t = 0$ and $t = \Delta t$, respectively. Jaques et al. [42] proposed to use a neural network to estimate velocity. From our experiments, our simple estimator works better than a neural network estimator.

13.3.4 Coordinate-aware decoder

The decoder provides a distribution $P(\mathbf{x}|\mathbf{q}) = \mathcal{N}(\hat{\mathbf{x}}, \mathbf{I})$ as output, given a generalized coordinate $\mathbf{q}$ as input, where the mean $\hat{\mathbf{x}}$ is the reconstruction image of the image data $\mathbf{x}$. Instead of using a black box decoder, we propose a coordinate-aware decoder. The coordinate-aware decoder first generates a static image $\mathbf{x}_i^c$ of every rigid body i in the system, at a default position and orientation, using a MLP with a constant input, i.e., $\mathbf{x}_i^c = \text{MLP}_i^{\text{dec}}(1)$. The coordinate-aware decoder then determines $\hat{\mathbf{x}}_i$, the image of rigid body i positioned and oriented on the image plane according to the generalized coordinates. The proposed decoder is inspired by the coordinate-consistent decoder by Jaques et al. [42]. However, the decoder of [42] cannot handle a system of multiple rigid bodies with constraints such as the Acrobot and the CartPole, whereas our coordinate-aware decoder can.

As in Jaques et al. [42], to find $\hat{\mathbf{x}}_i$ we use the inverse transformation matrix $\mathcal{T}^{-1}((x, y, \theta)_i^{\text{dec}})$

[3] We use the reparametrization trick proposed by Davidson et al. [21] to sample from a von Mises distribution.

[4] For a transformation that depends on one or more generalized coordinate, those generalized coordinates must be encoded before the transformation can be applied. In the CartPole example, we need to encode r before applying the transformation to put the attention window centered at the cart to encode ϕ. We use the mean of the distribution, i.e., q_j^{m} or $(\cos q_j^m, \sin q_j^m)$, for those transformations that depend on q_j.

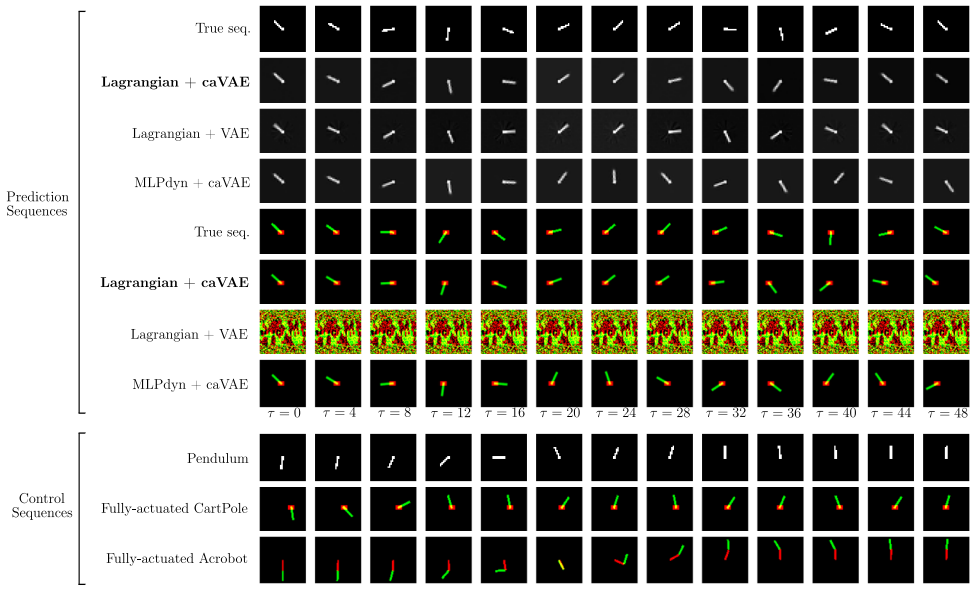

Figure 13.3: **Top**: Prediction sequences of Pendulum and CartPole with a previously unseen initial condition and zero control. Prediction results show both Lagrangian dynamics and coordinate-aware VAE are necessary to perform long term prediction. **Bottom**: Control sequences of three systems. Energy-based controllers are able to control the systems to the goal positions based on learned dynamics and encoding with Lagrangian+caVAE.

where $\mathcal{T}$ is given by (13.13) and $(x, y, \theta)_i^{\text{dec}} = T_i^{\text{dec}}(\mathbf{q}, \mathbf{c})$. In the CartPole example, $(q_1, q_2) = (r, \phi)$, and $T_1^{\text{dec}}(r) = (r, 0, 0)$ and $T_2^{\text{dec}}(r, \phi) = (r, 0, \phi)$. In the Acrobot example, $(q_1, q_2) = (\phi_1, \phi_2)$, and $T_1^{\text{dec}}(\phi_1) = (0, 0, \phi_1)$ and $T_2^{\text{dec}}(\phi_1, \phi_2) = (l_1 \sin\phi_1, l_1 \cos\phi_1, \phi_2)$. The reconstruction image is then $\hat{\mathbf{x}} = (\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n)$, where $\hat{\mathbf{x}}_i = \text{STN}(\mathbf{x}_i^c, \mathcal{T}^{-1}(T_i^{\text{dec}}(\mathbf{q}, \mathbf{c})))$.

13.3.5 Loss function

The loss $\mathcal{L}(\mathbf{X})$ consists of the sum of three terms:

$$\mathcal{L}(\mathbf{X}) = \underbrace{-\mathbb{E}_{\mathbf{q}^0 \sim Q}[\log P(\mathbf{x}^0|\mathbf{q}^0)] + \text{KL}(Q(\mathbf{q}^0|\mathbf{x}^0)||P(\mathbf{q}^0))}_{\text{VAE loss}} + \underbrace{\sum_{\tau=1}^{T_{\text{pred}}} ||\hat{\mathbf{x}}^\tau - \mathbf{x}^\tau||_2^2}_{\text{prediction loss}} + \underbrace{\lambda \sum_j \sqrt{\alpha_j^2 + \beta_j^2}}_{\text{vM regularization}}. \tag{13.16}$$

The VAE loss is a variational bound on the marginal log-likelihood of initial data $P(\mathbf{x}^0)$. The prediction loss captures inaccurate predictions of the latent Lagrangian dynamics. The vM regularization with weight λ penalizes large norms of vectors (α_j, β_j), preventing them from blowing up.

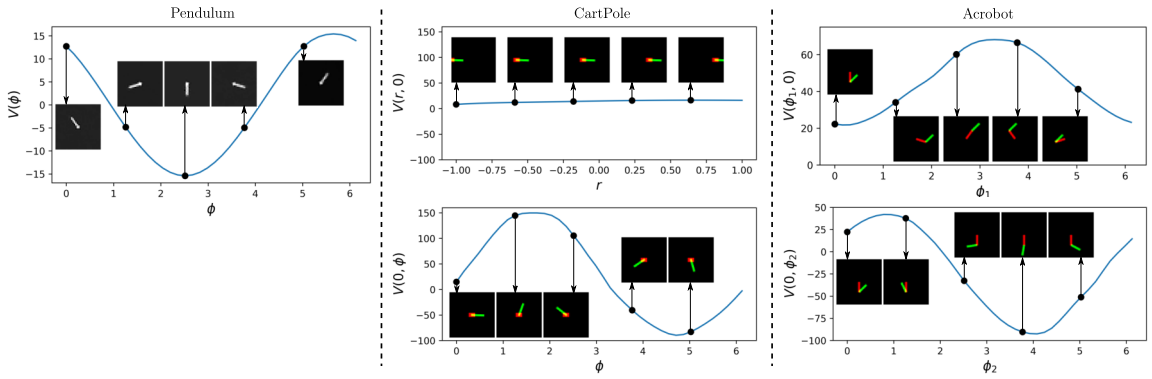

Figure 13.4: Learned potential energy with Lagrangian+caVAE of three systems and reconstruction images at selected coordinates. Both the learned coordinates and potential energy are interpretable.

13.4 Results

We train our model on three systems: the Pendulum, the fully-actuated CartPole and the fully-actuated Acrobot. The training images are generated by OpenAI Gym simulator [13]. The training setup is detailed in Supplementary Materials. As the mean square error in the image space is not a good metric of long term prediction accuracy [57], we report on the prediction image sequences of a previously unseen initial condition and highlight the interpretability of our model.

Lagrangian dynamics and coordinate-aware VAE improve prediction. As the Acrobot is a chaotic system, accurate long term prediction is impossible. Figure 13.3 shows the prediction sequences of images up to 48 time steps of the Pendulum and CartPole experiments with models trained with $T_{\text{pred}} = 4$. We compare the prediction results of our model (labelled as *Lagrangian+caVAE*) with two model variants: *MLPdyn+caVAE*, which replaces the Lagrangian latent dynamics with MLP latent dynamics, and *Lagrangian+VAE*, which replaces the coordinate-aware VAE with a traditional VAE. The traditional VAE fails to reconstruct meaningful images for CartPole, although it works well in the simpler Pendulum system. With well-learned coordinates, models that enforce Lagrangian dynamics result in better long term prediction, e.g., as compared to MLPdyn+caVAE, since Lagrangian dynamics with zero control preserves energy (see Supplementary Materials).

Learned potential energy enables energy-based control. Figure 13.4 shows the learned potential energy of the three systems and reconstruction images at selected coordinates with Lagrangian+caVAE. The learned potential energy is consistent with the true potential energy of those systems, e.g., the pendulum at the upward position has the highest potential energy while the pendulum at the downward position has the lowest potential energy. Figure 13.4 also visualizes the

learned coordinates. Learning interpretable coordinates and potential energy enables energy-based controllers. Based on the learned encoding and dynamics, we are able to control Pendulum and fully-actuated Acrobot to the inverted position, and fully-actuated CartPole to a position where the pole points upward. The sequences of images of controlled trajectories as shown in Figure 13.3 are generated based on learned dynamics and encoding with Lagrangian+caVAE as follows. We first encode an image of the goal position $\mathbf{x}^\star$ to the goal generalized coordinates $\mathbf{q}^\star$. At each time step, the OpenAI Gym simulator of a system can take a control input, integrate one time step forward, and output an image of the system at the next time step. The control input to the simulator is $\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\dot{\mathbf{q}})$ which is designed as in Section 13.2.2 with the learned potential energy, input matrix, coordinates encoded from the output images, and $\mathbf{q}^\star$.

Ablation study To understand which component in our model contributes to learning interpretable generalized coordinates the most, we also report results of four ablations, which are obtained by (a) replacing the coordinate-aware encoder with a black-box MLP, (b) replacing the coordinate-aware decoder with a black-box MLP, (c) replacing the coordinate-aware VAE with a coordinate-aware AE, and (d) a Physics-as-inverse-graphics (PAIG) model [42]. We observe that the coordinate-aware decoder makes the primary contribution to learning interpretable coordinates, and the coordinate-aware encoder makes a secondary contribution. The coordinate-aware AE succeeds in Pendulum and Acrobot tasks but fails in the CartPole task. PAIG uses AE with a neural network velocity estimator. We find that PAIG's velocity estimator overfits the training data, which results in inaccurate long term prediction. Please see Supplementary Materials for prediction sequences of the ablation study.

13.5 Conclusion

We propose an unsupervised model that learns planar rigid-body dynamics from images in an explainable and transparent way by incorporating the physics prior of Lagrangian dynamics and a coordinate-aware VAE, both of which we show are important for accurate prediction in the image space. The interpretability of the model allows for synthesis of model-based controllers.

Broader Impact

We focus on the impact of using our model to provide explanations for physical system modelling. Our model could be used to provide explanations regarding the underlying symmetries, i.e., con-

servation laws, of physical systems. Further, the incorporation of the physics prior of Lagrangian dynamics improves robustness and generalizability for both prediction and control applications.

We see opportunities for research applying our model to improve transparency and explanability in reinforcement learning, which is typically solved with low-dimensional observation data instead of image data. Our work also enables future research on vision-based controllers. The limitations of our work will also motivate research on unsupervised segmentation of images of physical systems.

Acknowledgements

This research has been supported in part by ONR grant #N00014-18-1-2873 and by the School of Engineering and Applied Science at Princeton University through the generosity of William Addy '82. Yaofeng Desmond Zhong would like to thank Christine Allen-Blanchette, Shinkyu Park, Sushant Veer and Anirudha Majumdar for helpful discussions.

13.6 Supplementary Materials

13.6.1 Conservation of energy in Lagrangian dynamics

In the following, we review the well known result from Lagrangian mechanics, which shows that with no control applied, the latent Lagrangian dynamics conserve energy, see, e.g., Goldstein et al. [27], Hand and Finch [35].

Theorem 10 (Conservation of Energy in Lagrangian Dynamics)**.** *Consider a system with Lagrangian dynamics given by Equation (3). If no control is applied to the system, i.e,* $\mathbf{u} = \mathbf{0}$*, then the total system energy* $E(\mathbf{q}, \dot{\mathbf{q}}) = T(\mathbf{q}, \dot{\mathbf{q}}) + V(\mathbf{q})$ *is conserved.*

Proof. We compute the derivative of total energy with respect to time and use the fact that, for any real physical system, the mass matrix is symmetric positive definite. We compute

$$\begin{aligned}
\frac{\mathrm{d}E(\mathbf{q}, \dot{\mathbf{q}})}{\mathrm{d}t} &= \frac{\partial E}{\partial \mathbf{q}} \dot{\mathbf{q}} + \frac{\partial E}{\partial \dot{\mathbf{q}}} \ddot{\mathbf{q}} \\
&= \frac{1}{2} \dot{\mathbf{q}}^T \frac{\mathrm{d}\mathbf{M}(\mathbf{q})}{\mathrm{d}t} \dot{\mathbf{q}} + \dot{\mathbf{q}}^T \frac{\mathrm{d}V(\mathbf{q})}{\mathrm{d}\mathbf{q}} + \dot{\mathbf{q}}^T \mathbf{M}(\mathbf{q}) \ddot{\mathbf{q}} \\
&= \dot{\mathbf{q}}^T \mathbf{g}(\mathbf{q}) \mathbf{u},
\end{aligned}$$

where we have substituted in Equation (3). Thus, if $\mathbf{u} = \mathbf{0}$, the total energy $E(\mathbf{q}, \dot{\mathbf{q}})$ is conserved. □

With Lagrangian dynamics as our latent dynamics model, we automatically incorporate a prior of energy conservation into physical system modelling. This explains why our latent Lagrangian dynamics result in better prediction, as shown in Figure 3.

This property of energy conservation also benefits the design of energy-based controllers $\mathbf{u}(\mathbf{q}, \dot{\mathbf{q}}) = \boldsymbol{\beta}(\mathbf{q}) + \mathbf{v}(\dot{\mathbf{q}})$. With only potential energy shaping $\boldsymbol{\beta}(\mathbf{q})$, we shape the potential energy so that the system behaves as if it is governed by a desired Lagrangian L_d. Thus, the total energy is still conserved, and the system would oscillate around the global minimum of the desired potential energy V_d, which is $\mathbf{q}^\star$. To impose convergence to $\mathbf{q}^\star$, we add damping injection $\mathbf{v}(\dot{\mathbf{q}})$. In this way, we systematically design an interpretable controller.

13.6.2 Experimental setup

Data generation

All the data are generated by OpenAI Gym simulator. For all tasks, we combine 256 initial conditions generated by OpenAI Gym with 5 different constant control values, i.e., $u = -2.0, -1.0, 0.0, 1.0, 2.0$. For those experiments with multi-dimensional control inputs, we apply these 5 constant values to each dimension while setting the value of the rest of the dimensions to be 0. The purpose is to learn a good $\mathbf{g}(\mathbf{q})$. The simulator integrates 20 time steps forward with the fourth-order Runge-Kutta method (RK4) to generate trajectories and all the trajectories are rendered into sequences of images.

Model training

There are two important hyperparameters - the prediction time step T_{pred} and the ODE solver used in training. A large prediction time step T_{pred} penalizes inaccurate long term prediction but requires more time to train. In practice, we found that $T_{\text{pred}} = 2, 3, 4, 5$ are able to get reasonably good prediction. In the paper, we present results of models trained with $T_{\text{pred}} = 4$. As for the ODE solver, it is tempting to use RK4 since this is how the training data are generated. However, in practice, using RK4 would make training extremely slow and sometimes the loss would blow up. It is because the operations of RK4 result in a complicated forward pass, especially when we also use a relatively large T_{pred}. Moreover, since we have no access to the state data in the latent space, we penalize the reconstruction error in the image space. The accuracy gained by higher-order ODE solvers in the latent space might not be noticable in the reconstruction error in the image space. Thus, during training, we use an first-order Euler solver. As the Euler solver is inaccurate especially for long term prediction, after training, we could use RK4 instead of Euler for integration to get

better long term prediction results with learned models.

As our data are generated with 20 times steps in each trajectory, we would like to rearrange the data so that each trajectory contains $T_{\text{pred}} + 1$ time steps, as stated in Section 3. In order to utilize the data as much as possible, we rearrange the data $((\tilde{\mathbf{x}}^1, \mathbf{u}^c), (\tilde{\mathbf{x}}^2, \mathbf{u}^c)), ..., (\tilde{\mathbf{x}}^{20}, \mathbf{u}^c))$ into $((\tilde{\mathbf{x}}^i, \mathbf{u}^c), (\tilde{\mathbf{x}}^{i+1}, \mathbf{u}^c), ..., (\tilde{\mathbf{x}}^{i+T_{\text{pred}}}, \mathbf{u}^c))$, where $i = 1, 2, ..., 20 - T_{\text{pred}}$.

For all the experiments, we use the Adam optimizer to train our model.

13.6.3 Ablation study details

We report on the following four ablations:

(a) **tradEncoder + caDecoder**: replacing the coordinate-aware encoder with a traditional black-box MLP

(b) **caEncoder + tradDecoder**: replacing the coordinate-aware decoder with a traditional black-box MLP

(c) **caAE**: replacing the coordinate-aware VAE with a coordinate-aware AE

(d) **PAIG**: a Physics-as-inverse-graphics model

Figure 13.5 shows the prediction sequences of ablations of Pendulum and CartPole. Our proposed model is labelled as caVAE. Since long term prediction of the chaotic Acrobot is not possible, Figure 13.6 shows the reconstruction image sequences of ablations of Acrobot. From the results, we find that PAIG and caAE completely fails in CartPole and Acrobot, although they work well in the simple Pendulum experiment. By replacing the coordinate-aware decoder, caEncoder+tradDecoder fails to reconstruct rigid bodies in CartPole and Acrobot. By replacing the coordinate-aware encoder, tradEncoder+caDecoder reconstructs correct images with well-learned coordinates in Pendulum and Acrobot, but in CartPole, the coordinates are not well-learned, resulting in bad prediction. Thus, we conclude that the coordinate-aware decoder makes the primary contribution to learning interpretable generalized coordinates and getting good reconstruction images, while the coordinate-aware encoder makes a secondary contribution.

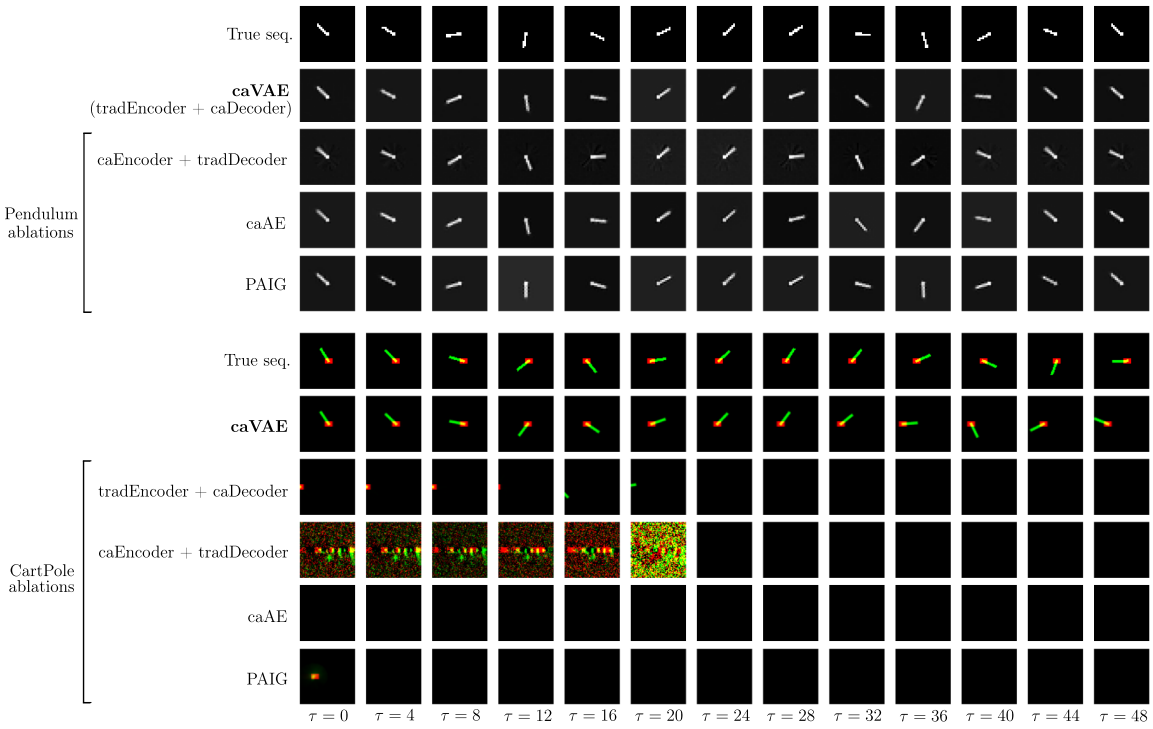

Figure 13.5: Prediction sequences of ablations of Pendulum and CartPole with a previously unseen initial condition and zero control. For the Pendulum experiment, the coordinate-aware encoder is a traditional MLP encoder. All the ablations get good predictions. For the CartPole experiment, all the ablations fail to get good predictions. The *PAIG* is able to reconstruct the cart initially but it fails to reconstruct the pole and make prediction. The *caAE* fails to reconstruct anything. The *caEncoder+tradDecoder* fails to reconstruct meaningful rigid bodies. The *tradEncoder+caDecoder* seems to extract meaningful rigid bodies but it fails to put the rigid bodies in the right place in the image, indicating the coordinates are not well learned.

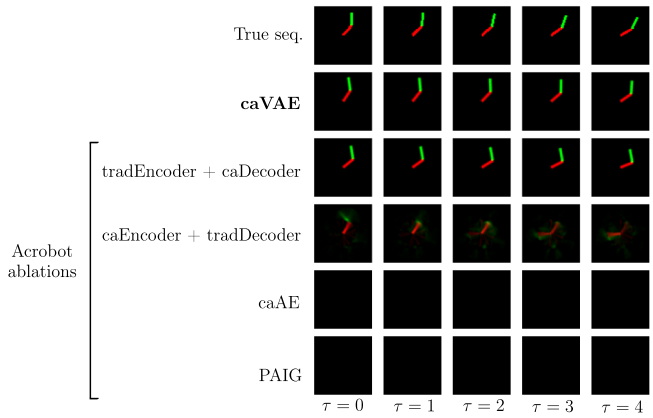

Figure 13.6: Reconstruction image sequences of ablations of Acrobot with a previously unseen initial condition and zero control. The *PAIG* and *caAE* fail to reconstruct anything. The *caEncoder+tradDecoder* fails to reconstruct the green link at all. The *tradEncoder+caDecoder* makes good reconstruction.

Bibliography

[1] D. Acemoglu, A. Ozdaglar, and E. Yildiz. Diffusion of innovations in social networks. In *IEEE Conf. Decision and Control*, pages 2329–2334, 2011.

[2] R. M. Anderson, B. Anderson, and R. M. May. *Infectious diseases of humans: dynamics and control*. Oxford university press, 1992.

[3] C. G. Antonopoulos and Y. Shang. Opinion formation in multiplex networks with general initial distributions. *Scientific Reports*, 8(1), 2018. ISSN 2045-2322. doi: 10.1038/s41598-018-21054-0.

[4] V. I. Arnold, A. B. Givental, and S. P. Novikov. Symplectic geometry. In *Dynamical systems IV*, pages 1–138. Springer, 2001.

[5] I. Ayed, E. de Bézenac, A. Pajot, J. Brajard, and P. Gallinari. Learning dynamical systems from partial observations. *arXiv:1902.11136*, 2019.

[6] P. Battaglia, R. Pascanu, M. Lai, D. Jimenez Rezende, and K. Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In *Advances in Neural Information Processing Systems 29*, pages 4502–4510, 2016.

[7] J. Baxter. A model of inductive bias learning. *Journal of Artificial Intelligence Research*, 12: 149–198, 2000.

[8] F. d. A. Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems 31*, pages 7178–7189, 2018.

[9] A. Bizyaeva, A. Franci, and N. E. Leonard. A general model of opinion dynamics with tunable sensitivity. *arXiv preprint arXiv:2009.04332*, 2020.

[10] A. M. Bloch, N. E. Leonard, and J. E. Marsden. Controlled lagrangians and the stabilization of mechanical systems. i. the first matching theorem. *IEEE Transactions on Automatic Control*, 45(12):2253–2270, 2000.

[11] A. M. Bloch, N. E. Leonard, and J. E. Marsden. Controlled lagrangians and the stabilization of euler–poincaré mechanical systems. *International Journal of Robust and Nonlinear Control*, 11 (3):191–214, 2001.

[12] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014.

[13] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. Openai gym. *arXiv e-prints*, art. arXiv:1606.01540, 2016.

[14] A. Byravan and D. Fox. Se3-nets: Learning rigid body motion using deep neural networks. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 173–180. IEEE, 2017.

[15] S. H. Cen, V. Srivastava, and N. E. Leonard. On robustness and leadership in markov switching consensus networks. In *2017 IEEE 56th Annual Conference on Decision and Control*, pages 1701–1706, 2017.

[16] D. E. Chang, A. M. Bloch, N. E. Leonard, J. E. Marsden, and C. A. Woolsey. The equivalence of controlled lagrangian and controlled hamiltonian systems. *ESAIM: Control, Optimisation and Calculus of Variations*, 8:393–422, 2002.

[17] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems 31*, pages 6571–6583, 2018.

[18] Z. Chen, J. Zhang, M. Arjovsky, and L. Bottou. Symplectic recurrent neural networks. In *International Conference on Learning Representations*, 2020.

[19] G. Como, W. S. Rossi, and F. Fagnani. Threshold models of cascades in large-scale networks. *arXiv preprint arXiv:1604.05490*, 2016.

[20] M. Cranmer, S. Greydanus, S. Hoyer, P. Battaglia, D. Spergel, and S. Ho. Lagrangian neural networks. *arXiv e-prints*, art. arXiv:2003.04630, 2020.

[21] T. R. Davidson, L. Falorsi, N. De Cao, T. Kipf, and J. M. Tomczak. Hyperspherical variational auto-encoders. *34th Conference on Uncertainty in Artificial Intelligence (UAI-18)*, 2018.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, 2019.

[23] B. Dey, A. Franci, K. Özcimder, and N. E. Leonard. Feedback controlled bifurcation of evolutionary dynamics with generalized fitness. In *American Control Conference*, pages 6049–6054, 2018.

[24] M. Fardad and G. Kearney. On a linear programming approach to the optimal seeding of cascading failures. In *IEEE Conf. Decision and Control*, pages 102–107, 2017.

[25] A. Garulli, A. Giannitrapani, and M. Valentini. Analysis of threshold models for collective actions in social networks. In *European Control Conference*, pages 211–216, 2015.

[26] J. M. Glover. *The quaternion Bingham distribution, 3D object detection, and dynamic manipulation*. PhD thesis, Massachusetts Institute of Technology, 2014.

[27] H. Goldstein, C. Poole, and J. Safko. *Classical Mechanics*. American Association of Physics Teachers, 2002.

[28] S. Gomez, A. Diaz-Guilera, J. Gomez-Gardeñes, C. J. Perez-Vicente, Y. Moreno, and A. Arenas. Diffusion dynamics on multiplex networks. *Physical Review Letters*, 110(2), 2013. ISSN 0031-9007, 1079-7114.

[29] I. Goodfellow, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT Press, 2016.

[30] I. J. Goodfellow, J. Shlens, and C. Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[31] M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6): 1420–1443, 1978.

[32] R. Gray, A. Franci, V. Srivastava, and N. E. Leonard. Multiagent decision-making dynamics inspired by honeybees. *IEEE Transactions on Control of Network Systems*, 5(2):793–806, 2018.

[33] S. Greydanus, M. Dzamba, and J. Yosinski. Hamiltonian neural networks. In *Advances in Neural Information Processing Systems 32*, pages 15379–15389, 2019.

[34] J. K. Gupta, K. Menda, Z. Manchester, and M. J. Kochenderfer. A general framework for structured learning of mechanical systems. *arXiv:1902.08705*, 2019.

[35] L. N. Hand and J. D. Finch. *Analytical Mechanics*. Cambridge University Press, 1998.

[36] H. Hanßmann, N. E. Leonard, and T. R. Smith. Symmetry and reduction for coordinated rigid bodies. *European journal of control*, 12(2):176–194, 2006.

[37] D. Haussler. Quantifying inductive bias: AI learning algorithms and Valiant's learning framework. *Artificial Intelligence*, 36(2):177–221, 1988.

[38] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[39] H. W. Hethcote. The mathematics of infectious diseases. *SIAM review*, 42(4):599–653, 2000.

[40] 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.

[41] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. In *Advances in Neural Information Processing Systems 28*, pages 2017–2025, 2015.

[42] M. Jaques, M. Burke, and T. Hospedales. Physics-as-inverse-graphics: Unsupervised physical parameter estimation from video. In *International Conference on Learning Representations*, 2020.

[43] M. Karl, M. Soelch, J. Bayer, and P. van der Smagt. Deep variational bayes filters: Unsupervised learning of state space models from raw data. *arXiv:1605.06432*, 2016.

[44] E. Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv e-prints*, art. arXiv:1801.01450, 2017.

[45] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the spread of influence through a social network. In *Proc. 9th ACM SIGKDD Int. Conf. Knowledge Discovery and Data Mining*, page 137–146, 2003. ISBN 1581137370. doi: 10.1145/956750.956769.

[46] D. P. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980*, 2014.

[47] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *International Conference on Learning Representations*, 2014.

[48] J. Kossen, K. Stelzner, M. Hussing, C. Voelcker, and K. Kersting. Structured object-aware physics prediction for video modeling and planning. In *International Conference on Learning Representations*, 2020.

[49] R. G. Krishnan, U. Shalit, and D. Sontag. Structured inference networks for nonlinear state space models. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[50] T. D. Kulkarni, A. Gupta, C. Ionescu, S. Borgeaud, M. Reynolds, A. Zisserman, and V. Mnih. Unsupervised learning of object keypoints for perception and control. In *Advances in Neural Information Processing Systems 32*, 2019.

[51] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society, Series B*, pages 157–224, 1988.

[52] M. Lelarge. Diffusion and cascading behavior in random networks. *Games and Economic Behavior*, 75(2):752–775, 2012.

[53] N. Levine, Y. Chow, R. Shu, A. Li, M. Ghavamzadeh, and H. Bui. Prediction, consistency, curvature: Representation learning for locally-linear control. In *International Conference on Learning Representations*, 2020.

[54] 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. *arXiv:1509.02971*, 2015.

[55] Y. Lim, A. Ozdaglar, and A. Teytelboym. A simple model of cascades in networks. Technical report, MIT, 2015.

[56] M. Lutter, C. Ritter, and J. Peters. Deep lagrangian networks: Using physics as model prior for deep learning. In *International Conference on Learning Representations*, 2019.

[57] M. Minderer, C. Sun, R. Villegas, F. Cole, K. P. Murphy, and H. Lee. Unsupervised learning of object structure and dynamics from videos. In *Advances in Neural Information Processing Systems 32*, pages 92–102, 2019.

[58] K. P. Murphy, Y. Weiss, and M. I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proc. 15th Conf. Uncertainty in Artificial Intelligence*, pages 467–475, 1999.

[59] S. Nair and N. E. Leonard. Stabilization of a coordinated network of rotating rigid bodies. In *2004 43rd IEEE Conference on Decision and Control (CDC)(IEEE Cat. No. 04CH37601)*, volume 5, pages 4690–4695. IEEE, 2004.

[60] S. Nair and N. E. Leonard. Stable synchronization of rigid body networks. *Networks & Heterogeneous Media*, 2(4):597, 2007.

[61] S. Nair and N. E. Leonard. Stable synchronization of mechanical system networks. *SIAM Journal on Control and Optimization*, 47(2):661–683, 2008.

[62] S. Nair, N. E. Leonard, and L. Moreau. Coordinated control of networked mechanical systems with unstable dynamics. In *42nd IEEE International Conference on Decision and Control (IEEE Cat. No. 03CH37475)*, volume 1, pages 550–555. IEEE, 2003.

[63] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1990.

[64] H. Nguyen and R. Zheng. Influence spread in large-scale social networks–a belief propagation approach. In *Machine Learning and Knowledge Discovery in Databases*, pages 515–530, 2012.

[65] R. Ortega, A. J. Van Der Schaft, I. Mareels, and B. Maschke. Putting energy back in control. *IEEE Control Systems Magazine*, 21(2):18–33, 2001.

[66] R. Ortega, A. J. Van Der Schaft, B. Maschke, and G. Escobar. Interconnection and damping assignment passivity-based control of port-controlled hamiltonian systems. *Automatica*, 38 (4):585–596, 2002.

[67] R. Pagliara and N. E. Leonard. Adaptive susceptibility and heterogeneity in contagion models on networks. *IEEE Transactions on Automatic Control*, 2020.

[68] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[69] F. L. Pinheiro, V. V. Vasconcelos, and S. A. Levin. Consensus and polarization in competing complex contagion processes. *arXiv e-prints*, art. arXiv:1811.08525, Nov 2018.

[70] D. Rosa and A. Giua. A non-progressive model of innovation diffusion in social networks. In *IEEE Conf. Decision and Control*, pages 6202–6207, 2013.

[71] S. B. Rosenthal, C. R. Twomey, A. T. Hartnett, H. S. Wu, and I. D. Couzin. Revealing the hidden networks of interaction in mobile animal groups allows prediction of complex behavioral contagion. *Proceedings of the National Academy of Sciences*, 112(15):4690–4695, 2015.

[72] W. S. Rossi, G. Como, and F. Fagnani. Threshold models of cascades in large-scale networks. *IEEE Transactions on Network Science and Engineering*, 6(2):158–172, 2019.

[73] D. J. Rowe, A. Ryman, and G. Rosensteel. Many-body quantum mechanics as a symplectic dynamical system. *Physical Review A*, 22(6):2362, 1980.

[74] S. Saemundsson, A. Terenin, K. Hofmann, and M. P. Deisenroth. Variational Integrator Networks for Physically Structured Embeddings. *arXiv e-prints*, art. arXiv:1910.09349, 2019.

[75] M. Salehi, R. Sharma, M. Marzolla, M. Magnani, P. Siyari, and D. Montesi. Spreading processes in multilayer networks. *IEEE Transactions on Network Science and Engineering*, 2(2):65–83, 2015. ISSN 2327-4697, 2334-329X. doi: 10.1109/TNSE.2015.2425961.

[76] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, pages 4470–4479, 2018.

[77] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, and P. Battaglia. Graph networks as learnable physics engines for inference and control. In *International Conference on Machine Learning (ICML)*, pages 4467–4476, 2018.

[78] A. Sanchez-Gonzalez, V. Bapst, K. Cranmer, and P. Battaglia. Hamiltonian graph networks with ode integrators. *arXiv e-prints*, art. arXiv:1909.12790, 2019.

[79] A. Sarlette, R. Sepulchre, and N. E. Leonard. Autonomous rigid body attitude synchronization. *Automatica*, 45(2):572–577, 2009.

[80] T. C. Schelling. *Micromotives and Macrobehavior*. Norton, 1978.

[81] H. Shao, Y. Xi, M. Mesbahi, D. Li, Y. Xu, and Z. Gan. Relative tempo of consensus dynamics on multiplex networks. *IFAC-PapersOnLine*, 50(1):5184–5189, July 2017. ISSN 2405-8963. doi: 10.1016/j.ifacol.2017.08.444.

[82] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550 (7676):354–359, 2017.

[83] T. R. Smith, H. Hanßmann, and N. E. Leonard. Orientation control of multiple underwater vehicles with symmetry-breaking potentials. In *Proceedings of the 40th IEEE Conference on Decision and Control*, volume 5, pages 4598–4603, 2001.

[84] T. Söderström and P. Stoica. *System identification*. Prentice-Hall, Inc., 1988.

[85] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27*, pages 3104–3112, 2014.

[86] P. Toth, D. J. Rezende, A. Jaegle, S. Racanière, A. Botev, and I. Higgins. Hamiltonian generative networks. In *International Conference on Learning Representations*, 2020.

[87] I. Trpevski, A. Stanoev, A. Koseska, and L. Kocarev. Discrete-time distributed consensus on multiplex networks. *New Journal of Physics*, 16(11):113063, 2014. ISSN 1367-2630. doi: 10.1088/1367-2630/16/11/113063.

[88] V. V. Vasconcelos, S. A. Levin, and F. L. Pinheiro. Consensus and polarization in competing complex contagion processes. *Journal of the Royal Society Interface*, 16(155):20190196, 2019. doi: 10.1098/rsif.2019.0196.

[89] M. Watter, J. Springenberg, J. Boedecker, and M. Riedmiller. Embed to control: A locally linear latent dynamics model for control from raw images. In *Advances in Neural Information Processing Systems 28*, pages 2746–2754, 2015.

[90] N. Watters, D. Zoran, T. Weber, P. Battaglia, R. Pascanu, and A. Tacchetti. Visual interaction networks: Learning a physics simulator from video. In *Advances in Neural Information Processing Systems 30*, pages 4539–4547, 2017.

[91] N. Watters, L. Matthey, M. Bosnjak, C. P. Burgess, and A. Lerchner. Cobra: Data-efficient model-based rl through unsupervised object discovery and curiosity-driven exploration. *arXiv e-prints*, art. arXiv:1905.09275, 2019.

[92] D. J. Watts. A simple model of global cascades on random networks. *Proceedings of the National Academy of Sciences*, 99(9):5766–5771, 2002.

[93] T. Wei, Y. Wang, and Q. Zhu. Deep Reinforcement Learning for Building HVAC Control. In *Proceedings of the 54th Annual Design Automation Conference (DAC)*, pages 22:1–22:6, 2017.

[94] O. Yağan and V. Gligor. Analysis of complex contagions in random multiplex networks. *Physical Review E*, 86(3):036103, 2012.

[95] L. Yang, A. Giua, and Z. Li. Minimizing the influence propagation in social networks for linear threshold models. *IFAC-PapersOnLine*, 50(1):14465 – 14470, 2017. ISSN 2405-8963. doi: https://doi.org/10.1016/j.ifacol.2017.08.2293.

[96] L. Yang, Z. Yu, M. A. El-Meligy, A. M. El-Sherbeeny, and N. Wu. On multiplexity-aware influence spread in social networks. *IEEE Access*, 8:106705–106713, 2020.

[97] H. P. Young. The dynamics of social innovation. *Proceedings of the National Academy of Sciences*, 108(Supplement 4):21285–21291, 2011.

[98] Y. D. Zhong and N. E. Leonard. A continuous threshold model of cascade dynamics. In *2019 IEEE Conference on Decision and Control*, pages 1704–1709, 2019.

[99] Y. D. Zhong and N. E. Leonard. Unsupervised learning of lagrangian dynamics from images for prediction and control. In *Advances in Neural Information Processing Systems*, 2020.

[100] Y. D. Zhong, V. Srivastava, and N. E. Leonard. On the linear threshold model for diffusion of innovations in multiplex social networks. In *IEEE Conf. Decision and Control*, pages 2593–2598, 2017.

[101] Y. D. Zhong, B. Dey, and A. Chakraborty. Symplectic ode-net: Learning hamiltonian dynamics with control. In *International Conference on Learning Representations*, 2020.

[102] Y. D. Zhong, B. Dey, and A. Chakraborty. Dissipative symoden: Encoding hamiltonian dynamics with dissipation and control into deep learning. In *ICLR 2020 Workshop on Integration of Deep Neural Models and Differential Equations*, 2020.

[103] Y. D. Zhong, V. Srivastava, and N. E. Leonard. Influence spread in the heterogeneous multiplex linear threshold model. *arXiv preprint arXiv:2008.04383*, 2020.