Pacific Northwest
NATIONAL LABORATORY

PNNL-33338

Sequential Decision Making (SDM) for Mesh Refinement and Model Selection in Multiscale, Multi-Physics Applications

March 4, 2022

R Tipireddy
V Amatya
WR Rosenthal
M Subramanian

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
Under contract DE-AC05-76RL01830

DISCLAIMER

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from the
Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062;
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@adonis.osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
email: orders@ntis.gov <https://www.ntis.gov/about>
Online ordering: http://www.ntis.gov

PNNL-33338

Sequential Decision Making (SDM) for Mesh Refinement and Model Selection in Multiscale, Multi-Physics Applications

March 4, 2022

R Tipireddy
M Subramanian
WR Rosenthal
V Amatya

Prepared for
the U.S. Department of Energy
Under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99352

Abstract

Intelligent automation and decision support are needed to enhance computational efficiency and robustness in multiscale and multi-physics problems, including materials science, manufacturing, and climate and weather modeling. Current scientific computing approaches for enabling decisions by scientists fail to explore the role of learning, reasoning, and probabilistic planning. Often these decisions are not performed in real-time during the computation but are made prior to the start of the computation, which must be interrupted in order to make changes to the prior choices. Such interruptions at different stages of the computation increase the total computing time and the need for a human expert to frequently monitor the results. State of art scientific computing methods consist of rule-based algorithms that cannot automatically adapt to a dynamically changing computing environment. The development of a Sequential Decision Making (SDM) framework will automate scientific computing by optimizing the policies for mesh refinement, time-stepping, model and algorithm selection, resource allocation, and pre and post-processing. Our agent SDM framework for scientific computing will consist of data-driven learning (Classifier), automated reasoning (contextual knowledge), and probabilistic planning (Reinforcement Learning). In this project, we focused on three problems to demonstrate our SDM framework on a set of ordinary and partial differential equations. Classification of Lorenz system regions using Feed-Forward Neural Networks examined learning in the SDM framework. On the other hand, reasoning and planning in the SDM framework were used in two problems: adaptive time-stepping for nonlinear ODEs using on-policy RL algorithms, and adaptive mesh refinement for 2-D PDEs using off-policy RL algorithms.

Executive Summary

Development of a Sequential Decision Making (SDM) framework will automate scientific computing by optimizing the policies for mesh refinement, time-stepping, model and algorithm selection, resource allocation, and pre and post processing. Our multi-agent SDM framework for scientific computing will consist of data-driven learning (Classifier), automated reasoning (contextual knowledge) and probabilistic planning (POMDP/RL). The data driven learning will analyze the problem environment to evaluate the systems state. Automated reasoning consists of probabilistic, physics-based or contextual knowledge reasoning about stochastic multiscale and multi-physics problems. This element will eliminate the non-feasible systems states using contextual knowledge reasoning. Finally, the probabilistic planning is the main decision making element which is based on both model-based and model-free, multi-agent reinforcement learning algorithms. In this approach different scientific computing agents will work in a cooperative manner to achieve common goal of obtaining efficient, accurate and stable numerical solution. Each agent will have a specific role such as adaptively choosing the optimal time step, adaptive mesh refinement, model selection, optimal resource allocation.

Although Artificial Intelligence (AI) and Machine Learning (ML) methods have proven to be beneficial in many fields such as robotics, optimal control etc., these techniques have not completely been utilized in the scientific computing applications. Model-based and model-free multi-agent reinforcement learning algorithms developed in this project will enhance the existing numerical methods for their numerical stability, accuracy and computational efficiency. Algorithms developed in this project can be easily adapted to efficiently compute the numerical solutions to mathematical models of systems with relevance to the DOE missions.

Automated decision making in complex systems relies on robust system state identification under uncertainty. System state identification via data-driven learning is the often the first step within a sequential decision making (SDM) framework [2]. An SDM framework may comprise of multiple computational modules for system state estimation, context-aware reasoning, and probabilistic planning. A computational agent in such settings may continuously interact with an external environment, compute the best policies (state to action mappings), and execute actions that maximize learning reward signals in the long run [19]. In this context, system state is often hidden or partially observable from an agent. This requires an agent to infer the state of the system indirectly through a combination of data driven learning and domain knowledge.

Acknowledgments

The research described in this paper is supported by the Mathematics of Artificial Reasoning in Science (MARS) Initiative at Pacific Northwest National Laboratory (PNNL). It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy under DE-AC05-76RL01830.

Contents

Figures

Tables

1.0 Introduction

In this project, we developed a sequential decision making (SDM) framework to automate the process of obtaining optimal solutions to stochastic multiscale and multi-physics problems, including method selection, workload balancing, and the use of quality control heuristics. Computing the solution of large-scale problems with multiscale processes, such as fluid flow and heat transfer in a reactor core, is computationally expensive and involves iterative stages of preprocessing, high performance computing, and post-processing. In the existing multi-physics application frameworks [1], human intervention is required due to the stochasticity of the decision variables that need to be analyzed and parameterized. Such decision variables can influence various phases of the application framework, including pre-processing, core computation, and result validation. Examples of the types of decision choices required are: real-time mesh refinement and model or algorithm selection in a dynamically changing computing environment; efficient exchange of parameters and information between coarse scale and fine scale physics models; and allocation of optimal computing resources. Often these decisions are not performed in real time during the computation but are made prior to the start of the computation, which must be interrupted in order to make changes to the prior choices. Such interruptions at different stages of the computation increases the total computing time and need for a human expert to frequently monitor the results. In this one-year project, we focused on three specific problems: a) classification of Lorenz system regions using Feed-Forward Neural Networks, examining learning in the SDM framework; b) adaptive time stepping for nonlinear ODEs using on-policy RL algorithms for reasoning and planning in an SDM framework; and c) adaptive mesh refinement for 2-D PDEs using off-policy RL algorithms, looking at reasoning and planning in SDM frameworks. We demonstrated our SDM framework on a set of ordinary and partial differential equations.

This report is organized as follows: In Section 2.0, we introduce the sequential decision making (SDM) framework. We introduce stability identification of a Lorenz system of equations as an example of learning in the SDM framework in Section 3.0. We provide an adaptive time stepping method for nonlinear ODEs using on-policy RL algorithms in Section 4.0 and present an adaptive mesh refinement approach for 2-D elliptic equations using off-policy RL algorithms in Section 5.0. Then, we demonstrate our SDM framework with numerical experiments in Section 6.0. Finally, we end the report with conclusions in Section 7.0.

2.0 Sequential Decision Making

In this section we present sequential decision making based on data driven learning, contextual reasoning and probabilistic planning. In particular we introduce a neural network based classification method for stability identification in a Lorenz system of equations, and describe on-policy and off-policy reinforcement learning algorithms for adaptive time stepping and adaptive mesh refinement for ordinary and partial differential equations, respectively. Fig. 1 shows our SDM framework based on learning, reasoning, and planning tasks.

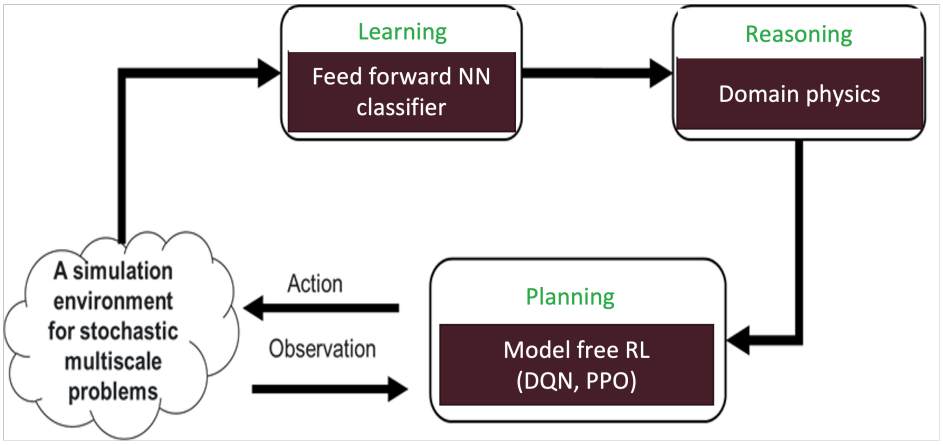


Figure 1: Sequential decision making.

2.1 Learning-based classification

Neural networks are a sequence of densely interconnected nodes. Through a process known as training, the networks learn the parameters, or the relationship between these nodes. The availability of large amounts of data and massive computational power are some of the reasons that have facilitated training of deep neural networks in recent years. Consequently, deep neural networks have garnered widespread attention across different scientific disciplines including speech recognition [7], image classification [6], language modeling [20] and even protein folding [17]. The idea of leveraging neural networks to study dynamical systems has been explored for many years [14]. Recently, both feed-forward and recurrent architectures have been used for several forecasting and prediction based tasks on chaotic systems. For example, feed-forward neural networks have been shown to predict extreme events in Hénon map [11] and LSTM [8] architectures have been used for forecasting high-dimensional chaotic systems [23]. Additionally, Reservoir Computing-based approaches have been leveraged for data-driven prediction of chaotic systems. There is also literature suggesting that a hybrid approach to forecasting that involves both machine learning and knowledge based models [15] leads to higher prediction accuracy for a longer period on chaotic systems. Another line of work involves discovering underlying models from data, wherein auto-encoder network architectures are used to recognize the coordinate transformation and the governing equations of a dynamical system [4]. All these examples highlight the general feasibility of using machine learning and neural networks to study a wide variety of tasks involving chaotic systems. Previous work has shown that feed-forward neural networks are reasonable candidates for predicting the timing and duration of oscillations between attractors in Lorenz63 systems [3].

Inspired by this approach, we leverage feed-forward neural networks for our novel task of classifying the data points of a Lorenz system. The solution of the Lorenz system consists of two regimes which we refer to as the left and right regimes. Unlike the approach proposed in [3], we do not restrict ourselves to the boundary $x = 0$ to distinguish between the left and right regimes. Since we are interested in studying a broader range of Lorenz systems, we adopt the mean value of x-coordinates as a boundary to distinguish between the left and right regimes.

One common problem associated with neural network architectures is a lack of generalization to validation data that is drawn from a different distribution as compared to the data used for training. This mismatch between training and validation distributions, also referred to as covariate shift [18] is a well-studied aspect in most of the scientific fields that have benefited from advances in neural network architectures. For example, within the speech recognition community, this issue is referred to as acoustic mismatch and is a critical factor that affects the deployment of speech recognition systems in real world environments [21]. A similar terminology used in the field of natural language processing is domain mismatch, which renders tasks like cross-lingual document classification especially challenging [10]. In the context of chaotic systems, there has been some published work [16] that discusses the generalization capabilities of neural networks trained on Lorenz systems. One of the results of this work on Lorenz63 systems is that neural networks trained only on a part of the system's phase space struggle to skillfully forecast in regions that are excluded from the training phase space. These observations are consistent with our initial results in Section 6.3, where we notice and analyze the drop in performance of neural networks in the mismatched case.

In this project, we further show how a very simple yet effective normalization approach helps in counteracting the mismatch problem. As we will show in Section 6.4, this pre-processing step of normalization helps in training neural networks to perform well on a wide variety of validation data for the task of classification of stable/unstable data points. To the best of our knowledge, neural networks that classify data points of Lorenz system as stable or unstable, performance analysis in the mismatched case, as well as the proposed normalization scheme, have not been examined in the literature.

2.2 MDP and Deep Reinforcement Learning

In this section we present Markov decision process and describe various on-policy and off-policy reinforcement algorithms. Reinforcement learning (RL) can be described as a tuple $\langle S, A, R, P, p_0 \rangle$ such that

- $S = \{s_1, \cdots, s_n\}$ is a set of all valid states,
- $A = \{a_1, \cdots, a_n\}$ is a set of all valid actions,
- $R : S \times A \times S \to \mathbb{R}$ is the reward function $r_t = R(s_t, a_t, s_{t+1})$,
- $P : S \times A \to \mathcal{P}(\mathcal{S}$ is the transition probability such that $P(s'|a, s)$ is the probability of transitioning from state s to state s' after taking action a, and
- $p_0(s_0)$ is the probability distribution of the initial state.

In reinforcement learning [19], the decision maker or "agent" continuously interacts with an external environment by selecting a set of actions. The environment responds to these actions with changes in its state and offers an instantaneous reward. The objective of the agent is to learn about the environment while interacting with it and maximize the long-term reward. The reinforcement learning method can be described as a Markov Decision Process (MDP). It

consists of: a set of states $\mathcal{S}$; initial state distribution $p(s_0)$; a set of actions $\mathcal{A}$; dynamics describing the state transition $\mathcal{T}(s_{t+1}|s_t, a_t)$ that maps state-action pairs (s_t, a_t) at time t to the state distribution at time $t+1$; a reward function $\mathcal{R}(s_t, a_t, s_{t+1})$ that gives an instantaneous reward value when the state transitions from s_t to s_{t+1} upon executing the action a_t; and a discount factor $\gamma \in [0, 1]$ which discounts the value of future rewards at the current time.

2.3 Deep Q-Learning

The Deep Q-Network (DQN) was developed by Deep Mind in 2015 to solve a wide range of Atari games. A DQN combines the RL algorithm Q-learning with deep neural networks along with experience replay.

2.3.1 Q-learning

Q-learning is based on a state-action value function called a Q-function $Q^{\pi}(s, a)$, which is defined on a policy π. The Q-function measures the expected sum of discounted rewards by taking action a when in state s while following the policy π. The optimal Q-function obeys the Bellman equation given by

$$Q*(s, a) = \mathbb{E}[r + \gamma \max_{a'}(Q*(s', a'))] \tag{2.3.1}$$

In Q-learning, the Bellman equation is solved iteratively to obtain the optimal Q-function, which can be expressed as

$$Q_{i+1}(s, a) = \mathbb{E}[r + \gamma \max_{a'}(Q_i(s', a'))] \tag{2.3.2}$$

and $Q_i \to Q^*$ as $i \to \infty$.

2.3.2 Deep Q-Network (DQN)

As the number of states in the state space and actions in the action space increase, the problem becomes intractable to solve the Bellman equation using traditional Q-learning methods such as a Q-table. Recent developments in deep neural networks (DNN), such as function approximators in a variety of applications, are attractive tools to model a Q-function. Let the Q-function be parameterized by θ. The Q-network $Q(s, a, \theta)$ modeled with DNNs can be trained by minimizing the loss function,

$$L_i(\theta_i) = \mathbb{E}_{s,a,s' \approx \rho(\cdot)}[(y_i - Q(s, a, \theta_i))^2], \tag{2.3.3}$$

where $y_i = r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$ is the temporal difference (TD) and $y_i - Q$ is the TD error. Q-learning is an off-policy algorithm that learns about the greedy policy $a = \max_a Q(s, a; \theta)$.

2.4 On-policy and off-policy RL

In this section, we introduce several types of reinforcement learning algorithms which utilize policy gradient and Q-learning.

Algorithm 1 DQN algorithm [13]

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights
for episode = $1, M$ **do**
    Initialize sequence $s_1 = \{s_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ **do**
        With probability ϵ select a random action a_t
        Otherwise select a random action $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action a_t in emulator and observe reward r_t and state s_{t+1}
        Set $s_{t+1} = s_t$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in D
        Sample random minibatch of transitions $\phi_j, a_j, r_j, \phi_{j+1}$ from D
        Set $y_j = \begin{cases} r_j, & \text{for terminal} \phi_{j+1} \\ r_j + \gamma Q(\phi_{j+1}, a', ; \theta), & \text{for non-terminal} \phi_{j+1} \end{cases}$
        Perform gradient descent step on $y_j - Q(\phi_j, a_j; \theta)^2$

2.4.1 Vanilla Policy Gradient

In Vanilla Policy Optimization (VPG), also known as the REINFORCE algorithm, the policy π_θ parameterized by θ is optimized using a policy gradient algorithm. The parameters at step $k+1$ are updated using

$$\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\pi_{\theta_k}) \tag{2.4.4}$$

where $J(\pi_{\theta_k})$ is the finite-horizon undercounted return of the policy.

2.4.2 Exploration versus exploitation

VPG trains a stochastic policy in an on-policy way. This means that it explores the action space by sampling actions according to the latest version of its stochastic policy. The amount of randomness in action selection depends on both initial conditions and the training procedure. Over the course of training, the policy typically becomes progressively less random, as the update rule encourages it to exploit rewards that it has already found. This may cause the policy to get trapped in local optima.

2.4.3 Proximal Policy Optimization

PPO-Penalty approximately solves a KL-constrained update and penalizes the KL-divergence in the objective function, instead of making it a hard constraint. It automatically adjusts the penalty coefficient over the course of training so that it is scaled appropriately. IN contrast, PPO-Clip does not have a KL-divergence term in the objective and does not have a constraint at all. Instead, it relies on specialized clipping in the objective function to remove incentives for the new policy to drift far from the old policy. Here, we focus only on PPO-Clip (the primary variant used at OpenAI).

PPO-clip updates policies via

$$\theta_{k+1} = \arg\max_{\theta} \mathop{\mathbb{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)], \tag{2.4.5}$$

where

$$L(s, a, \theta_k, \theta) = \min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A^{\pi_{\theta_k}}(s, a), g(\epsilon, A^{\pi_{\theta_k}}(s, a))\right), \tag{2.4.6}$$

and

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A, & A \geq 0 \\ (1-\epsilon)A, & A < 0 \end{cases} \tag{2.4.7}$$

3.0 Lorenz System State Stability Identification using Neural Networks

In this section, we demonstrate the use of neural networks to classify data points of a Lorenz system as stable/unstable, analyzing their performance in the mismatched case, and proposed a novel normalization scheme.

3.1 Lorenz system of equations

The Lorenz-63 system (which we will simply term a Lorenz system, hereafter) is a dynamical system consisting of three coupled differential equations,

$$\frac{dx}{dt} = \sigma(y - x), \tag{3.1.1a}$$

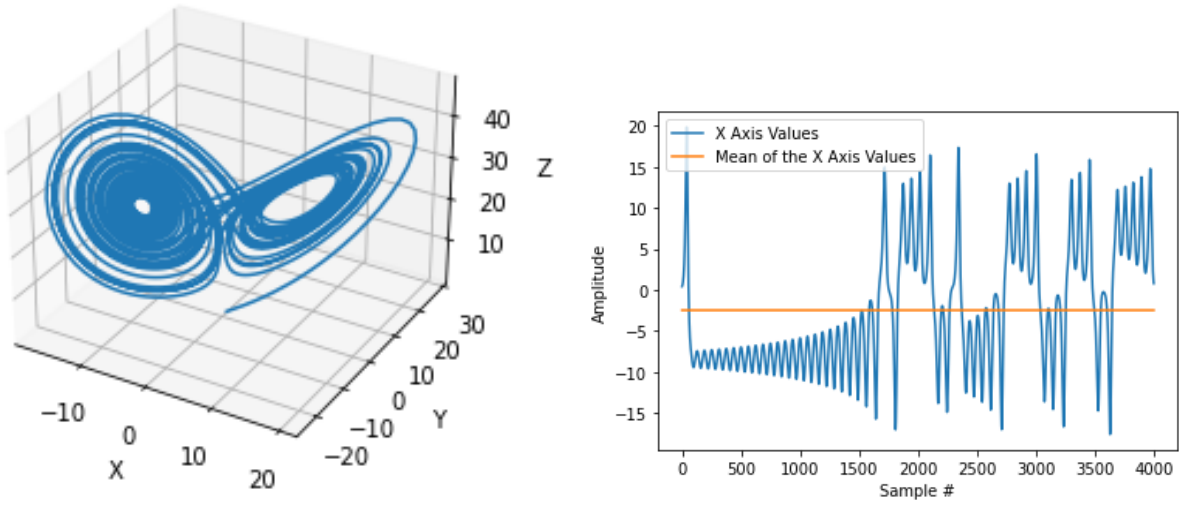
$$\frac{dy}{dt} = x(\rho - z) - y, \tag{3.1.1b}$$

$$\frac{dz}{dt} = xy - \beta z, \tag{3.1.1c}$$

where x, y, and z are the state variables, and σ, ρ, and β are the parameters. In this work, we set $\sigma = 10$, $\rho = 28$, and $\beta = \frac{8}{3}$ [12]. We represent the time derivatives by $\frac{dx}{dt}$, $\frac{dy}{dt}$, and $\frac{dz}{dt}$, respectively, and x_0, y_0 and z_0 are the initial conditions of the Lorenz system in the x, y, and z directions, respectively.

Fig. 2a shows the trajectory of data points of the system state of a typical Lorenz system. The Lorenz system is a highly chaotic system. Even a small perturbation in the initial conditions can cause the trajectories to diverge significantly. The system is characterized by two distinctive regimes that give rise to a butterfly-shaped figure. Starting from similar initial conditions, the data points of a Lorenz system can either remain in the same regime or can alternate between the two regimes. In this work, we are interested in identifying those data points of the Lorenz system that will either undergo a regime change in the future time steps or would have undergone a regime change in the past few time steps. In other words, we are interested in isolating those data points whose past or future data points lie in different regimes. We refer to such data points as unstable data points. We train a neural network to distinguish between stable and unstable data points based on labeled training examples of Lorenz systems. We then test the ability of the neural network models to identify the unstable data points on unseen validation data. Since the Lorenz system is a highly chaotic system, even a small change in the initial conditions will cause a major shift in the location of stable and unstable data points. Consequently, it is a challenging task for neural network models to isolate the unstable data points on unseen validation data.

In this work, we use neural network models to classify unstable data points in both matched (i.e when the initial conditions for training and validation data are sampled from the same interval) and mismatched conditions (i.e when the initial conditions for training and validation data are sampled from different intervals). We first demonstrate that it is particularly difficult for neural network models to reliably classify the unstable data points in mismatched conditions. We further observe that certain normalization schemes can greatly improve the performance of neural network models in mismatched conditions.



(a) A typical Lorenz system.

(b) x-axis values and their mean.

Figure 2: A typical Lorenz system simulation

3.2 Labeling strategy

The classification of data points as stable or unstable can be posed as a supervised learning problem that requires labeled data points during training. We use a two-step process to generate the labels for stable/unstable classification. In the first step, we introduce a heuristic method to identify the right and left regimes of the Lorenz attractor. In the second step, we label the data points as stable or unstable based on the regimes identified in the first step.

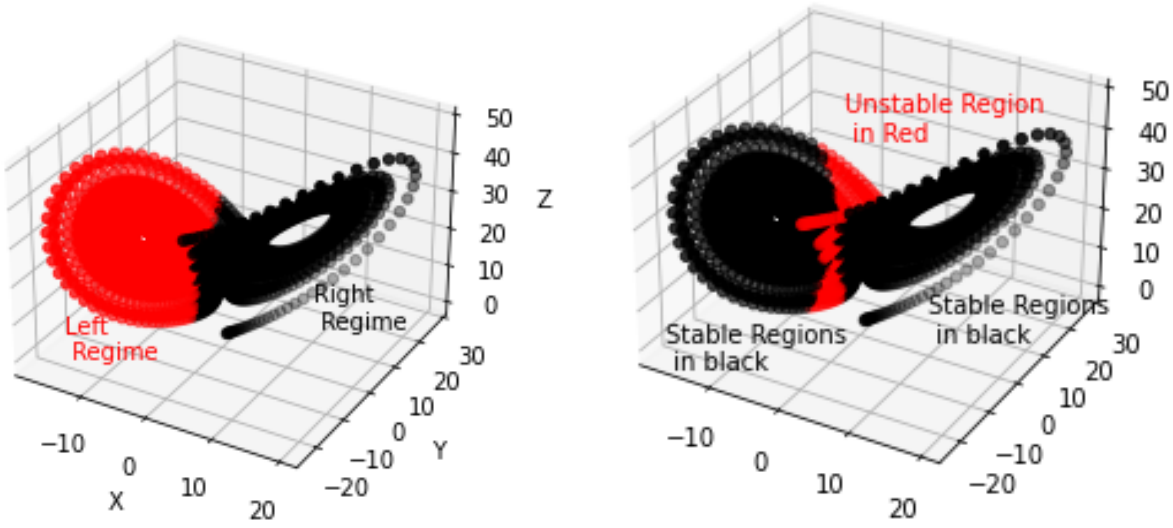
3.2.1 Identification of left and right regimes

In [3], the authors sampled the initial conditions for the dynamic variables from the interval $[0, 1]$ for their experiments. They used the condition $x < 0$ to define the left regimes and $x \geq 0$ to define the right regimes. Since we want to apply the results of our classification to a wide variety of Lorenz systems which may not be perfectly aligned with the boundary $x = 0$, we adopt a different approach. Specifically, we define the left and right regimes of the Lorenz attractor based on the mean value of x-coordinate. The Fig. 2b shows the x-axis values and their mean.

In our heuristic-approach, if the x-coordinate of the data point has a value greater than the mean in the x-direction ($x > mean(x)$), then the data point belongs to the right regime. Otherwise, it belongs to the left regime ($x < mean(x)$). Fig. 3a shows a Lorenz system with left and right regimes labeled using this approach.

3.3 Labeling the data points as stable or unstable

After defining the left and right regimes, we label the data points at every time step t as stable or unstable. Specifically, we consider a window around the data point at time step t. In this study, we work with a window of 5 data points from the past time steps $t-1$, $t-2$, $t-3$, $t-4$, $t-5$ and 5 data points from the future time steps $t+1$, $t+2$, $t+3$, $t+4$, $t+5$. A data point at time



(a) Left and right regimes.

(b) Stable and unstable regions.

Figure 3: Lorenz system labeling. a) A Lorenz system with labeled left and right regimes. The regimes are labeled based on the mean value of the x-coordinate. b) A Lorenz system with stable and unstable regions.

step t is considered stable if the past 5 and the future 5 neighboring data points belong to the same regime. The data point is labeled unstable even if one of the 10 neighbors belongs to the different regime. Here the number of points is a hyper-parameter that we chose as based on trial and error to manually classify the regions into stable and unstable. This number can be calibrated in advance for different dynamical systems. Fig. 3b shows a Lorenz system with labeled stable and unstable data points.

3.4 Training and validation datasets

For generating a Lorenz System such as the one shown in Fig. 2a, we specify the initial condition x_0, y_0 and z_0 for the dynamic variables. We then integrate the system with a time step of 0.01 using ODEINT [1] and the Runge-Kutta 4-5 (RK45) algorithm provided by scipy [22]. For every Lorenz System, we generate 4000 data points. For training, we use 25 such Lorenz systems, resulting in a total of $100,000$ data points for the training set. For validation, we use 5 such Lorenz systems, resulting in a total of $20,000$ data points for the validation set.

3.5 Initial conditions for training dataset

We experiment with three different initial condition intervals for the training data set. The Eqs. (3.5.2) describe the intervals from which the initial condition values for the training data are sampled randomly. It is important to remember that each of the initial conditions x_0, y_0 and z_0 are sampled independently from the choice of intervals.

$$x_0, y_0, z_0 \sim [0, 1] \tag{3.5.2a}$$

$$x_0, y_0, z_0 \sim [-1, 0] \tag{3.5.2b}$$

$$x_0, y_0, z_0 \sim [-1, 1] \tag{3.5.2c}$$

For each of the initial conditions shown in Eqs. (3.5.2), we generate 25 Lorenz systems according to the procedure described above. These Lorenz systems are then used to train different neural networks. We are interested in evaluating whether a neural network trained on the chaotic Lorenz System with the initial condition intervals specified in Eqs. (3.5.2) can achieve accurate classification performance on validation sets generated from a wide variety of initial conditions.

3.6 Initial conditions for the validation dataset

Eqs. (3.6.3) describe the intervals from which the initial condition values for the validation data are sampled randomly for our experiments. We use different random initial seeds while generating the initial conditions for training and validation data. This ensures that the randomly sampled initial conditions for training and validation data are different, even though the intervals from which these are sampled might be the same.

$$x_0, y_0, z_0 \sim [0, 1] \tag{3.6.3a}$$

$$x_0, y_0, z_0 \sim [-1, 0] \tag{3.6.3b}$$

$$x_0, y_0, z_0 \sim [-1, 1] \tag{3.6.3c}$$

$$x_0, y_0, z_0 \sim [2, 4] \tag{3.6.3d}$$

$$x_0, y_0, z_0 \sim [0, 10] \tag{3.6.3e}$$

$$x_0, y_0, z_0 \sim [-10, 10] \tag{3.6.3f}$$

It is particularly challenging for neural networks to accurately classify the stable and unstable points when the initial conditions for training and validation data are sampled from different intervals. We will also see how the normalization scheme discussed in Section 6.4 can help improve the performance in these cases.

3.7 Neural network architecture

In this section, we describe the neural network architecture used to classify the data points of the Lorenz System (See Fig. 4). The neural network takes as input the three Lorenz variables x, y, and z of the data point, as well as the three time derivatives $\frac{dx}{dt}$, $\frac{dy}{dt}$, and $\frac{dz}{dt}$. Thus, the input to the network is a 6-dimensional feature vector. The neural network outputs the probabilities of the input feature vector being stable or unstable. The network consists of four fully connected layers. The first two layers have 512 neurons each and have hyperbolic tangent and rectified linear unit (ReLU) activation functions, respectively. The third layer consists of 256 neurons and ReLU activation functions. The final layer consists of 2 neurons and sigmoid activation function that predicts the probabilities of the data points belonging to either the stable or unstable classes. We use an Adam optimizer [9] with a learning rate of 0.001 and a binary cross-entropy loss function. The neural network was implemented using the Keras library [5].

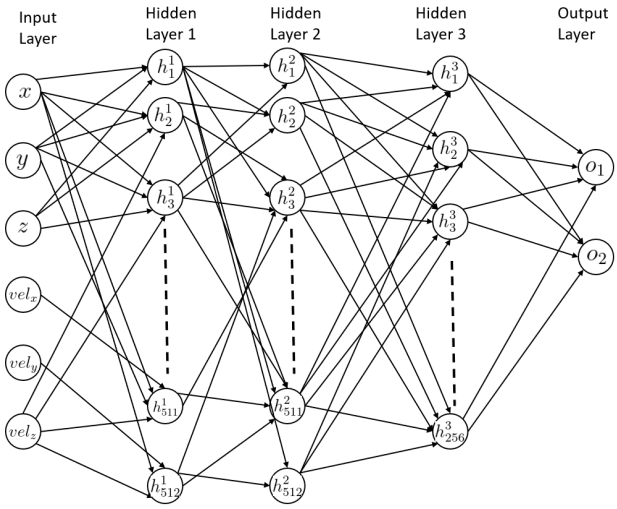


Figure 4: Schematic of the fully-connected feed-forward neural network architecture used in the experiments. Note that h_m^n denotes the m^{th} hidden node in n^{th} layer.

4.0 Adaptive Time Stepping for Nonlinear ODEs Using On-policy RL Algorithms

In this section we describe the numerical methods used to solve each Lorenz System and generate its data points. Runge-Kutta methods are a family of methods used to solve initial-value problems for ordinary differential equations such as

$$\begin{aligned} \dot{y}(t) &= f(t, y) \\ y(t_0) &= y_0. \end{aligned} \tag{4.0.1}$$

A fourth-order Runge-Kutta method (rk4) with fixed time step h can be written as

$$\begin{aligned} y_{n+1} &= y_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= hf(t_n, y_n) \\ k_2 &= hf\left(t_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \\ k_3 &= hf\left(t_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \\ k_4 &= hf(t_n + h, y_n + k_3) \end{aligned} \tag{4.0.2}$$

Adaptive or embedded Runge-Kutta algorithms such as the Dormand-Prince algorithm (rk45) can be used to adaptively change the time step of the algorithm during the integration to improve computational efficiency. The Dormand-Prince algorithm can be written as

$$y_{n+1} = y_n + \frac{35}{384}k_1 + \frac{500}{1113}k_3 + \frac{125}{192}k_4 - \frac{2187}{6784}k_5 + \frac{11}{84}k_6 \tag{4.0.3}$$

and

$$z_{n+1} = y_n + \frac{5179}{57600}k_1 + \frac{7571}{16695}k_3 + \frac{393}{640}k_4 - \frac{92097}{339200}k_5 + \frac{187}{2100}k_6 + \frac{1}{40}k_7, \tag{4.0.4}$$

where y_{n+1} and z_{n+1} are 4^{th} and 5^{th} order solutions respectively and the values of $k_1, \cdots, k_7$ are given by

$$\begin{aligned} k_1 &= hf(t_n, y_n) \\ k_2 &= hf\left(t_n + \frac{h}{5}, y_n + \frac{k_1}{5}\right) \\ k_3 &= hf\left(t_n + \frac{3h}{10}, y_n + \frac{3k_1}{40} + \frac{9k_2}{40}\right) \\ k_4 &= hf\left(t_n + \frac{4h}{5}, y_n + \frac{44k_1}{45} - \frac{56k_2}{15} + \frac{32k_3}{9}\right) \\ k_5 &= hf\left(t_n + \frac{8h}{9}, y_n + \frac{19372k_1}{6561} - \frac{25360k_2}{2187} + \frac{64448k_3}{6561} - \frac{212k_4}{729}\right) \\ k_6 &= hf\left(t_n + h, y_n + \frac{9017k_1}{3168} - \frac{355k_2}{33} + \frac{46732k_3}{5247} + \frac{49k_4}{176} - \frac{5103k_5}{18656}\right) \\ k_7 &= hf\left(t_n + h, y_n + \frac{35k_1}{384} + \frac{500k_3}{1113} + \frac{125k_3}{192} - \frac{2187k_4}{6784} + \frac{11k_5}{84}\right) \end{aligned}$$

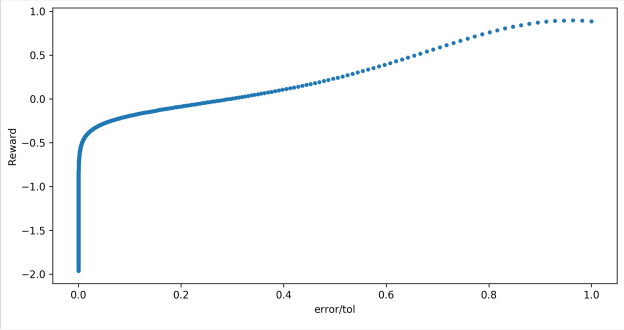


Figure 5: An empirical reward function for reinforcement learning that enables the efficient adaptive time stepping of the Runge-Kutta 4-5 algorithm

The adaptive algorithm (rk45) simultaneously computes the 4^{th} and 5^{th}-order solutions. The local error is given by

$$error = |z_{n+1} - y_{n+1}| \tag{4.0.5}$$

and the optimal time step h_{opt} at time $t + 1$ is computed as

$$h_{opt} = h \left(\frac{\epsilon h}{2error} \right)^{\frac{1}{5}}, \tag{4.0.6}$$

where ϵ is the desired tolerance. In this work, we used on-policy RL algorithms such as proximal policy optimization techniques to optimize the time step of the $rk45$ algorithm. To implement the RK algorithms we define Markov decision process (MDP) tuples as: states $S = h, k_1, \cdots, k_7$, action $A = dt_0, \cdots, dt_n$ and the empirical reward function R as

$$R = \begin{cases} 10^{(\cosh(eq-0.95)^5-1)} + \log 10(eq^{0.25}); & eq < 1 \\ -2, & eq \geq 1, \end{cases} \tag{4.0.7}$$

where $eq = error/\epsilon$. Here the reward function R is plotted as a function of eq in Fig. 5. It shows that the reward is greater when the ratio eq is less than 1 and close to 1. Such a reward function pushes the time step up toward an optimally large size without exceeding the error tolerance.

5.0 Adaptive Mesh Refinement for 2-D PDEs Using Off-policy RL Algorithms

Over the decades, adaptivity has been a well-established tool used to improve the resolution of rough solutions. Since analytic solutions to multiphysics problems are rarely available, one has to resort to numerical solution techniques. In order to obtain a ”good” approximation of the true solution, a rather fine discretization mesh appears to be necessary. This leads to extremely large scale problems on the discrete level which pose many challenges to numerical procedures. The goal of mesh adaptation is to coarsen the mesh in subdomains where the (continuous) solution and data of the problem is ”calm”, and to use a fine mesh only in regions where we expect nonlinearity or non-smoothness over a significant portion of the solution or data. In a sense, one wishes to minimize the number of nodes in a finite element discretization of the problem under the restriction to maintain solution accuracy. To achieve this goal, error estimates and local error indicators are required in order to guide that adaptation process. In recent years, automatic adaptivity has become an area of interest in the field of applied mathematics. The convergence of the adaptive solution process, as well as its quasi-optimality in terms of its computational complexity, with respect to a properly specified approximation class has been well-established. High quality software implementations are available to test the algorithms and validate the analysis.

5.1 Model problem

Let $\Omega \subset \mathbb{R}^2$ be a bounded polygonal domain having boundary Γ consisting of disjoint parts Γ_D and Γ_N, with Γ_D being a closed set with respect to Γ. We consider the following model problem:

$$-\nabla \cdot a\nabla u = f \text{ in } \Omega, \tag{5.1.1a}$$

$$\frac{\partial u}{\partial n} = 0 \text{ on } \Gamma_N, \tag{5.1.1b}$$

$$u = 0 \text{ on } \Gamma_D, \tag{5.1.1c}$$

where a is a non-negative real-valued bounded function on Ω, and f and g_N are assumed to be square-integrable functions on Ω and Γ_N, i.e., they live in $L^2(\Omega)$ and $L^2(\Gamma_N)$, respectively.

5.2 Notation

Here and in the sequel, we employ the standard notation for the well-known Sobolev space $H^1(\Omega)$, which is the space of all square integrable functions admitting L^2 first-order weak derivatives. Let $(\cdot,\cdot)_S$ and $\|\cdot\|_S$ denote the L^2-inner product and norm on a set S, respectively, and the subscript S is omitted when $S = \Omega$. Let $\mathscr{T}_h$ be a geometrically conforming shape-regular triangulation of $\bar{\Omega}$. Associated with $\mathscr{T}_h$, we introduce the following notation:

- h_K = diameter of cell K ($h = \max_{K\in\mathscr{T}_h} h_K$),
- $|K|$ = area of the cell K,
- $\mathscr{E}_h^i$ = the set of all the edges of the interior cells in $\mathscr{T}_h$
- $\mathscr{E}_h^D$ = the set of all the edges of the cells in $\mathscr{T}_h$ excluding edges which intersect the boundary Γ_N,

- $\mathscr{E}_h^N$ = the set of all the edges of the cells in $\mathscr{T}_h$ which intersect the boundary Γ_N,
- e = the edge of a cell.
- h_e = length of an edge.
- $\mathcal{P}_N(K)$ is the space of polynomials on cell K having degree at most N.
- $(u,v)_S$ is the L^2-inner product of u and v on $S \subset \mathbb{R}^d$, $d = 1, 2$.
- $(u,v)_{\mathscr{T}_h} = \sum_{T \in \mathscr{T}_h} (\nabla u, \nabla v)_T$ and $\|u\|^2_{\mathscr{T}_h} = (u,u)_{\mathscr{T}_h}$.

5.3 Finite element approximation

5.3.1 Weak formulation

Find $u \in V$ such that

$$a(u,v) = \ell(v), \quad v \in V. \tag{5.3.2}$$

where

$$a(u,v) := \int_\Omega \nabla u \cdot \nabla v \, dx, \quad \ell(v) = \int_\Omega fv \, dx - \int_{\Gamma_N} g_N v \, ds.$$

Thanks to the Lax-Milgram Lemma, a unique solution exists for (5.3.2). The conforming finite element space of order N is defined by

$$V_h := \{v_h \in V \mid v_h|_K \in \mathcal{P}_N(K), K \in \mathscr{T}_h\},$$

where $u_h \in V_h$, satisfying

$$a(u_h, v_h) = \ell(v_h) \quad \forall v_h \in V_h, \tag{5.3.3}$$

The well-posedness holds due to the Lax-Milgram Lemma.

We introduce a gradient-recovery-type error estimator based on the Zienkiewicz-Zhu (ZZ) estimator. It is defined as the L^2-norm of the difference between the recovered and the numerical gradients, where the recovered gradient is an L^2-projection of the numerical gradient in an $H(div)$-conforming space. On any $T \in \mathscr{T}_h$, let $RT_{N-1}(T)$ be the Raviart-Thomas space of index $N-1$ ($N \geq 1$), i.e.,

$$RT_{N-1}(T) = P_{N-1}(T)^2 + \mathbf{x}P_{N-1}(T).$$

The $H(div;\Omega)$-conforming broken Raviart-Thomas space of index $N-1$ is given by

$$RT_{N-1} := \{\mathbf{q} \in H(div;\Omega) \mid \mathbf{q}|_T \in RT_{N-1}(T), \ T \in \mathscr{T}_h\}.$$

5.4 Adaptive strategies

The implementation of the adaptive algorithm is done accordance with the cycle:

$$\text{SOLVE} \implies \text{ESTIMATE} \implies \text{MARK} \implies \text{REFINE}$$

where

- SOLVE amounts to seeking a solution $u_h \in V_h$ such that.

$$a_h(u_h, v_h) = \ell(v_h) \quad v_h \in V_h \quad \text{holds.}$$

- ESTIMATE is the computation of the error estimator η_h
- MARK deals with the selection of the elements and edges for refinement using the bulk criterion

$$\theta\ \eta_h\ \leq\ \eta_{\mathcal{M}}.$$

  where $\mathcal{M}$ denotes the set of cells and edges in $\mathscr{T}_h$ selected for refinement.

- REFINEMENT is realized using the newest vertex bisection in case $\mathscr{T}_h$ consists of triangles.

The adaptive mesh refinement process is driven by ESTIMATE through estimators which could be: residual-type *a posteriori* error estimators (popular owing to their low computational cost) or a wide range of averaging estimators and estimators based on the solution to local problems. The role of the estimator is two-fold, namely: drive the adaptive process and play the role of the global discretization error. The latter is achieved by proving the equivalence of the estimator to the global discretization error up to constants. For residual-type estimators, establishing the equivalence is challenging because of the dependence of the constants on sensitive parameters such as the high wave number or the inverse of the viscosity. In response to this, equilibration estimators have been proposed where the constants are unity. However, this estimator is expensive to compute since it relies on solving local problems to reconstruct the flux. The goal is to automate the selection of estimators in the “ESTIMATE” step based on the data of the problem and computed solution.

5.5 Residual-based estimator

We consider the residual-type *a posteriori* error estimator

$$\eta_h := \Big(\sum_{T \in \mathscr{T}_h} \eta_T^2 + \sum_{e \in \mathscr{E}_h^D} \eta_{e,1}^2 + \sum_{e \in \mathscr{E}_h^N} \eta_{e,2}^2 \Big)^{1/2},$$

consisting of the element residuals

$$\eta_T := h_T\ \|f + \Delta u_h\|_{0,T}, \quad T \in \mathbb{T}_h(\Omega),$$

and the edge residuals

$$\eta_{e,1} := h_e\ \left\| \left[\frac{\partial u_h}{\partial \mathbf{n}} \right]_e \right\|_e, \quad e \in \mathscr{E}_h^D,$$
$$\eta_{e,2} := h_e\ \left\| \frac{\partial u_h}{\partial \mathbf{n}} \right\|_e, \quad e \in \mathscr{E}_h^N.$$

5.5.1 Recovery-based estimator

In the absence of any information related to the exact gradient, we estimate the difference between the numerical (possibly non-smooth) gradient and a smooth gradient constructed to estimate the solution error.

5.6 Numerical examples

We present the existing results for the following benchmark problems focusing on the performance of the estimator and the optimal complexity achieved.

5.6.1 Smooth problem

$$\begin{aligned} -\nabla \cdot \big(a \nabla u\big) &= f \quad \text{in } \Omega := (0,1)^2, \\ u &= 0 \quad \text{on } \Gamma_D := \{0\} \times [0,1] \cup [0,1] \times \{0\}, \\ \mathbf{n} \cdot \nabla u &= g_N \quad \text{on } \Gamma_N := \Gamma \setminus \Gamma_D, \end{aligned}$$

where $a = 1$ and the data is chosen according to $u(x,y) = \frac{0.5 \sin(\pi x) \sin(\pi y)}{\pi^2}$. $\theta = 0.25$

5.6.2 L-Shape domain

$$\begin{aligned} -\nabla \cdot \big(a \nabla u\big) &= f \quad \text{in } \Omega := [-1,1]^2 \setminus [0,1]^2, \\ u &= 0 \quad \text{on } \Gamma_D := \{0\} \times [0,1] \cup [0,1] \times \{0\}, \\ \mathbf{n} \cdot \nabla u &= g_N \quad \text{on } \Gamma_N := \Gamma \setminus \Gamma_D, \end{aligned}$$

where $a = 1$ and the data (in polar coordinates) is chosen according to $u(r,\phi) = r^{\frac{2}{3}} \sin(\frac{2}{3}\phi)$. $\theta = 0.25$.

5.6.3 Contrast problem

$$\begin{aligned} -\nabla \cdot \big(a \nabla u\big) &= f \quad \text{in } \Omega := [-1,1]^2, \\ u &= 0 \quad \text{on } \Gamma_D = \Gamma. \end{aligned}$$

Here we choose a as follows

$$a = \begin{cases} -1 & (-1,0) \times (-1,0) \cup (0,1) \times (0,1); \\ 1 & \text{otherwise} \end{cases}$$

and $f = 1$ with no information about the exact solution.

6.0 Numerical Experiments

In this section, we provide numerical results for the stability identification of the Lorenz system using feed-forward neural networks; adaptive time stepping using an on-policy RL algorithm, namely proximal policy optimization; and adaptive mesh refinement for elliptic PDEs using an off-policy RL algorithm, namely the Q-learning algorithm.

6.1 Lorenz system state stability identification

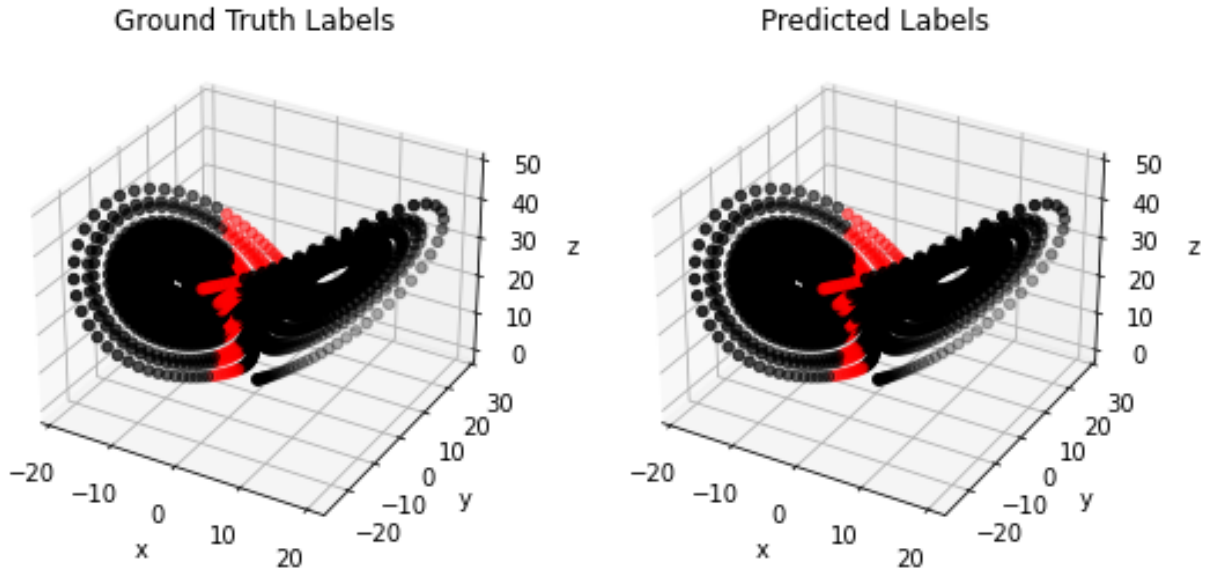
The Lorenz system is a chaotic system which is highly sensitive to initial conditions. Even a small perturbation in the initial conditions can cause the trajectories to diverge to a large extent and can affect the location of stable and unstable data points. Hence, it is a non-trivial task for neural network models to predict the unstable data points of Lorenz systems which have not been encountered during training. As we discuss in succeeding sections, it is especially challenging for neural networks to predict the unstable data points of Lorenz systems in mismatched conditions i.e when the initial conditions of Lorenz systems used for validation have been sampled from a different interval as compared to Lorenz systems used for training. In subsection 6.2, we present the results of predicting unstable data points using neural networks when the initial conditions for training and validation have been sampled from the same interval. In subsection 6.3, we discuss the results in mismatched conditions. Note that our dataset is unbalanced. The number of stable data points (majority class) is much greater than the number of unstable data points (minority class). In such cases, using the accuracy scores as a performance evaluation metric is misleading since the model will have a high accuracy even if it always predicts the majority class. Hence we do not report our results in terms of an accuracy score. Instead we report the results in terms of precision and recall scores which are defined in Eqs. 6.1.1.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \tag{6.1.1a}$$

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives} \tag{6.1.1b}$$

6.2 Initial conditions for training and validation data sampled from same interval

The example shown in Fig 6 shows the results of applying neural networks for predicting unstable data points when the initial condition variables x_0, y_0, and z_0 for both the training and the validation data are sampled from the same interval. The initial conditions for the training data for the model are sampled according to Eq. (3.5.2a), and those for validation data are sampled according to Eq. (3.6.3a). It is important to note that even though the sampling intervals for initial conditions are the same, the randomly sampled initial conditions for training and validation data within this interval are different. We observe that the neural network performs reasonably well in predicting the unstable data points with a precision of 0.90 and a recall of 0.744. Fig 7 shows another example of applying neural networks for the classification of unstable data points when the initial conditions for both the training and validation data are sampled from the same interval. In this example, the initial conditions for the training data for the model are sampled according to Eq. (3.5.2b), and those for validation data are sampled



(a) Ground truth labels.

(b) Predicted labels.

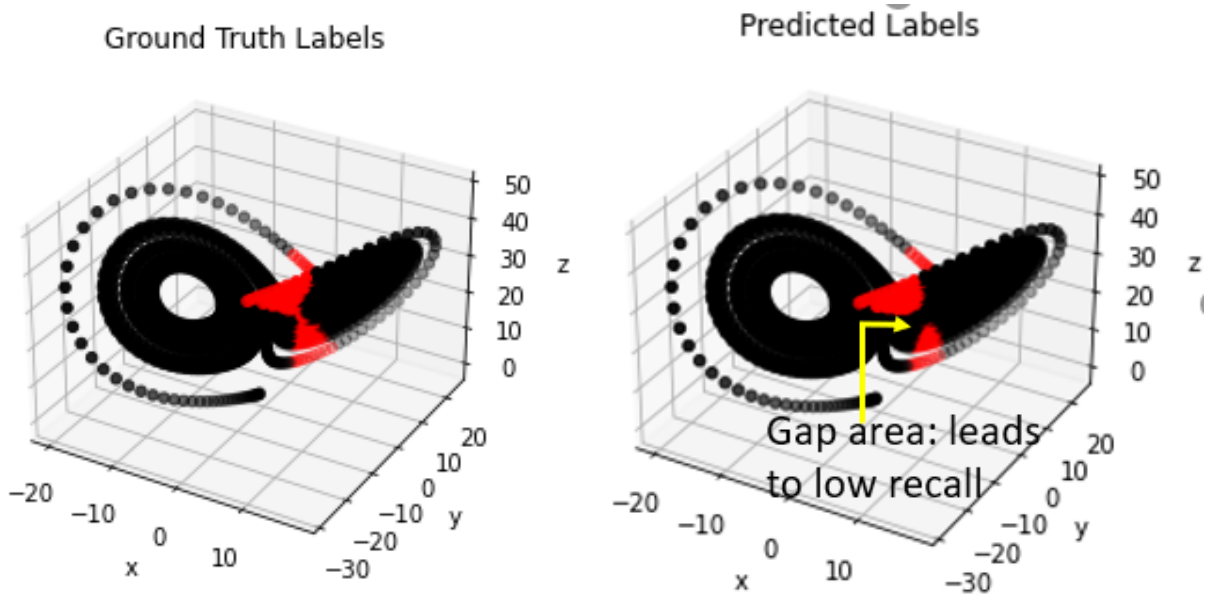
Figure 6: A classification result using neural network models in matched conditions. The initial conditions for training data were sampled according to Eq. (3.5.2a), and for validation data they were sampled according to Eq. (3.6.3a). The precision and recall scores are 0.9 and 0.744, respectively.

according to Eq. (3.6.3b). The precision and recall scores obtained are 0.94 and 0.78, respectively.

We observe that for both the examples, the recall scores are lower than those for precision. According to Eq. (6.1.1b), a lower recall score corresponds to a greater number of false negatives. This effect is better seen in Fig 7, where the model sometimes fails to predict the unstable data points in the gap region. This observation is also prevalent in mismatched conditions. We will see in Section 6.4 that the normalization scheme helps in reducing the false negatives and in improving the recall scores.

6.3 Initial conditions for training and validation data sampled from different intervals

Fig. 8 illustrates the performance of neural network models in mismatched conditions. The initial conditions for training data are sampled according to Eq. (3.5.2a), and for validation data they are sampled according to Eq. (3.6.3b). In this example, the neural network models almost completely miss the region of unstable data points and fail to identify them reliably. This is also reflected in the low precision and recall values of 0.01 and 0.004, respectively. As a reference, Fig. 9 shows the result on the same validation set as Fig. 8 using a neural network model that has the initial conditions for training data sampled according to Eq. 3.5.2b. A comparison of Fig. 8 and Fig. 9 shows the importance of initial conditions on the performance of neural networks for the specific task to identify the unstable data points of a Lorenz System. One can clearly see that using the features described in Section 3.7 as-is limits the usability of the neural network models. The models cannot be used for the mismatched scenario.



(a) Ground truth labels.

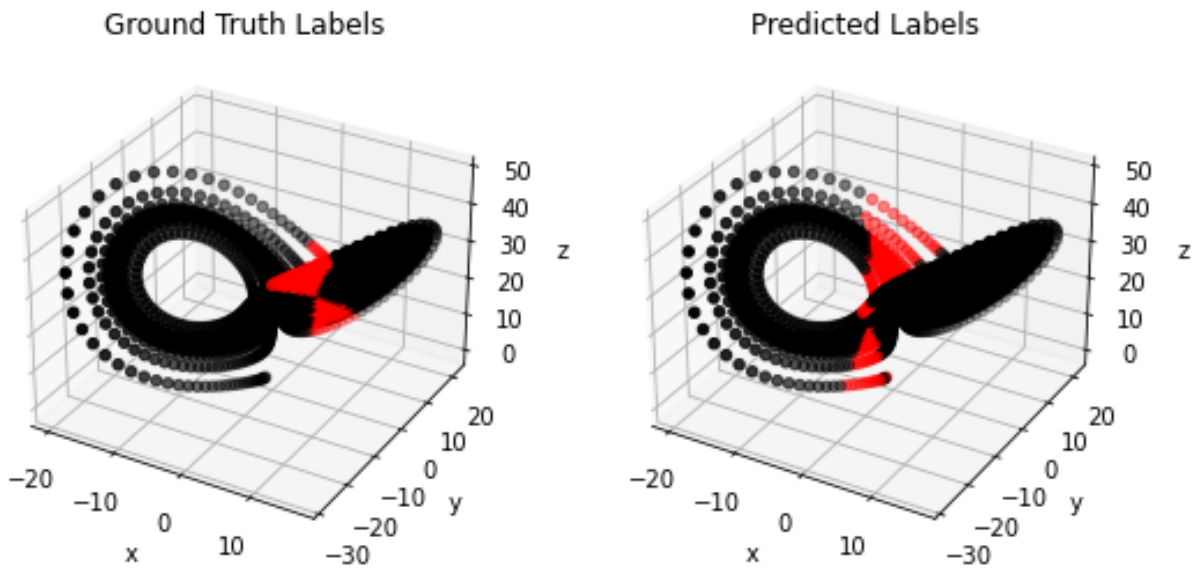
(b) Predicted labels.

Figure 7: A classification result using neural network models in matched conditions. The initial conditions for training data were sampled according to Eq. (3.5.2b), and for validation data they were sampled according to Eq. (3.6.3b). The precision and recall scores are 0.94 and 0.78, respectively.

Neural network models that are generalizable should be able to perform reliably on a broad variety of validation data. To this end, we train neural network models that have the initial conditions for the training data sampled from a slightly larger interval. Specifically, we use Eq. (3.5.2c) to sample the initial conditions for training data. Fig. 10 shows the classification result when the initial conditions were sampled according to Eq. (3.5.2c) for training data and Eq. (3.6.3b) for validation data. The precision and recall values are 0.846 and 0.125, respectively. Again, the low recall score indicates a high number of false negatives (which is also reflected in the small number of data points labeled in red in Fig. 10). Although the classification results of the models shown in Fig. 10 are not comparable to matched case shown in Fig. 9, these models are able to capture the region of unstable data points better than those in Fig. 8. Since it is impractical to have a matched model for every validation set (one whose initial conditions during training have been sampled from the same interval, such as the one shown in Fig. 9), we think that a model trained according to Eq. (3.5.2c) (such as the one shown in Fig. 10) might serve as a good candidate for training neural networks that can perform reliably on a wide variety of validation data. In the section 6.4, we show how the normalization scheme helps in further improving the performance of models trained according to Eq. (3.5.2c) in mismatched conditions.

6.4 Normalization

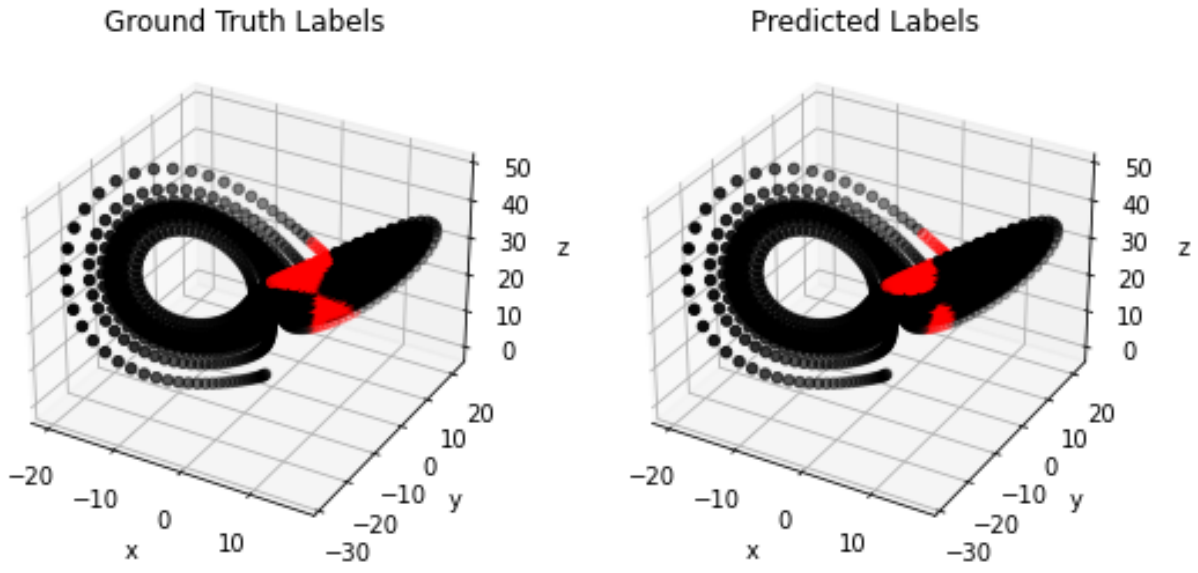
In order to understand why the classification results shown in Fig. 8 are worse than those in Fig. 9, we look at the statistics of the training and validation data. Fig. 12 compares the histograms and kernel density estimates of the features of training and validation data in the



(a) Ground truth labels.

(b) Predicted labels.

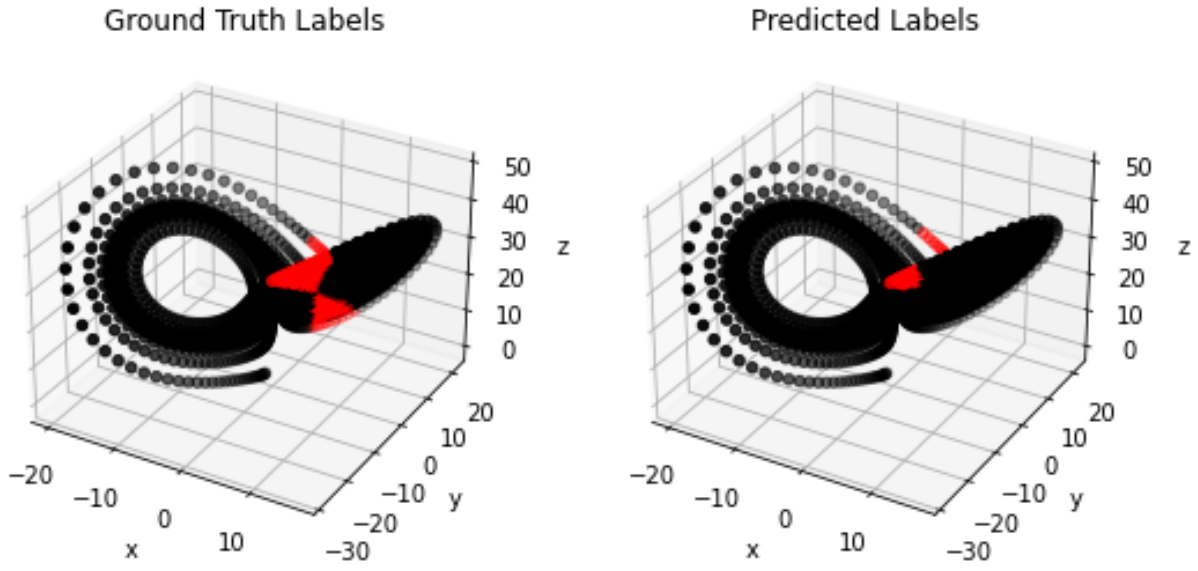
Figure 8: A classification result using neural network models in mismatched conditions. The initial conditions were sampled according to Eq. (3.5.2a) for training data and Eq. (3.6.3b) for validation data. The precision and recall scores are 0.01 and 0.004, respectively.



(a) Ground truth labels.

(b) Predicted labels.

Figure 9: A classification result using neural network models in matched conditions. The initial conditions were sampled according to Eq. (3.5.2b) for training data and Eq. (3.6.3b) for validation data. The precision and recall scores are 0.7955 and 0.5659, respectively.



(a) Ground truth labels.

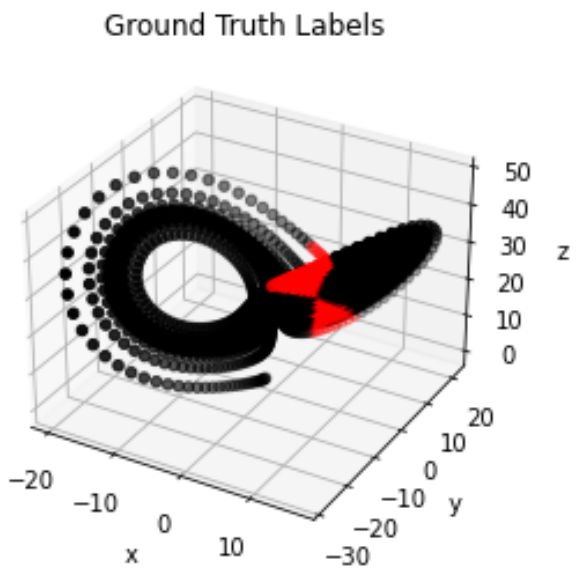
(b) Predicted labels.

Figure 10: A classification result using neural network models in mismatched conditions. The initial conditions were sampled according to Eq. (3.5.2c) for training data and Eq. (3.6.3b) for validation data. The precision and recall scores are 0.846 and 0.125, respectively.

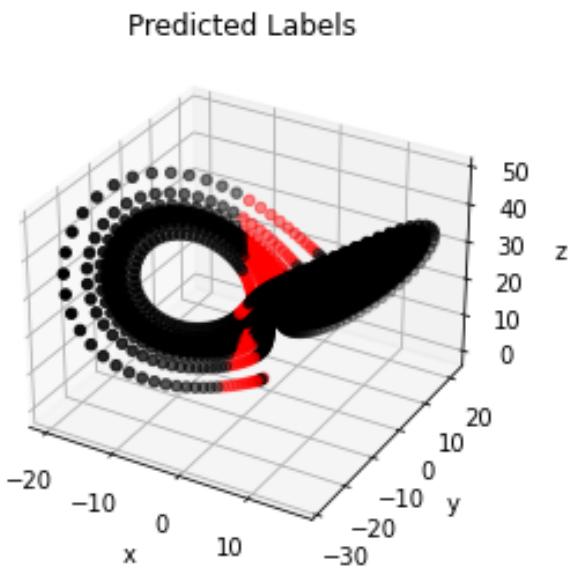
mismatched and matched cases. For simplicity, only the x-coordinate features are shown. In Fig. 12a, the initial conditions of training data were sampled according to Eq. (3.5.2a) and those of the validation data were sampled according to Eq. (3.6.3b). We note the strong mismatch between the histograms and kernel density estimates of training and validation data in Fig. 12a. This mismatch leads to a degradation in the performance of the neural networks. In contrast, the initial conditions of the training data were sampled according to Eq. (3.5.2b) in Fig. 12b, and the initial conditions for validation data were sampled according to Eq. (3.6.3b). Since the initial conditions for training and validation data were sampled from the same interval in Fig. 12b, there is no mismatch between their histograms and the kernel density estimates. Consequently, the performance of neural network models trained on this data is better, as evidenced by higher precision and recall scores.

In this work, we employ a normalization scheme that reduces the mismatch between training and validation data. Intuitively, we think that the smaller the mismatch between the distributions of training and validation data, the better the performance will be. With such a normalization scheme, the initial conditions for the training data can be sampled from a relatively small interval and the trained neural network models will give reliable performance on different kinds of validation data.

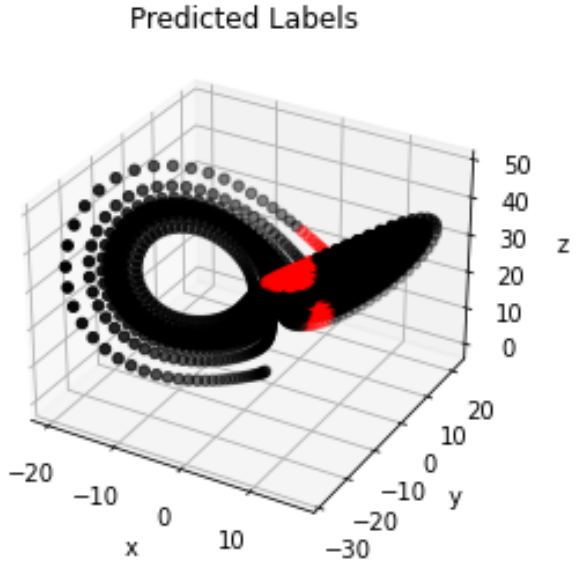
As described in section 3.4, we use 25 Lorenz systems for training the neural network. We sample the initial conditions of training data according to Eq. (3.5.2c), as we think this interval is a good candidate for training generalizable models. We normalize each of the 25 Lorenz systems separately. As noted in Section 3.7, the input to the neural network is a 6-dimensional feature vector. We calculate the mean and standard deviation along each of the feature dimensions for each of the Lorenz systems. Specifically, if x represents the x-coordinate of one of the data points of the Lorenz System, we transform x according to Eq. 6.4.2a, where the



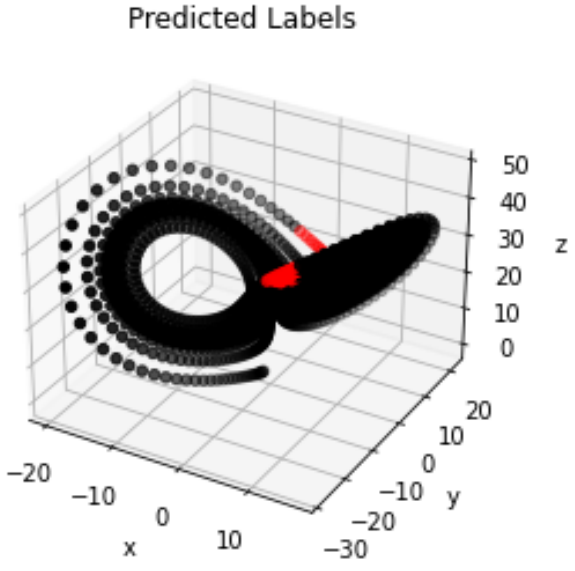
(a) Ground truth labels.



(b) Predicted labels.

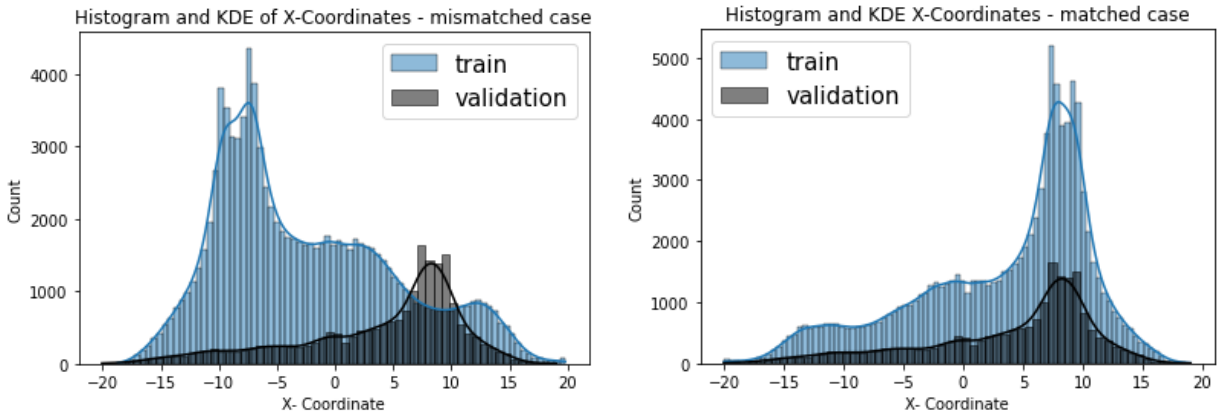


(c) Predicted labels.



(d) Predicted labels.

Figure 11: A classification result using neural network models. The initial conditions were sampled according to Eq. (3.6.3b) for validation data. The initial conditions for training data were sampled according to Eq. (3.5.2a) for Fig. 7a, Eq. (3.5.2b) for Fig. 11b, and Eq. (3.5.2c) for Fig. 11c.



(a) Histogram & KDE – mismatched case.

(b) Histogram & KDE – matched case.

Figure 12: Histogram and Kernel Density Estimates (KDE) of x-coordinates of the training and validation data for the mismatched and matched cases. The initial conditions for training data were sampled according to Eq. (3.5.2a) for Fig. 12a and according to Eq. (3.5.2b) for Fig. 12b. The initial conditions for validation data were sampled according to Eq. (3.6.3b) for both the cases.

mean u_x and standard deviation σ_x are given by Eq. (6.4.2b) and Eq. (6.4.2c), respectively.

$$x = \frac{x - \mathsf{u}_x}{\sigma_x} \tag{6.4.2a}$$

$$\mathsf{u}_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{6.4.2b}$$

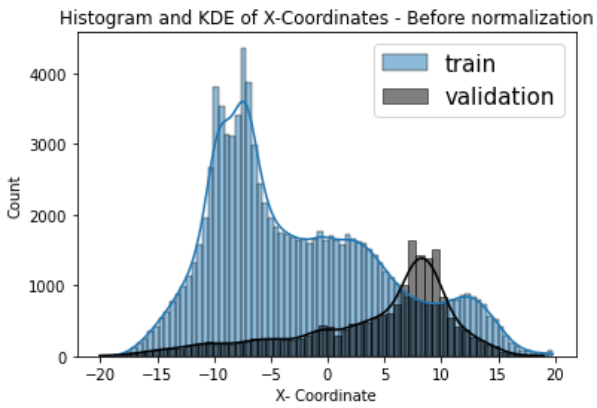
$$\sigma_x = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mathsf{u}_x)^2 \tag{6.4.2c}$$

We perform the normalization on both the training and validation data. Fig. 13 compares the histograms and kernel density estimates of the training and validation data before and after normalization. One can clearly see that after normalization (Fig. 13b), the histograms and kernel density estimates of the training and validation data have a greater overlap and a reduced mismatch.

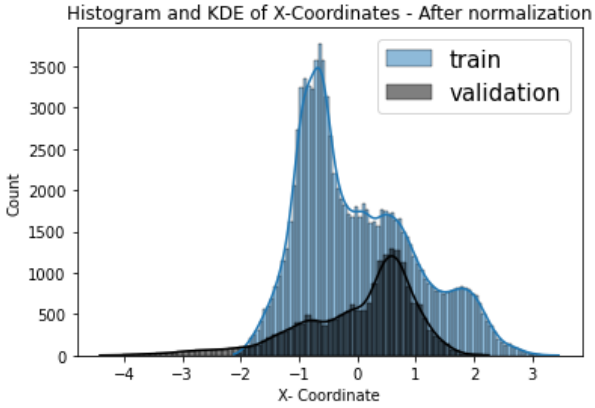
Fig. 14 shows the location of stable and unstable data points before and after normalization. Thus, we are able to verify that the normalization scheme does not alter the relative location of stable and unstable data points. It merely shifts the data points along the axes.

6.4.1 Normalization

Fig. 15 shows the classification result in the mismatched case when the neural network is trained on normalized feature vectors. The initial conditions for training are sampled according to Eq. 3.5.2c, and those for validation are sampled according to Eq. 3.6.3b. The precision are recall scores are 0.983 and 0.9727, respectively. One can also see that the results of Fig. 15 clearly outperform those of Fig. 8, Fig. 9, and Fig. 10. This also highlights that neural network

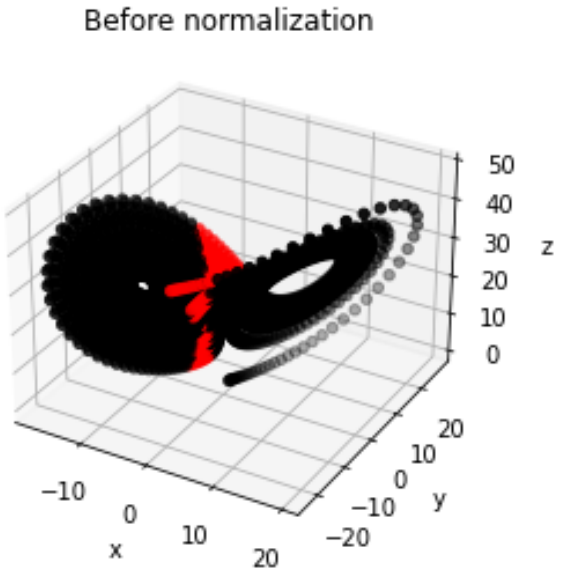


(a) Histogram & KDE – mismatched case before normalization.

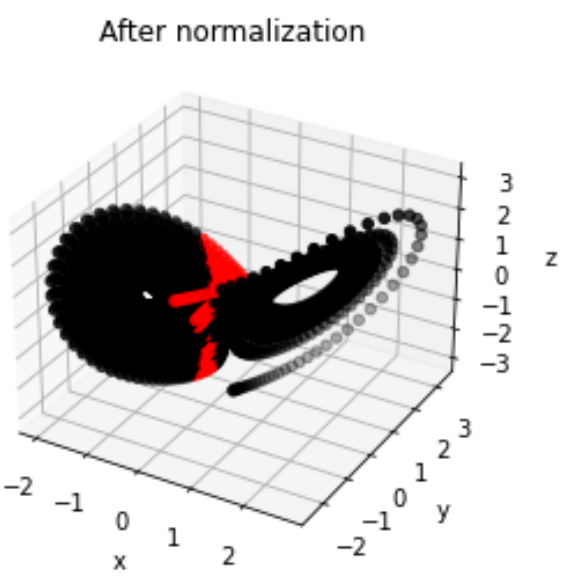


(b) Histogram & KDE – mismatched case after normalization.

Figure 13: Histogram and Kernel Density Estimates (KDE) of x-coordinates of the training and validation data for the mismatched case, before and after normalization. The initial conditions for training data for Fig. 13a and Fig. 13b were sampled according to Eq. (3.5.2a). The initial conditions for validation data were sampled according to Eq. (3.6.3b).



(a) Stable and unstable data points of Lorenz System before normalization.



(b) Stable and unstable data points of a Lorenz System after normalization.

Figure 14: Comparison of the location of stable and unstable data points of a Lorenz System before and normalization. Note that the only change is in the ranges of axes values.

Table 6.4.1: Average precision and recall values – without normalization.

Training	Validation	Mean Precision	Mean Recall	Stddev-Precision	Stddev-Recall
$[0, 1]$	$[0, 1]$	0.82	0.617	0.122	0.170
$[-1, 0]$	$[-1, 0]$	0.632	0.455	0.247	0.202
$[0, 1]$	$[-1, 0]$	0.034	0.026	0.049	0.044
$[-1, 0]$	$[0, 1]$	0.028	0.032	0.014	0.025
$[-1, 1]$	$[-1, 0]$	0.704	0.128	0.195	0.06

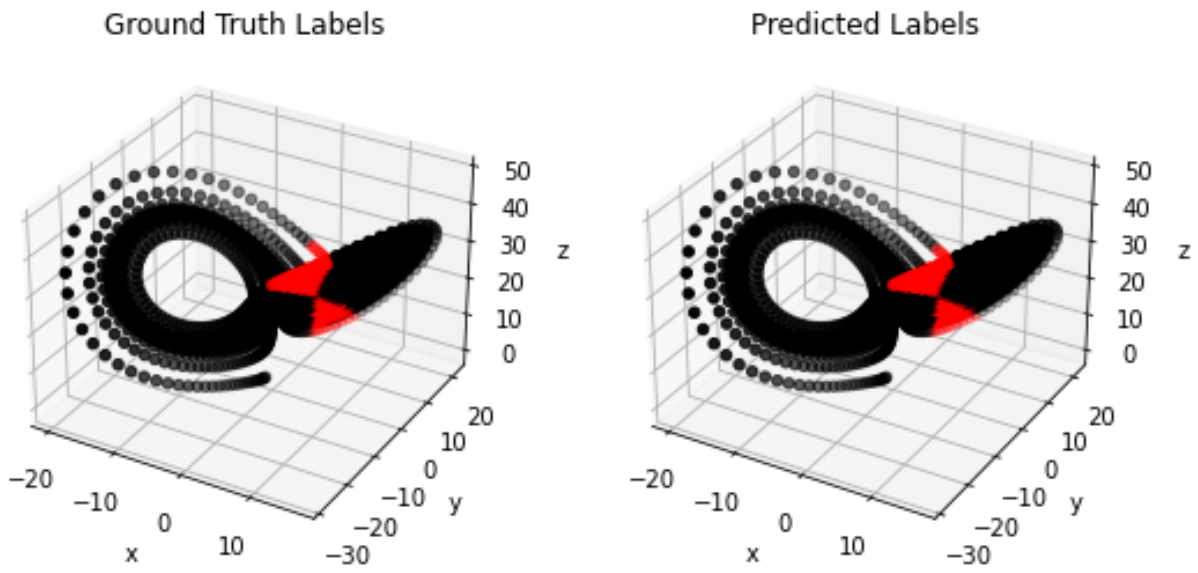
models are sensitive to the mismatch between distributions of training and validation data and that normalization schemes are necessary to reduce the mismatch.

We also verify whether neural networks trained on normalized data, whose initial conditions are sampled according to Eq. (3.5.2c) can perform reliably on a wide variety of validation data. To this end, we test the neural network model on validation data whose initial conditions are sampled according to Eq. (3.6.3d), Eq. (3.6.3e), and Eq. (3.6.3f). Fig. 16 shows the results of applying neural network models when the initial conditions of validation data were sampled according to Eq. (3.6.3d). This interval is completely outside of the interval used for sampling the initial conditions of the training data, (Eq. (3.5.2c)). Yet, we see that the classification performance is reasonably good, with a precision score of 0.99 and a recall score of 0.939. Fig. 17 and Fig. 18 show the classification performance when the initial conditions for validation data were sampled according to Eq. (3.6.3e) and Eq. (3.6.3f), respectively. Both these intervals are much wider compared to those for sampling initial conditions for training data, (Eq. (3.5.2c)). Again, we see that the performance of neural network models is reasonably accurate. The precision and recall scores of the classification result shown in Fig. 17 are 1 and 0.91, respectively. The precision and recall scores of the classification result shown in Fig. 18 are 1 and 0.95, respectively.

Tables 6.4.1 and 6.4.2 summarize the results of our experiments by depicting the average precision and recall scores without and with normalization, respectively. For each row, neural network models are trained using training data whose initial conditions are sampled from the intervals specified in the first column. As described in Section 3.4, we use 25 Lorenz systems for training and 5 Lorenz systems for validation. The mean precision and recall values are obtained by averaging over the 5 validation Lorenz systems. We also show the standard deviations to quantify how much the precision and recall scores deviate from the mean. Per Table 6.4.1, we observe that the performance of the neural network models drops significantly in the mismatched case (Rows 3 and 4 of Table 6.4.1). We choose the interval $[-1, 1]$ to sample initial conditions for training data for our further experiments, per our observation that it represents a valid subset of the validation data intervals that the neural network is likely to encounter. The results in Table 6.4.2 indicate that our assumption is justified. The neural networks whose training data are sampled from the interval $[-1, 1]$ perform well on a wide variety of validation datasets. These results also show that our normalization scheme greatly helps in improving the performance of neural networks on mismatched data and is a promising step towards training generalizable neural networks for this classification task.

Table 6.4.2: Average precision and recall values – with normalization. (IC = initial conditions.)

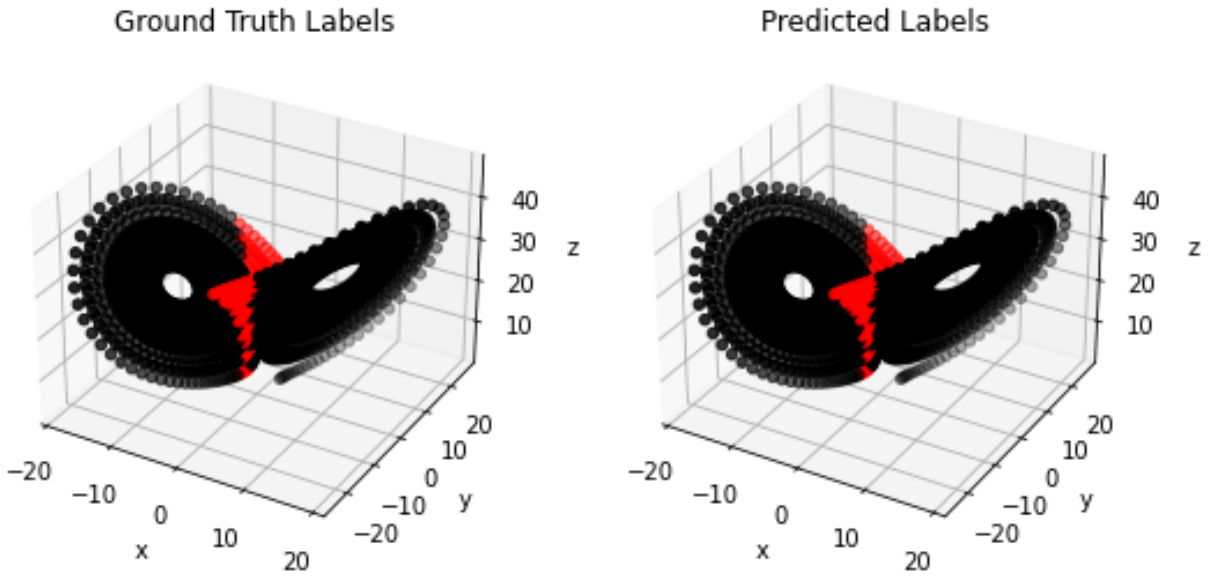
Training sample x_0, y_0, z_0 from	Interval to sample ICs for Validation	Mean Precision	Mean Recall	Stddev-Precision	Stddev-Recall
$[-1, 1]$	$[0, 1]$	0.96	0.964	0.032	0.007
$[-1, 1]$	$[-1, 0]$	0.963	0.969	0.039	0.015
$[-1, 1]$	$[-1, 1]$	0.982	0.978	0.007	0.005
$[-1, 1]$	$[2, 4]$	0.988	0.934	0.01	0.015
$[-1, 1]$	$[0, 10]$	0.975	0.955	0.025	0.028
$[-1, 1]$	$[-10, 10]$	0.952	0.9464	0.066	0.032



(a) Ground truth labels.

(b) Predicted labels.

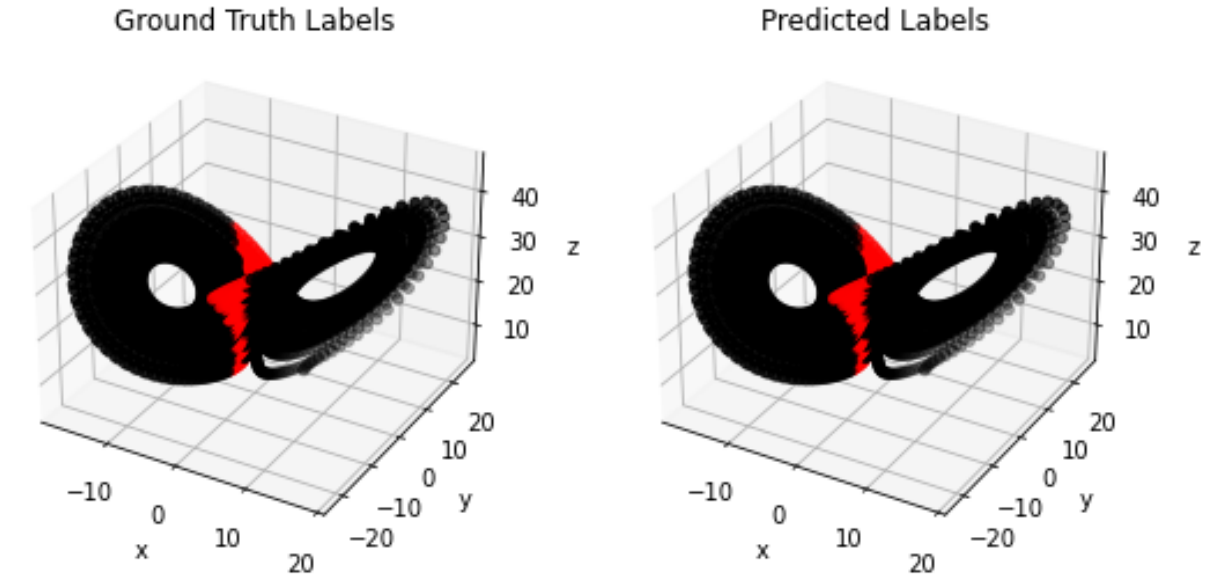
Figure 15: Classification results using neural network models in mismatched conditions, after normalization. The initial conditions for training data were sampled according to Eq. (3.5.2c), and for validation data they were sampled according to Eq. (3.6.3b). The precision and recall scores are 0.983 and 0.972, respectively.



(a) Ground truth labels.

(b) Predicted labels.

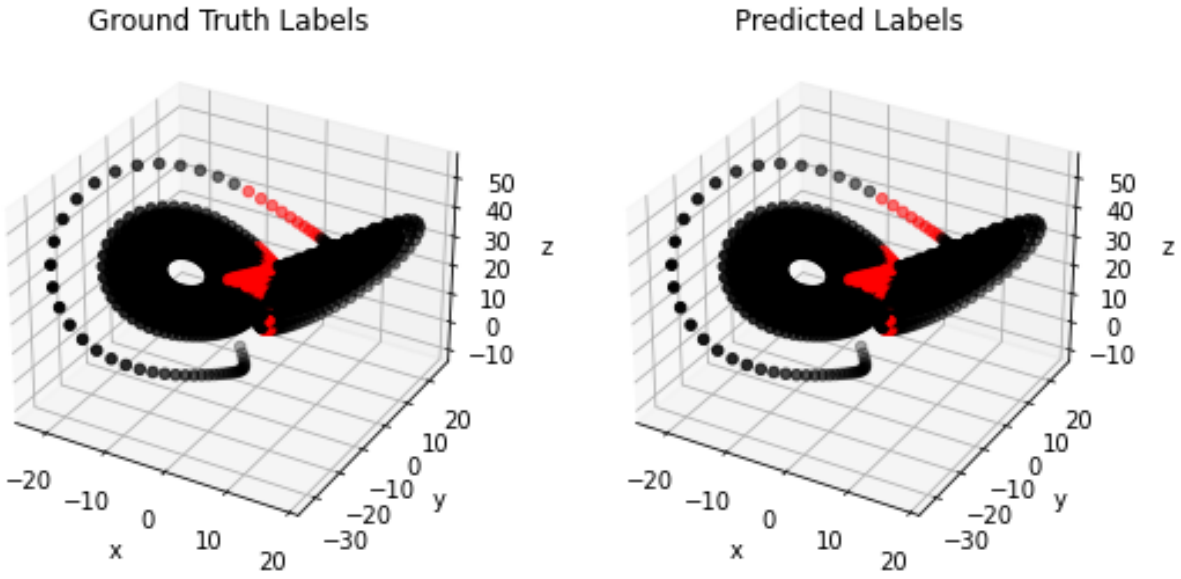
Figure 16: Classification results using neural network models in mismatched conditions, after normalization. The initial conditions for training data were sampled according to Eq. (3.5.2c), and for validation data they were sampled according to Eq. (3.6.3d). The precision and recall scores are 0.99 and 0.939, respectively.



(a) Ground truth labels.

(b) Predicted labels.

Figure 17: Classification results using neural network models in mismatched conditions, after normalization. The initial conditions for training data were sampled according to Eq. (3.5.2c), and for validation data they were sampled according to Eq. (3.6.3e). The precision and recall scores are 1 and 0.91, respectively.



(a) Ground truth labels.

(b) Predicted labels.

Figure 18: Classification results using neural network models in mismatched conditions, after normalization. The initial conditions for training data were sampled according to Eq. (3.5.2c), and for validation data they were sampled according to Eq. (3.6.3f). The precision and recall scores are 1 and 0.95, respectively.

6.5 Adaptive time stepping for nonlinear ODEs

In this section we show numerical results for the Van der Pol oscillator and the Lorenz equations, in which we compare the numerical cost in terms of number of the time steps and the number of function evaluations using the Dormand-Prince $rk45$ algorithm and RL-enhanced $rk45$ algorithm.

6.5.1 Van der Pol oscillator

Fig. 19 shows the solution of the Van der Pol oscillator computed using the Dormand-Prince $rk45$ algorithm in scipy and RL-enhanced $rk45$ algorithms. Fig. 20 gives the corresponding reward function obtained during the RL training method.

In these numerical experiments, we observed a total number of time steps of 63 from scipy $rk45$ and 69 from the RL-enhanced $rk45$. This shows that the scipy based $rk45$ is computationally more efficient than the RL-enhanced $rk45$. However, we noticed that the RL-enhanced $rk45$ is more efficient in terms of the number of function evaluations required, compared to the number used in the scipy-based $rk45$. This is due to the fact that the scipy-based implementation rejects the solution more often than the RL-enhanced $rk45$.

6.5.2 Lorenz equations

Similarly, Fig. 21 shows the solution of the Lorenz equations computed using the Dormand-Prince $rk45$ algorithm in scipy and the RL-enhanced $rk45$ algorithm. Fig. 22 shows

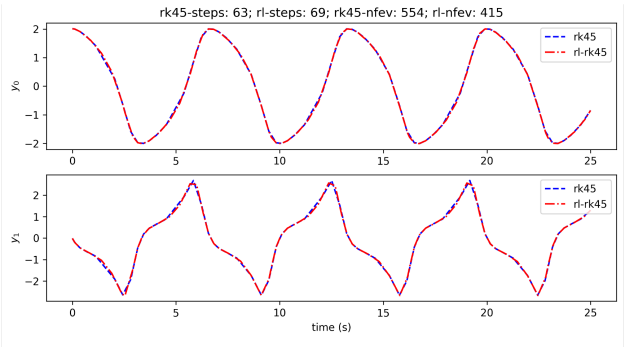


Figure 19: Van der Pol oscillator solution.

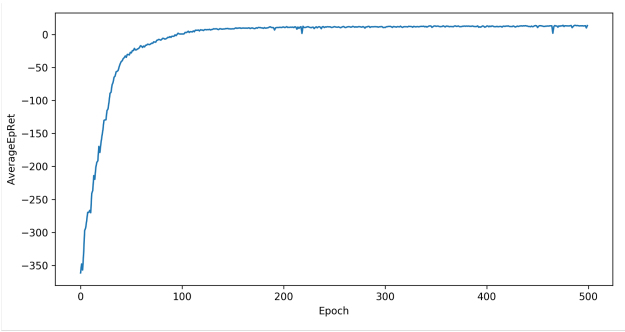


Figure 20: Van der Pol oscillator reward function.

the corresponding reward function obtained during the RL training method.

For the Lorenz equations, we noticed that RL-enhanced $rk45$ performed poorly both in terms of the number of time steps (1064) and the number of function evaluations (2881), compared to those used by the scipy-based $rk45$ with 134 and 480, respectively. This is understandable as the Lorenz system is a chaotic system and it is much harder to train RL algorithms on such systems, warranting further study.

6.6 Adaptive mesh refinement for 2-D PDEs

In this section, we compare the efficiency of adaptive mesh refinement using RL-enhanced adaptive mesh refinement to that of classical adaptive mesh refinement methods such as uniform, residual-based, and ZZ mesh refinement approaches. Fig. 24 plots the relationship between error and degrees-of-freedom (DoFs) for RL-based adaptive mesh refinement and classical adaptive mesh refinement techniques.

Fig. 23 shows the reward function convergence during the RL training process. In this work, we observe that although RL-based adaptive mesh refinement performed better than uniform mesh refinement, it did not show any better results than residual and ZZ mesh refinement approaches.

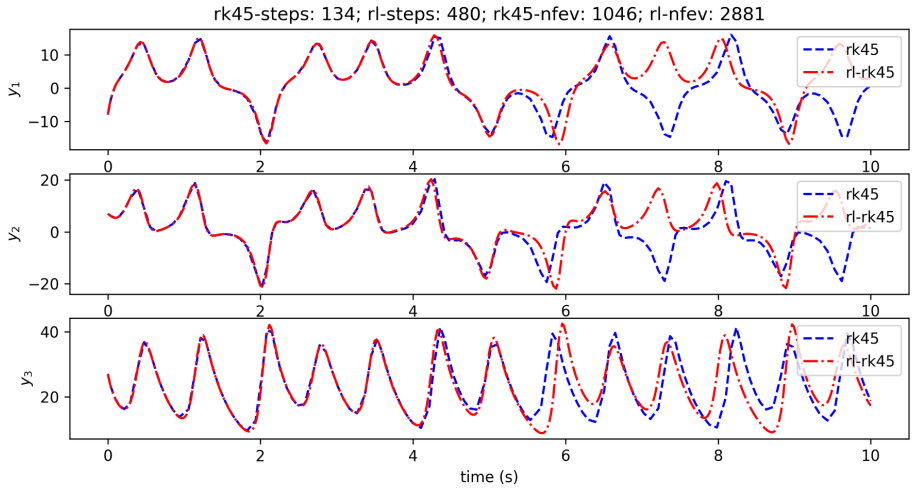


Figure 21: Solution of Lorenz equations.

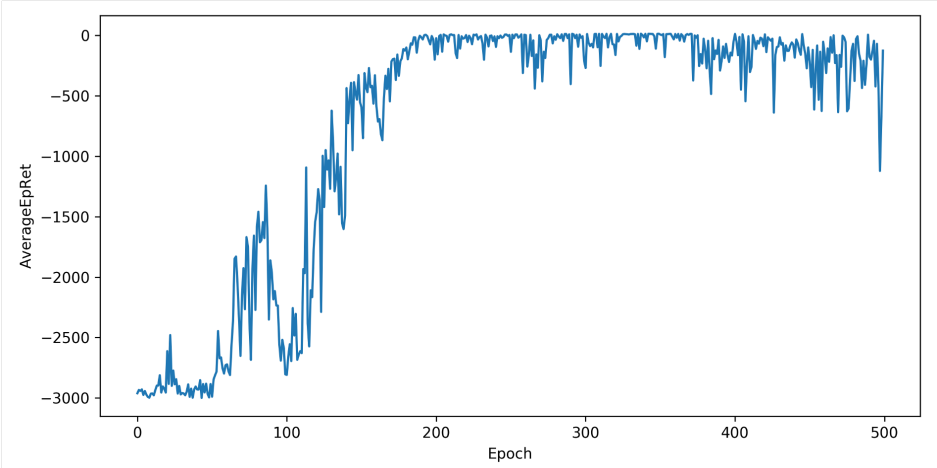


Figure 22: Lorenz equations reward.

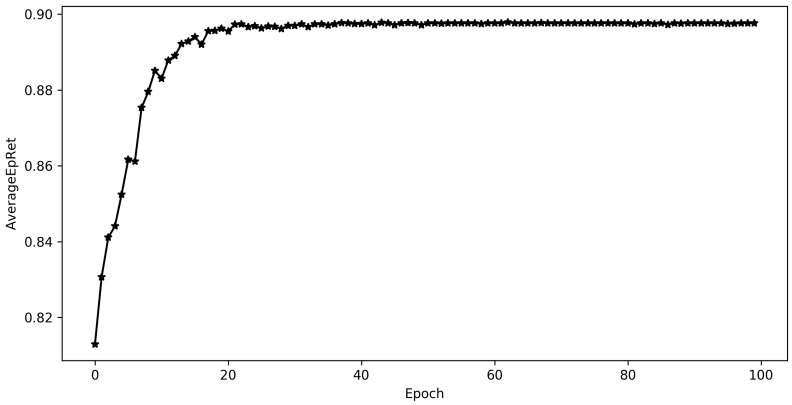


Figure 23: Reward during off-policy RL-training.

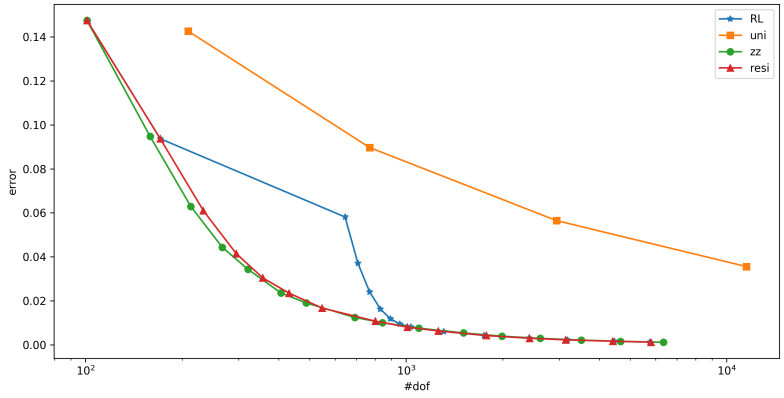


Figure 24: Comparison of errors from RL-based adaptive mesh refinement and classical adaptive mesh refinement techniques over a range of DoFs.

7.0 Conclusions

In this project, we developed a sequential decision making framework (SDM) based on leaning, reasoning and planning approaches. We used feed-forward neural networks as the learning tool within the SDM framework to classify stability regions in Lorenz System of equations which are chaotic in nature. We employed on-policy and off-policy reinforcement learning (RL) algorithms, namely, proximal policy optimization (PPO) and deep Q-networks (DQNs), respectively. These were employed to develop RL-enhanced adaptive time stepping schemes to solve nonlinear ordinary differential equations, as well as adaptive mesh refinement techniques for 2-D elliptic equations. Our numeral results suggest that classification performed using a feed-forward neural network model can accommodate sensitivity to initial conditions in the training and validation datasets. The classification performance degrades when there is a mismatch between initial conditions used for generating training and validation datasets. We introduce a normalization scheme and show that it significantly improves the classification performance of neural network models in mismatched conditions. More broadly, our results show the feasibility of using neural networks to study stability aspects of chaotic systems, like the Lorenz-63 system. Improvement in state estimation via neural network-based classification can significantly enhance automated decision making in complex systems using a SDM framework. We observed that the proposed RL-enhanced adaptive time stepping method performed better than Dormand-Prince method in terms of the number of function evaluations for the Van der Pol oscillator. However, we did not observe much improvement for the highly chaotic Lorenz equations. We also notice that the RL-approach for adaptive mesh refinement for 2-D elliptic equations performed better than uniform refinement. On the other hand, its performance is at most equal to that of the classical adaptive mesh refinement benchmark. This approach needs further study to improve RL-training. Future work will explore the scalability and explainability aspects of our normalization scheme for neural network-based classification applied to state estimation in large scale complex systems. We will also explore the adaptation of SDM frameworks to large multiscale and multi-physics problems.

References

[1] Karsten Ahnert et al. “Odeint – Solving Ordinary Differential Equations in C++”. In: (2011). DOI: 10.1063/1.3637934. URL: http://dx.doi.org/10.1063/1.3637934.

[2] Saeid Amiri, Mohammad Shokrolah Shirazi, and Shiqi Zhang. “Robot Sequential Decision Making using LSTM-based Learning and Logical-probabilistic Reasoning”. In: *CoRR* (2019).

[3] Eduardo L Brugnago et al. “Classification strategies in machine learning techniques predicting regime changes and durations in the Lorenz system”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30.5 (2020), p. 053101.

[4] Kathleen Champion et al. “Data-driven discovery of coordinates and governing equations”. In: *Proceedings of the National Academy of Sciences* 116.45 (2019), pp. 22445–22451.

[5] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[6] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[7] G. Hinton et al. “Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups”. In: *IEEE Signal Processing Magazine* 29.6 (2012), pp. 82–97. DOI: 10.1109/MSP.2012.2205597.

[8] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[9] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[10] Guokun Lai et al. *Bridging the domain gap in cross-lingual document classification*. 2019. arXiv: 1909.07009 [cs.CL].

[11] Martin Lellep et al. “Using machine learning to predict extreme events in the Hénon map”. In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 30.1 (Jan. 2020), p. 013113. ISSN: 1089-7682. DOI: 10.1063/1.5121844. URL: http://dx.doi.org/10.1063/1.5121844.

[12] Edward N Lorenz. “Deterministic Nonperiodic Flow Journal of the Atmospheric Sciences Vol. 20”. In: *No. In. XX* (1963).

[13] Volodymyr Mnih et al. “Playing atari with deep reinforcement learning”. In: *arXiv preprint arXiv:1312.5602* (2013).

[14] KS Narendra and K Parthasarathy. “Identification and control of dynamical systems using neural networks”. In: *IEEE transactions on neural networks* 1.1 (). ISSN: 2162-2388.

[15] Jaideep Pathak et al. “Hybrid Forecasting of Chaotic Processes: Using Machine Learning in Conjunction with a Knowledge-Based Model”. In: *CoRR* abs/1803.04779 (2018). arXiv: 1803.04779. URL: http://arxiv.org/abs/1803.04779.

[16] Sebastian Scher and Gabriele Messori. “Generalization properties of feed-forward neural networks trained on Lorenz systems”. In: *Nonlinear processes in geophysics* 26.4 (2019), pp. 381–399.

[17] Andrew W Senior et al. “Improved protein structure prediction using potentials from deep learning”. In: *Nature.* 577.7792 (). ISSN: 0028-0836.

[18] Masashi Sugiyama and Motoaki Kawanabe. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation*. MIT press, 2012.

[19] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[20] Ashish Vaswani et al. “Attention is All you Need”. In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon et al. Vol. 30. Curran Associates, Inc., 2017, pp. 5998–6008. URL: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[21] Emmanuel Vincent et al. “An analysis of environment, microphone and data simulation mismatches in robust speech recognition”. In: *Computer Speech I& Language* 46 (2017), pp. 535–557. ISSN: 0885-2308. DOI: https://doi.org/10.1016/j.csl.2016.11.005. URL: http://www.sciencedirect.com/science/article/pii/S0885230816301231.

[22] Pauli Virtanen et al. “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python”. In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[23] Pantelis R Vlachas et al. “Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks”. In: *Proceedings.* 474.2213 (). ISSN: 1364-5021.

Pacific Northwest National Laboratory

902 Battelle Boulevard
P.O. Box 999
Richland, WA 99354
1-888-375-PNNL (7665)

www.pnnl.gov