SUSTAINING CHAOS USING DEEP REINFORCEMENT LEARNING

by

Sumit Vashishtha

A Thesis Submitted to the Faculty of

The Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University

Boca Raton, FL

May 2020

Copyright 2020 by Sumit Vashishtha

SUSTAINING CHAOS USING DEEP REINFORCEMENT LEARNING

by

Sumit Vashishtha

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Siddhartha Verma, Department of Ocean and Mechanical Engineering, and has been approved by the members of his supervisory committee. It was submitted to the faculty of the Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

SUPERVISORY COMMITTEE:

Siddhartha Verma, Ph.D.
Thesis Advisor

Issac Elishakoff, Ph.D.

Stewart Glegg, Ph.D.

Manhar Dhanak, Ph.D.
Chair, Department of Ocean and Mechanical Engineering

Stella N. Batalama, Ph.D.
Dean, The Engineering and Computer Science

Robert W. Stackman Jr., Ph.D.
Dean, Graduate College

Date

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my academic advisor, Dr. Siddhartha Verma. Had it not been the kindness he had shown in granting me immense academic leeway to work on a topic of my choice, this work would not have been possible. His innate instinct to make things simple and beautiful has never failed to cherish my spirits. For their valuable guidance, acknowledgements are due to the other esteemed committee members (Prof. Elishakoff and Prof. Glegg) of this thesis as well. I would also like to thank Prof. JA Scott Kelso and Prof. KR Sreenivasan. Enriching discussions with them during the last one year or so have greatly reinforced my vision and ambition to decipher the meaningful (and essential) constructs that govern the dynamics of nature. Support from family and friends can never be overemphasized, and I would like to thank my parents, Vishu (my sis.), flat-mates (Uma di, Aritra da, Mazhar bhai), lab-mates (Aishwarya, Eric, and Kayhan), and other friends (Amir Khusro Kazemi, Ignacio, Ishwarya). They were always delighted in bestowing their kindness upon me. Finally, I would like to wholeheartedly extend my thanks to the FAU administration (In particular, Ms. Calnick, Mr. Nicholas Kelly, Ms. Barbara, and Ms. Elizabeth). They strove hard to make my stay at FAU as smooth as it has been.

ABSTRACT

Author:	Sumit Vashishtha
Title:	Sustaining Chaos using Deep Reinforcement Learning
Institution:	Florida Atlantic University
Thesis Advisor:	Dr. Siddhartha Verma
Degree:	Master of Science
Year:	2020

Numerous examples arise in fields ranging from mechanics to biology where disappearance of Chaos can be detrimental. Preventing such transient nature of chaos has been proven to be quite challenging. The utility of Reinforcement Learning (RL), which is a specific class of machine learning techniques, in discovering effective control mechanisms in this regard is shown. The autonomous control algorithm is able to prevent the disappearance of chaos in the Lorenz system exhibiting meta-stable chaos, without requiring any a-priori knowledge about the underlying dynamics. The autonomous decisions taken by the RL algorithm are analyzed to understand how the system's dynamics are impacted. Learning from this analysis, a simple control-law capable of restoring chaotic behavior is formulated. The reverse-engineering approach adopted in this work underlines the immense potential of the techniques used here to discover effective control strategies in complex dynamical systems. The autonomous nature of the learning algorithm makes it applicable to a diverse variety of non-linear systems, and highlights the potential of RL-enabled control for regulating other transient-chaos like catastrophic events.

Dedicated to the loving memory of Grandpa

SUSTAINING CHAOS USING DEEP REINFORCEMENT LEARNING

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

Nonlinear dynamical systems are known to exhibit a rich variety of different long-term behaviors such as stationary points, limit cycles, quasiperiodic, and chaotic motion. Some systems possess a multitude of coexisting attractors. This coexistence of several attractors for a given set of system parameters is termed as multistability. Such multistability was first invented in the perception [2] system of living beings. Other examples of systems exhibiting multi-stability includes lasers, chemical reactions, an ensemble of coupled neurons, climate, and many more. Such multi-stability is often beneficial in the sense that the multitude of coexisting states in the system can correspond to different system performances suitable for different tasks. In order to maintain the system in some desirable state, or to switch it from one state to another, different control techniques [3] such as delayed-feedback, harmonic perturbation have been developed. Though a good scope of improvement lies in this area of control itself, yet another control challenge for multi-stable systems with coexisting chaotic and non-chaotic attractors has been the control of catastrophic bifurcations known as crisis. During crisis, the system converges to the non-chaotic attractor after a few chaotic transients. Such transient-chaos [4] is often undesirable, and has been conjectured to be the culprit for several deleterious phenomena such as voltage collapse in electric power systems [5] and species extinction in ecology [6]. The goal of this thesis had been the development of a control-technique for transient chaos based on the machine learning method known as reinforcement learning.

The outline of this document is as follows. Chapter 2 introduces basic properties of Chaos and its control. A mathematical overview of different reinforcement learning algorithms is given in Chapter 3. Finally, In Chapter 4, the problem of the control of transient chaos is discussed, and the success and operation of reinforcement learning based controller in controlling transient chaos is described. Conclusions are made in Chapter 5.

CHAPTER 2
CHAOS

The mathematician Pierre Simon Laplace once remarked [7] that ”An intelligence which could comprehend all the forces by which nature is animated and the respective situation of the beings who compose itan intelligence sufficiently vast to submit these data to analysis . . . for it, nothing would be uncertain and the future, as the past, would be present to its eyes”. In other words, if one exactly knows the initial-state of a system, one can be ascertained of its past and future. However, an Exact knowledge of a real-world initial state is never possible. Until the 19th century, the tacit assumption had always been that approximate knowledge of the initial state implies approximate knowledge of the final state. This notion was first challenged by Poincaré [8], who, while studying the three-body-problem in celestial mechanics, observed that ”it may happen that small differences in the initial conditions produce very great ones in the final phenomena... Prediction becomes impossible.” It is this sensitive dependence on initial conditions which came to be known as chaos. Poincaré’s ideas were dismissed by many as pathologies or artifacts of noise or methodological shortcomings. It took nearly another century for the establishment of chaos theory, when in 1963, Edward Lorenz found chaos in the numerical solutions of a simplified 12-variable dynamical model of the weather where chaos was apparent in the gradual divergence of two time series calculated using identical equations but slightly different initial conditions. This effect, however, is deterministic, not random. In this chapter, we will discuss a few mathematical terminologies related to chaos, areas where chaos has been relevant, and the popular methods used to control chaos.

2.1 MATHEMATICAL TERMINOLOGIES

In order to define different mathematical concepts related to chaos, we introduce the 3-variable Lorenz system of equations. These equations correspond to a simplified representation of the Rayleigh Benard Convection problem, and can be written as follows,

$$\frac{dx}{dt} = \sigma(y - x) \tag{2.1a}$$
$$\frac{dy}{dt} = x(\rho - z) - y \tag{2.1b}$$
$$\frac{dz}{dt} = xy - \beta z \tag{2.1c}$$

Here σ, ρ, $\beta > 0$ are parameters, σ is the Prandtl number, ρ is the Rayleigh number. β has no name.(In the convection problem it is related to the aspect ratio of the rolls. The system has only two non-linearities, the quadratic terms xy and xz .There is an important symmetry in the Lorenz equations. If we replace $(x, y) \to (-x, -y)$ in, the equations stay the same. Hence, if $[x(t), y(t), z(t)]$ is a solution, so is $[-x(t), -y(t), z(t)]$. Another

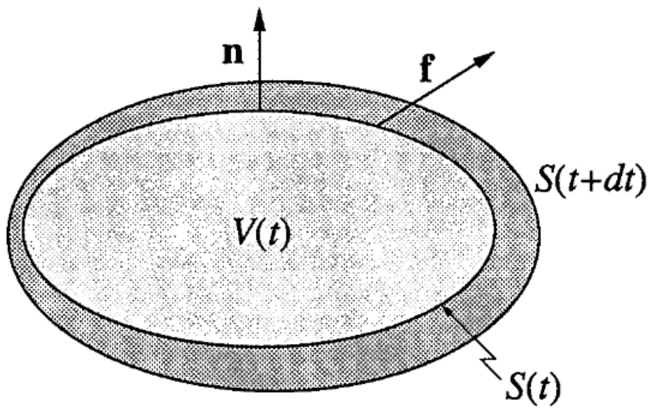


Figure 2.1: Schematic for temporal evolution of a volume of initial conditions. Adapted from ref. [1]

important property of the Lorenz system is that they are dissipative in nature. That is, if we start with a huge solid blob of initial conditions, it eventually shrinks to a limiting set of zero volume. All trajectories starting in the blob end up somewhere in this limiting set.

This can be shown by considering the evolution of the arbitrary surface $S(t)$ of volume $V(t)$ given in Fig. 2.1. Consider the points on S as initial conditions for trajectories, and let them evolve for an infinitesimal time dt. Then S evolves into a new surface $S(t+dt)$, the volume $V(t+dt)$, and from there the rate of change of volume $\dot{V}$ can be written as,

$$V(t+dt) = V(t) + \int_S (\mathbf{f}.\mathbf{n}dt)dA \tag{2.2a}$$

$$\dot{V} = \int_S (\mathbf{f}.\mathbf{n})dA \tag{2.2b}$$

Using the divergence theorem, the above expression for $\dot{V}$ can be rewritten as,

$$\dot{V} = \int_V \nabla.\mathbf{f}dV \tag{2.3}$$

This divergence for the Lorenz system can be evaluated as follows,

$$\nabla.\mathbf{f} = \frac{\partial[\sigma(y-x)]}{\partial x} + \frac{\partial[x(\rho-z)-y]}{\partial y} + \frac{\partial[xy-\beta z]}{\partial z} \tag{2.4a}$$

$$\nabla.\mathbf{f} = -(\sigma+1+\beta) \tag{2.4b}$$

Thus the divergence for the Lorenz system is always less than zero. The corresponding equation for the evolution of volume, $V(t)$ becomes, $V(t) = V(0)e^{-(\sigma+1+\beta)t}$, which implies that the volumes in phase space shrink exponentially fast for the Lorenz system. This has important ramifications about the permissible solutions for the Lorenz system. Some of them can be stated (refer to [1] for proofs) as follows,

1. There are no quasiperiodic solutions of the Lorenz equations.
2. It is impossible for the Lorenz system to have either repelling fixed points or repelling closed orbits. (By repelling, we mean that all trajectories starting near the fixed point or closed orbit are driven away from it.)
3. From 2, we can say that all fixed points must be sinks or saddles, and closed orbits (if they exist) must be stable or saddle-like.

The fixed points for the Lorenz system can be easily evaluated using Eqs. 4.2a, 4.2b, and 4.2c. The origin $(x^*, y^*, z^*) = (0, 0, 0)$ is a fixed point for all values of the parameters. It is like the motionless state of the waterwheel. For $\rho > 1$, there is also a symmetric pair of fixed points $x^* = y^* = \pm\sqrt{\beta(\rho - 1)}, z^* = \rho - 1$. Lorenz called them C^+ and C^-. They represent left- or right-turning convection rolls. As $\rho \to 1$, C^+ and C^- coalesce with the origin in a pitchfork bifurcation, and so the origin is globally stable for $\rho < 1$. It is linearly-stable for other $\rho > 1$. The fixed points C^+ and C^- can be readily shown to be linearly stable for the range of values of parameter ρ, $1 < \rho < \rho_h$, where $\rho_h = \frac{\sigma(\sigma+\beta+3)}{(\sigma-\beta-1)}$, and corresponds to a value where C^+ and C^- looses their stability in a subcritical Hopf-bifurcation. For $\rho > \rho_h$, there are no attractors in the neighbourhood of C^+ and C^-. The

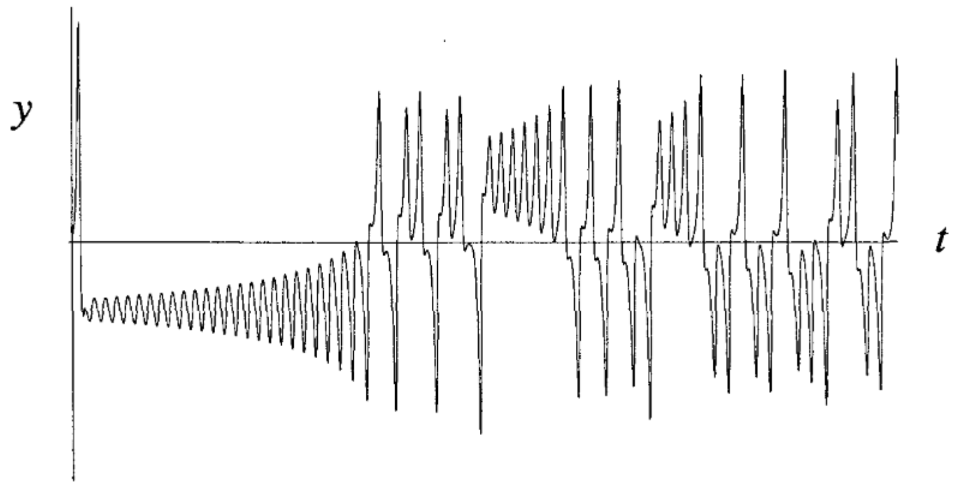


Figure 2.2: Temporal variation of y-component of Lorenz system of equations in the chaotic regime ($\rho = 28$). Adapted from ref. [1]

y-component of the solution is shown in Fig. 2.2, for $\rho = 28$, $\beta = 8/3$, and $\sigma = 10$. Note that, after an initial transient, the solution settles into an irregular oscillation that persists as $t \to \infty$, but never repeats exactly. The motion is aperiodic. If the solution is visualized as a trajectory in phase space, e.g. when x(t) is plotted against z(t), a butterfly pattern appears. When the trajectory is viewed in all three dimensions, rather than in a two-dimensional projection, it appears to settle onto an exquisitely thin set that looks like a pair of butterfly wings.

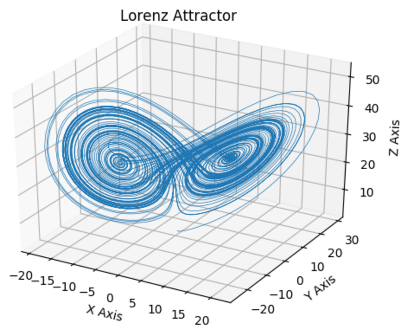


Figure 2.3: Chaotic attractor for the Lorenz system

This limiting set is the attracting set of zero-volume known as a strange attractor [9]. Fig. 2.3 shows a schematic of this strange attractor. Note that no trajectoriy in the strange-attractor intersect each other, and it is a set of points with zero volume but infinite surface area. In other words, a strange attractor has a fractal [10] structure. Numerical experiments suggest that it has a fractal dimension of about 2.05.

Finally, note that the motion on the chaotic-attractor exhibits sensitive dependence on initial conditions. This means that two trajectories starting very close to each other will rapidly diverge from each other, and thereafter have totally different futures. The practical implication for this is that long-term predictions become impossible in a chaotic-system. There is a time horizon beyond which prediction breaks down, as shown in Fig. 2.4. If we measure the initial conditions of an experimental chaotic system very accurately, no measurement is perfectthere is always an error between our estimate and the true initial state. After a time t, the discrepancy grows to $|\delta(t)|$ $|\delta_0|e^{\lambda t}$. If a is a measure of the tolerance, i.e., if a prediction is within a of the true state, we consider it acceptable. Then our prediction becomes intolerable when $|\delta(t)| \geq a$, and this occurs after a time,

$$t_{horizon} = \mathcal{O}(\frac{1}{\lambda}ln(\frac{a}{\delta_0})) \tag{2.5}$$

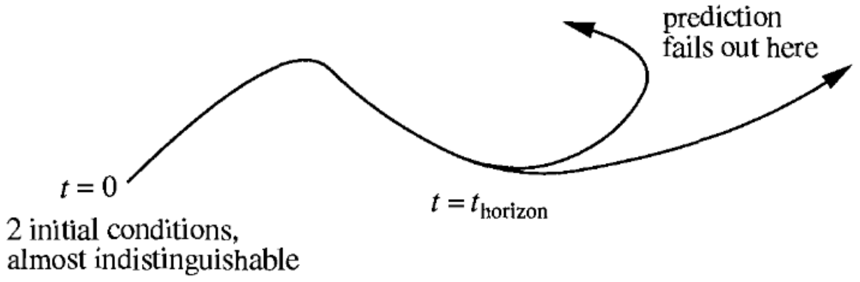

prediction fails out here
$t = 0$
2 initial conditions, almost indistinguishable
$t = t_{horizon}$

Figure 2.4: Schematic representing the concept of impossible long-term prediction in a chaotic scenario. Adapted from ref. [1].

where λ is known as the Lyapunov exponent, and can be evaluated for a dynamical system using different techniques [11]. Note that an n-dimensional system has n Lyapunov exponents, and exponent mentioned above is one of them corresponding to exponential divergence of initial conditions , i.e. a positive exponent.

2.2 CHAOS-CONTROL

A typical chaotic attractor has embedded within it an infinite number of unstable periodic orbits. Thus, one major advantage of chaos is that a rich variety of dynamical behaviors become available without the need for large controls, or for designing separate systems for each of these behaviors.

The idea of the chaos-control methods discussed here is to identify unstable periodic orbits inside the chaotic attractor and to stabilise them by slightly modifying some control parameters. The main idea is to show the strategy for control, which can be used in other control goals (such as transient chaos control, which we describe later).

There exists a huge list of experiments [12] where chaos control has been applied successfully. Examples include laser systems, chemical and mechan- ical systems, a magneto-elastic ribbon, and several others. Additionally, there are claims that the same mechanism also works in the control of biological systems such as the heart or the brain.

2.2.1 OGY method

In the Ott, Grebogi and Yorke (OGY) method [13] of chaos-control, , the goal is to obtain improved performance and a desired attracting time-periodic motion by making only small time-dependent perturbations in an accessible sys- tern parameter. This is done as follows,

1. An n-dimensional map , $Z_{n+1} = f(Z_n, p)$, is constructed. Here p is some accessible system parameter which can be changed in some small neighborhood of its nominal value p^*. In the case of continuous-time systems, such a map can be constructed by introducing a transversal surface of section for system trajectories (Poincaré map). For this value of p^*, there is a periodic orbit within the attractor around which one would like to stabilize the system.

2. If a fixed point, Z_f, is chosen, then the dynamics around If F is a chosen fixed point of the map f of the system existing for the parameter value p^*, then, in the close vicinity of this fixed point, we can assume that the dynamics are linear and can be expressed approximately by,

$$Z_{n+1} - Z_f = \mathbf{M}(Z_{n+1} - Z_f) \tag{2.6}$$

The elements of the matrix $\mathbf{M}$ can be calculated using the measured chaotic time series and analyzing its behavior in the neighborhood of the fixed point. Further, the eigenvalues λ_s,λ_u, and eigenvectors e_s, e_u. of this matrix can be found. The eigenvectors determine the stable and unstable directions in the small neighborhood of the fixed point. A linear approximation valid for small $|p - p^*|$ can be written as,

$$Z_{n+1} = p_n g + (\lambda_n e_n f_n + \lambda_s e_s f_s)(Z_n - p_n g) \tag{2.7}$$

Here f_s, f_u, are the contravariant eigenvectors ($f_s e_s = f_u e_u = 1$, $f_s e_u = f_u e_s = 0$), and $g = \frac{\partial Z_f}{\partial p}$. Because Z_{n+1} should fall on the stable manifold of Z_f, p is chosen , such that $f_u Z_{n+1} = 0$ giving us,

$$p_n = \frac{\lambda_n Z_n f_u}{(\lambda_u - 1) g f_u} \tag{2.8}$$

Note that no model of dynamics is required in this method. However, there are certain limitations of the OGY method. OGY method requires a permanent computer analysis of the state of the system. However, the changes in parameter are only discrete (depending on map). Thus, only the periodic-orbits with their maximal Lyapunov exponent small compared to the reciprocal of the time interval between parameter changes can be stabilized. Moreover, owing to rare and small perturbation of parameters, the control is quite sensitive to fluctuation noise. Another disadvantage of this method is that the control procedure can be applied only if the controlled trajectory is in the neighborhood of the appropriate unstable orbit or unstable fixed point. If far, the trajectory must be bought closer to the unstable periodic orbit before application of the main control, using a technique known as **targeting**. Some of these diadvantages are overcome in the Pyragas method [14] described next.

2.2.2 Delayed-Feedback control

In this method, a time-continous delayed feedback is applied to the dynamical system. The method can be formalized using the equations below,

$$\frac{dy}{dt} = P(y, \mathbf{x}) + F(t) \tag{2.9}$$

$$\frac{d\mathbf{x}}{dt} = \mathbf{Q}(y, \mathbf{x}) \tag{2.10}$$

Here, y is the output variable, and the output variable and the vector $\mathbf{x}$ describes the remaining variables of the dynamic system, and $F(t) = K(y(t-\tau) - y(t))$ is the delayed feedback. If time delay, τ, is equal to (or close to) the period of one of the unstable periodic orbits embedded in the chaotic attractor, the associated UPO can be stabilized for suitable values of K. Besides the noise-resistance of the controller , and it sismplicity, the major advantage of this control strategy is that it stabilizes an original solution of the system, and hence no (or very little, depending on the choice of delay) external energy is required to maintain the control goal once it has been attained.

2.3 TRANSIENT CHAOS

The appearance of chaos with finite lifetime is known as transient chaos. It is an example of a nonequilibrium state that is different from the asymptotic state, and cannot thus be understood from the asymptotic behavior alone. In such case, one observes a moving around of the system in an apparently chaotic manner and then, often rather suddenly, a settling down to a steady state which is either a periodic or a chaotic motion, but is of different type than the transients. Transient chaos finds relevance [15] in different areas such as, transients in chemical reactions in closed containers when reaction is started from a far from equilibrium state, in modeling of certain epidemiological data, e.g., on the spread of chickenpox, shimmy (an irregular dancing motion) of the front wheels of motorcycles and airplanes, satellite encounters and the escapes from major planets, trapping of advected material or pollutant around obstacles, in the classical dynamics of electrons in nanostructures, and so forth. More details about transient chaos and its control will be discussed in Chapter 4.

CHAPTER 3
REINFORCEMENT LEARNING

Reinforcement learning [16] is the third paradigm of machine learning; supervised and unsupervised learning being the other two. Unlike the latter two approaches reinforcement learning is much more focused on goal- directed learning from interaction with the environment. Reinforcement learning is learning what to do, how to map situations to actions so as to maximize a numerical reward signal. The learner is not told which actions to take, but instead must discover which actions yield the most reward by trying them. Since it first appeared in the 1950s, reinforcement learning has produced many different applications over the past decades. However, its revolution started a couple of years ago when the researchers from DeepMind (an English startup) pre- sented to the world some interesting ideas [17]. DeepMind presented a system that was able to play any Atari game without prior knowledge or experience. With the help of raw pixels as only inputs, this system outperformed the world champion of the GO game. Since then Reinforcement learning has burgeoned its presence [18] in different fields ranging from playing games to finance. This chapter is divided into four sections. In Sec. 3.1, we discuss the key concepts, terminologies, and basic formulations for reinforcement learning (RL). In, Sec. 3.2, a brief introduction to kinds of RL algorithms is given. Sec. 3.3 discusses policy-gradient methods in RL, and describes the working of some key RL algorithm belonging to this class.

3.1 BASICS OF REINFORCEMENT LEARNING

The main characters of RL are the agent and the environment. The environment is the world that the agent lives in and interacts with. At every step of interaction, the agent sees a (possibly partial) observation of the state of the world, and then decides on an action to take. The environment changes when the agent acts on it, but may also change on its own.

The agent also perceives a reward signal from the environment, a number that tells it how good or bad the current world state is. The goal of the agent is to maximize its cumulative reward, called return. Reinforcement learning methods are ways that the agent can learn behaviors to achieve its goal.

To be more formal, we need to be discuss key components of a typical RL problem – states and observations, action spaces, policies, trajectories, different formulations of return, the RL optimization problem, and value functions.

3.1.1 States and Observations

State s of a world is a complete description about it, without any hidden information, whereas an observation o is partial description of a state, and may omit information. In deep RL, the states and observations are mostly represented by a real-valued vector, matrix, or higher-order tensor. For example, a visual observation could be represented by the RGB matrix of its pixel values; the state of a robot might be represented by its joint angles and velocities.

Based on the information of the state with the agent, an environment can be categorized. When the agent is able to observe the complete state of the environment, the environment is said to be fully observed. When the agent can only see a partial observation, the environment is said to be partially observed.

3.1.2 Action spaces

Different environments allow and require different kinds of actions. The set of all valid actions in a given environment is often called the action space. Some environments have discrete action spaces, where only a finite number of moves are available to the agent. Examples of discrete action environments include the environments for games of Atari and

Go. Other environments have continuous action spaces, and actions are real-valued vectors. This is the case, for instance, when the agent controls a robot in a physical world.

This distinction in action-spaces plays an important role in the choice of RL algorithm. Some families of algorithms can be directly applied in one case, and need substantial reworking for the other case.

3.1.3 Policies

A policy is a rule used by an agent to decide what actions to take. A policy can be deterministic (denoted by $a_t = \mu(s_t)$) or stochastic(denoted by $a_t \sim \pi(\cdot|s_t)$)

In deep RL, policies are often computable functions that depend on a set of parameters (e.g. the weights and biases of a neural network) which can be adjusted to change the behavior using some optimization algorithm. The parameters of such a policy are often represented by θ or ϕ as a subscript on the policy symbol, and is denoted as,

$$a_t = \mu_\theta(s_t) \tag{3.1}$$

$$a_t \sim \pi_\theta(\cdot|s_t) \tag{3.2}$$

In the following, we discuss stochastic policies in some more detail,

Stochastic Policies

There are two kinds of stochastic policies in deep RL: categorical policies and diagonal Gaussian policies. Whereas the former can be used in discrete action spaces, the later policy type is used in continuous action spaces. In order to use and train stochastic policies, two key computations are centrally important: (a) sampling actions from the policy, and (b) computing log likelihoods of particular actions, $\log \pi_\theta(a|s)$.

Categorical policy is like a classifier over discrete actions and the neural network for a categorical policy is build the same way as for a classifier: the input being observation, followed by hidden layers (convolutional or densely-connected), and then one final linear layer that gives logits for each action, followed by a softmax to convert the logits into probabilities. Given the probabilities for each action, sampling is straightforward. In order to calculate the Log-Likelihood, the last layer of probabilities in the network is denoted as as $P_\theta(s)$, and the log-likelihood for an action a given as,

$$\log \pi_\theta(a|s) = \log \left[P_\theta(s)\right]_a \tag{3.3}$$

Note that $P_\theta(s)$ is a vector with as many entries as there are actions.

A diagonal Gaussian policy [19] always has a neural network that maps from observations to mean actions, $\mu_\theta(s)$. A multivariate Gaussian matrix, however, comprises of two parts:a mean vector and a standard deviation vector. Thus, we need to take care of the standard deviation vector as well. The standard deviation vector can either be a single vector of log standard deviations, $\log \sigma$, which is not a function of state, or with a separate network that maps from states to log standard deviations, $\log \sigma_\theta$. With this information, the action a can be sampled as follows,

$$a = \mu_\theta(s) + \sigma_\theta(s) \odot z, \tag{3.4}$$

Here $\odot$ denotes the elementwise product of two vectors, and z is a vector of noise sampled from a spherical Gaussian [20]. The log-likelihood of a k -dimensional action a, for a diagonal Gaussian with mean $\mu = \mu_\theta(s)$ and standard deviation $\sigma = \sigma_\theta(s)$, is computed as follows,

$$\log \pi_\theta(a|s) = -\frac{1}{2}\left(\sum_{i=1}^{k}\left(\frac{(a_i - \mu_i)^2}{\sigma_i^2} + 2\log \sigma_i\right) + k \log 2\pi\right) \tag{3.5}$$

3.1.4 Trajectories

A trajectory $\tau = (s_0, a_0, s_1, a_1, ...)$, also known as a episode [16], is a sequence of states and actions in the world. Note that in order to evaluate the state s_{t+1} from state s_t, a state-transition function is required. It depends on the natural laws of the environment, and on the most recent action, a_t. It can be either deterministic(transition represented as $s_{t+1} = f(s_t, a_t)$) or stochastic (transition represented as $s_{t+1} \sim P(\cdot|s_t, a_t)$). Note that the very first state s_0, needs to be randomly sampled from a start-state distribution, i.e. $s_0 \sim \rho_0$.

3.1.5 Rewards and the reinforcement learning problem

The notion of reward is very important in RL, as the goal of RL agent is to maximize the cumulative reward over a trajectory, τ. The reward r_t, in general depends on the current state of the world, the action just taken, and the next state of the world,, given as,

$$r_t = R(s_t, a_t, s_{t+1}) \tag{3.6}$$

The agent can receive either of two returns from the environment; (a) finite-horizon undiscounted return, which is the sum of rewards obtained in a fixed window of steps:

$$R(\tau) = \sum_{t=0}^{T} r_t. \tag{3.7}$$

or (b) infinite-horizon discounted return, which is the sum of all rewards ever obtained by the agent, but discounted by how far off in the future theyre obtained, using a discount factor $\gamma \in (0, 1)$:

$$R_\gamma(\tau) = \sum_{t=0}^{\infty} \gamma^t r_t. \tag{3.8}$$

Intuitively, a discount factor [16] accounts for the fact that cash now is better than cash later. Mathematically, it facilitates the convergence of infinite-horizon discounted return to a finite value.

Irrespective of the choice of the return measure (infinite-horizon discounted, or finite-horizon undiscounted reward), and the choice of policy, the goal in RL is to select a policy which maximizes expected return when the agent acts according to it. Supposing both the environment transitions and the policy to be stochastic, the probability of a T-step trajectory can be written as,

$$P(\tau|\pi) = \rho_0(s_0) \prod_{t=0}^{T-1} P(s_{t+1}|s_t, a_t)\pi(a_t|s_t). \tag{3.9}$$

The expected return, $J(\pi)$, can then be written as,

$$\begin{aligned} J(\pi) &= \int_{\tau} P(\tau|\pi)R(\tau) && (3.10) \\ &= \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)]. && (3.11) \end{aligned}$$

Mathematically, the central-optimization problem in RL can be reduced to the following,

$$\pi^* = \arg\max_{\pi} J(\pi), \tag{3.12}$$

Here π^* is the optimal policy.

3.1.6 Value Functions

In order to know the value of a state, or a state-action pair, value functions are used. They give the expected return when starting with some state or state-action pair, and then acting according to a particular policy afterwards. Four different Value functions can be defined as follows,

1. The On-Policy Value Function, $V^{\pi}(s)$, gives the expected return when starting with state s and always acting according to policy π,

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)\,|s_0 = s] \tag{3.13}$$

2. The On-Policy Action-Value Function, $Q^{\pi}(s,a)$, gives the expected return when starting in state s, taking an arbitrary action a (which may not have come from the policy), and then forever after acting according to policy π,

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)\,|s_0 = s, a_0 = a] \tag{3.14}$$

3. The Optimal Value Function, $V^*(s)$, gives the expected return when starting in state s and always acting according to the optimal policy in the environment,

$$V^*(s) = \max_{\pi} \mathop{\mathbb{E}}_{\tau\sim\pi}[R(\tau)\,|s_0 = s] \tag{3.15}$$

4. The Optimal Action-Value Function, $Q^*(s,a)$, gives the expected return when starting in state s, taking an arbitrary action a, and then forever after acting according to the optimal policy in the environment,

$$Q^*(s,a) = \max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} [R(\tau) \,| s_0 = s, a_0 = a] \tag{3.16}$$

Note that given Q^*, the optimal action, $a^*(s)$, can be directly obtained using,

$$a^*(s) = \arg\max_a Q^*(s,a). \tag{3.17}$$

The four value functions defined obey what are known as, **Bellman equations**, which are simple self-consistency equations. In a nutshell, Bellman equations underlie the fact that the value of starting point is the reward one expects to get from being there, plus the value of wherever one lands next, and can be written as follows for the on-policy value functions,

$$\begin{aligned} V^{\pi}(s) &= \mathop{\mathbb{E}}_{a \sim \pi, s' \sim P} r(s,a) + \gamma V^{\pi}(s'), && (3.18) \\ Q^{\pi}(s,a) &= \mathop{\mathbb{E}}_{s' \sim P} r(s,a) + \gamma \mathop{\mathbb{E}}_{a' \sim \pi} Q^{\pi}(s',a') && (3.19) \end{aligned}$$

where $s' \sim P$ is shorthand for $s' \sim P(\cdot|s,a)$, and indicates that the next state s' is sampled from the environments transition rules; $a \sim \pi$ is shorthand for $a \sim \pi(\cdot|s)$; and $a' \sim \pi$ is shorthand for $a' \sim \pi(\cdot|s')$.

The Bellman equations for the optimal value functions are

$$\begin{aligned} V^*(s) &= \max_a \mathop{\mathbb{E}}_{s' \sim P} r(s,a) + \gamma V^*(s'), && (3.20) \\ Q^*(s,a) &= \mathop{\mathbb{E}}_{s' \sim P} r(s,a) + \gamma \max_{a'} Q^*(s',a'). && (3.21) \end{aligned}$$

Note that the major difference between the Bellman equations for the on-policy value functions and the optimal value functions, is the absence or presence of the $\max$ over actions which reflects the fact that, in order to act optimally, it must pick whichever action leads to the highest value.

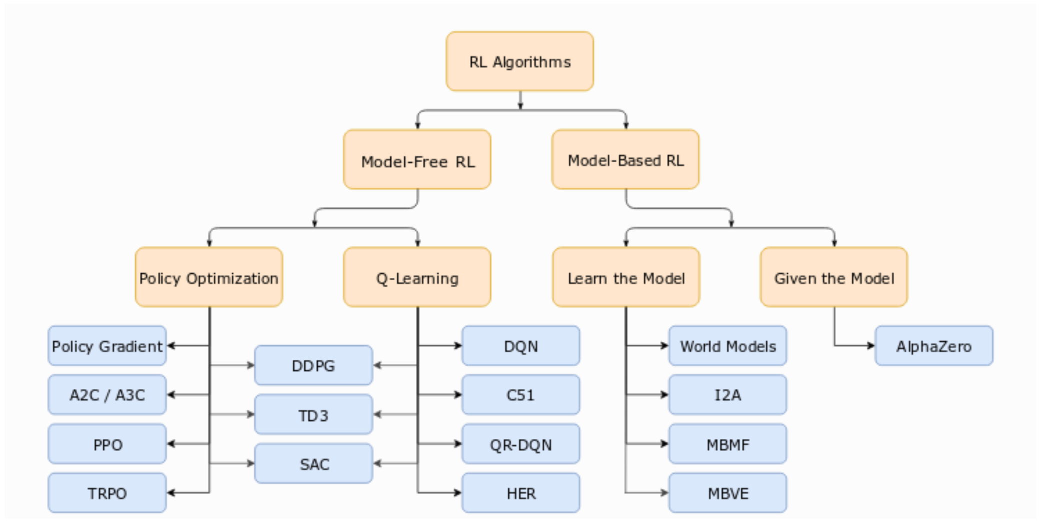


Figure 3.1: Classification of different reinforcement learning algorithms by OpenAI

Finally, a variant of value functions that is often used in RL is known as **Advantage function**. The advantage function $A^{\pi}(s,a)$ corresponding to a policy π quantifies the advantage in taking a specific action a in state s, over randomly selecting an action according to $\pi(\cdot|s)$, assuming that action is chosen as dictated by π forever after. Mathematically, the advantage function is defined as follows,

$$A^{\pi}(s,a) = Q^{\pi}(s,a) - V^{\pi}(s). \tag{3.22}$$

3.2 REINFORCEMENT LEARNING ALGORITHMS

The most important classification criterion for reinforcement learning algorithms is based on the fact whether the agent has access to (or learns) a model of the environment. Here a model of the environment implies a function which predicts state transitions and rewards. RL algorithms which use a model are known as **model-based methods**, and those that dont are called **model-free**. As shown in fig. 3.1, the former method includes popular algorithms like PPO (proximal policy optimization) and Q-learning. Examples in the later category includes algorithms like I2A (Imagination Augmented Agents), etc. In a model-based algorithms the agent enjoys significant gains in sampling efficiency as the presence of a model allows the agent to plan by thinking ahead, seeing what would happen for a range of possible choices, and hence giving the flexibility of deciding between its options. That said, ground-truth models of real problem environments are usually not available to the agent, and the agent should learn the model purely from experience, which is challenging, both computationally and fundamentally. The biggest challenge is that bias in the model can be exploited by the agent, resulting in an agent which performs well with respect to the learned model, but behaves badly in the real environment. Therefore, despite the advantages associated with model-based learning, model-free learning algorithms are most widely used owing to the associated flexibility and ease of implementation. In the following two subsections, we describe more details about what is learned in model based and model-free RL.

3.2.1 Model free algorithms

The two main approaches of representing and training agents with model-free RL are Policy optimization and Q learning.In the former approach, the parameters θ, that model a policy, $\pi_\theta(a|s)$, are optimized either directly by gradient ascent on the performance objective $J(\pi_\theta)$, or indirectly, by maximizing local approximations of $J(\pi_\theta)$. In these optimizations each update uses data collected while acting according to the most recent version of the policy, and hence the optimization is **on-policy**. Examples include A2C (Actor-critic) and PPO (proximal policy optimization). In the later approach, an approximator $Q_\theta(s, a)$ is learned for the optimal action-value function, $Q^*(s, a)$, and the optimization is almost always performed **off-policy** (i.e. each update can use data collected at any point during training.) as the objective function in these methods should just obey Bellman equations (and this is the optimization problem in these methods). The corresponding policy is obtained via the connection between Q^* and π^*. Examples of Q-learning methods include the popular DQN (Deep Q-network), an algorithm which launched Deen Reinforcement learning. The primary strength of policy optimization methods is that they directly optimize for the principle thing, i.e. optimal-policy, and thus these methods are stable and reliable. Q-learning methods, however, only indirectly optimize for agent performance (by training Q_θ to satisfy Bellman equation), and these methods are less stable [21]. Nevertheless, Q-learning methods are substantially more sample efficient when they do work, because they can reuse data more effectively than policy optimization techniques. Certain algorithms operate at the edge of Q learning and policy optimization, and carefully trade-off between the strengths and weaknesses of both algorithm types. Examples include, DDPG (deep deterministic policy gradient) and SAC (soft actor-critic).

3.2.2 Model based algorithms

There are different ways of using models. The simplest case is that of **Pure Planning**. Here the policy is never explicitly represented, and pure planning techniques like model-predictive control (MPC) are used to select actions. In MPC, each time the agent observes the environment, it computes a plan which is optimal with respect to the model, where the plan describes all actions to take over some fixed window of time after the present. The agent executes the first action of the plan, and discards the rest of it. A follow-on to pure planning involves using and learning a representation of the policy, $\pi_\theta(a|s)$, and using a planning algorithm (like Monte Carlo Tree Search) in the model, to generate actions for the plan by sampling from its current policy. In this way, the planning algorithm produces an action better than what the policy alone would have produced, thus justifying the name ”expert iteration.” AlphaZero [22], a computer program developed by artificial intelligence research company DeepMind to master the games of chess, shogi and go, is an example of this approach. Given the individual advantages of model-based and model-free learning, hybrid versions of the two approaches are also available. In one such approach, a model-free RL algorithm is used to train a policy or Q-function, but in order to update the agent certain fictitious-experiences are generated using a model to augment the real-world experiences which in turn reduces the sample complexity. e.g. MBVE (Model-based Value estimation) [23], and World Models [24].

3.3 POLICY OPTIMIZATION USING POLICY GRADIENTS

In this section, we elaborate on the theory of the operation of Policy optimization RL algorithms using a method known as Policy Gradients Method (PGM). After a basic discussion, two key Policy gradient algorithms are discussed, TRPO and PPO.

Here, . The eventual goal is to maximize the expected return $J(\pi_\theta) = \mathbb{E}_{\tau \sim \pi_\theta} R(\tau)$ by updating the parameters θ. Here, π_θ, is a stochastic, parameterized policy, and $R(\tau)$ is the finite-horizon undiscounted return. Following formalizes the optimization of the policy by gradient ascent,

$$\theta_{k+1} = \theta_k + \alpha \left. \nabla_\theta J(\pi_\theta) \right|_{\theta_k} . \tag{3.23}$$

When policy optimization is solely based on the formalism in Eq. 3.23, the method is known as **Vanilla Policy Gradient (VPG) [25]**. Note that the gradient of policy performance, $\nabla_\theta J(\pi_\theta)$, is called the policy gradient, and is the gradient of expected sum of rewards sampled over multiple trajectories, τ, given as follows,

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} R(\tau) && (3.24)\\
&= \nabla_\theta \int_\tau P(\tau|\theta) R(\tau) && (3.25)\\
&= \int_\tau \nabla_\theta P(\tau|\theta) R(\tau) && (3.26)\\
&= \int_\tau P(\tau|\theta) \nabla_\theta \log P(\tau|\theta) R(\tau) && (3.27)\\
&= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \nabla_\theta \log P(\tau|\theta) R(\tau) && (3.28)\\
\therefore \nabla_\theta J(\pi_\theta) &= \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) && (3.29)
\end{aligned}$$

Note that, here, $P(\tau|\theta) = \rho_0(s_0) \prod_{t=0}^{T} P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t)$ is the probability of a trajectory $\tau = (s_0, a_0, ..., s_{T+1})$ given that actions come from π_θ.

The above expectation can be estimated with a sample mean by collecting a set of trajectories $\mathcal{D} = \{\tau_i\}_{i=1,\ldots,N}$, where each trajectory is obtained by letting the agent act in the environment using the policy π_θ. The policy gradient $\hat{g}$ can then be estimated as follows,

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau). \tag{3.30}$$

where $|\mathcal{D}|$ is the number of trajectories in $\mathcal{D}$ (here, N).

Having this, we can compute the policy gradient and take an update step using Eq. 3.23.

Note that in the expression of the policy gradient, Eq. 3.29, log-probabilities of each action is pushed in proportion to $R(\tau)$, the sum of all rewards ever obtained. A better approach would be to consider rewards obtained after the relevant actions are taken. This can be expressed as follows,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}). \tag{3.31}$$

The reinforcing of actions proportional to oncoming rewards, instead of the past-rewards, all of which had zero mean, but nonzero variance reduces the noise in the estimate of policy gradient, and hence the number of smaple trajectories needed for a meaningful estimation of $\hat{g}$. Actually, the above policy gradient expressions can be replaced with a generalized form [26] given as,

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{\tau \sim \pi_\theta} \sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \Phi_t. \tag{3.32}$$

where Φ_t could be any of,

$$\Phi_t = R(\tau), \tag{3.33}$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}), \tag{3.34}$$

$$\Phi_t = \sum_{t'=t}^{T} R(s_{t'}, a_{t'}, s_{t'+1}) - b(s_t), \tag{3.35}$$

$$\Phi_t = Q^{\pi_\theta}(s_t, a_t), \tag{3.36}$$

$$\Phi_t = A^{\pi_\theta}(s_t, a_t) \tag{3.37}$$

Note that the first three choices for Φ_t lead to the same expected value for the policy gradient, however with different variances. The fourth choice corresponds to using On-Policy Action-Value Function for Φ_t, whereas the last choice corresponds to using the Advantage Function, $A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$, which describes how much better or worse the current action is than other actions on average. Such a formulation of policy gradients based on advantage functions is widely used.

3.3.1 Other Policy Gradient methods

Vanilla Policy gradient, discussed in the preceding section keeps new and old policies close in parameter space. However, even seemingly small differences in parameter space can have very large differences in performance, and a single bad step can collapse the policy performance. In order to prevent this, the updated policy should be kept close to the old policy. This is attained in two other methods, Trust Region Policy Optimization (TRPO)[27] and Proximal Policy Optimization (PPO)[28] using a surrogate advantage functional to update the policy as follows,

$$\theta_{k+1} = \arg\max_{\theta} L(\theta_k, \theta) \tag{3.38}$$

Along with above, TRPO uses second-order natural gradient optimization to update parameters within a trust-region of a fixed maximum Kullback-Leibler divergence between old and updated policy distribution. PPO wanes off the complexity in that approach by making use of an ingeniously designed objective given as,

$$L(\theta_k, \theta) = \underset{s,a \sim \pi_{\theta_k}}{\mathbb{E}} (min((\Pi(s, a, \theta_k, \theta) A^{\pi_{\theta_k}}(s, a)), g(\epsilon, A^{\pi_{\theta_k}}(s, a))) \tag{3.39}$$

where $\Pi(s, a, \theta_k, \theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)}$, and,

$$g(\epsilon, A) = \begin{cases} (1+\epsilon)A & A \geq 0 \\ (1-\epsilon)A & A < 0 \end{cases} \tag{3.40}$$

When the advantage $A^{\pi_{\theta_k}}(s, a) > 0$, taking action a in state s is preferable to the average of all actions that could be taken in that state, and it is natural to update the policy to favor this action. Similarly, when $A^{\pi_{\theta_k}}(s, a) > 0$, taking action a in state s represents a poorer choice than the average of all actions that could be taken in that state, and it is natural to update the policy to decrease the probability of taking this action. The above choice of L makes sure that such updates lies within the limits $(1 - \epsilon)$ and $(1 + \epsilon)$, where ϵ, known as the clipping parameter, is a hyper-parameter.

CHAPTER 4
RL ENABLED CONTROL OF TRANSIENT CHAOS

4.1 TRANSIENT CHAOS AND ITS CONTROL

Chaos is desirable and advantageous in many situations. For instance, in mechanics, exciting the chaotic motion of several modes spreads energy over a wide frequency range[29], thereby preventing undesirable resonance. Chaotic advection in fluids enhances mixing, as chaos brings about an exponential divergence of fluid packets that are initially in close proximity[30]. In biology, the absence of chaos may lead to an emergence of synchronous dynamics in the brain, which can result in epileptic seizures[31]. Moreover, the absence of chaos may also indicate the presence of other pathological conditions[32, 33].

In some cases, Chaos can become transient in nature, where the dynamics eventually converge to non-chaotic attractors. The typical route by which this happens is known as a crisis[34], where for certain parameter-values of the non-linear system, a chaotic-attractor collides with its basin-boundary and becomes a saddle. A saddle has a fractal structure with infinitely many gaps along its unstable-manifold. Any initial condition attracted towards this chaotic-attractor-turned-saddle escapes to an external periodic- or a fix-point-attractor. Such transient-chaos is often undesirable, and has been conjectured to be the culprit for phenomena such as voltage collapse in electric power systems[5] and species extinction in ecology[6]. It also plays a crucial role in governing the dynamics of shear flows in pipes and ducts at low Reynolds numbers[35, 36].

Given the importance of these phenomena, controlling transient-chaos is a pressing issue. Some attempts to restore chaos in such scenarios have been made in the past. [33] maintained chaos in transiently-chaotic regimes of one- and two-dimensional maps using small perturbations. Their method relied on accurate analytical knowledge of the dynamical system, and required a priori phase-space knowledge of escape regions from chaos. Another method utilized the natural dynamics around the saddle[37, 38], where small regions near a chaotic-saddle through which trajectories escape were identified. Then a set of “target” points in these regions were found, which yield trajectories that can stay near the chaotic saddle for a relatively long time. When the solution trajectory falls in this escape region, it is perturbed to the nearest target point so that the trajectory can persist near the chaotic saddle for a long time. The identification of such escape regions and target points can be challenging, and requires either an a priori computation of the probability distribution of escape times in different regions of state-space[37], or information from the return map constructed from local maxima or minima of a measured time series[38]. Such approaches become difficult for high-dimensional dynamical systems, and have been illustrated for 2D maps/flows at the most. One particular control technique that worked for the 3D Lorenz-system was described by[39]. The method was based on finding a certain control-perturbation set in the phase space, called a “safe set”, which avoids the escape of the trajectories to the fix-points. Identifying such a safe set can be prohibitively expensive computationally, and such safe sets may not exist for all dynamical systems. A useful alternative in such scenarios is to adopt control approaches which do not require a-priori knowledge of the system’s dynamics. Gadaleta and Dangelmayr[40] demonstrated an early application of such techniques, where they used Reinforcement Learning to stabilize unstable fixed points and periodic orbits in chaotic systems. We adopt a similar approach in this work, where a Reinforcement Learning-based autonomous controller continually perturbs the parameters of the Lorenz system to discover an optimal strategy for preventing transiently-chaotic behaviour of the system.

4.2 CONTROL ALGORITHM

In recent years, a machine-learning technique called deep Reinforcement Learning (RL) has shown great promise in control-optimization problems[17], and it has been successfully used to uncover complex underlying physics in Navier-Stokes simulations of fish-swimming[41]. The aim of the present work is to illustrate the utility of deep RL in determining small control-perturbations to the parameters of the Lorenz system[42], such that a sustained chaotic behavior is maintained despite the uncontrolled dynamics being transiently-chaotic. In doing so, no prior analytical knowledge about the dynamical system, and no special schemes to find escape regions, target points and safe sets will be employed. The RL algorithm is able to autonomously determine an optimal strategy to restore chaos, by continually interacting with the dynamical system.

As depicted in Fig. 4.1, a reinforcement learning problem consists of five major elements – a learning agent, an environment described by a model $\boldsymbol{Y}$ (the Lorenz system in our case), state-space S, action-space A, and reward r_t. Initially, the RL agent interacts with its environment in a trial-and-error manner. At each time step t, the agent receives the current state s_t of the environment, and selects an action a_t following a policy $\pi(a_t|s_t)$. This action allows the agent to perturb the state of the environment, and move to a new state s_{t+1} by evaluating the given model $\boldsymbol{Y}$ of the environment. Upon affecting this transition, the agent is rewarded (or punished) with reward r_t. This process continues until the agent reaches a terminal state, at which point a new episode starts over. The return received from each episode is the discounted cumulative reward with discount factor γ, which lies between 0 and 1. The discount factor makes it feasible to emphasize the importance of maximizing long-term reward, which enables the agent to prefer actions that are beneficial in the long-term. The cumulative reward, $R(a_t|s_t)$, is given as,

$$R(a_t|s_t) = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{4.1}$$

4.3 TRAINING THE AGENT TO SUSTAIN CHAOS

The goal of the RL agent is to maximize the cumulative reward by discovering an optimal policy π^*. There are a variety of methods available for attaining this. We make use of Proximal Policy Optimization (PPO)[28] which is a type of policy Gradient Method (PGM)[25, 16]. PPO is suitable for continuous-control problems[43], and it is simpler in its mathematical implementation compared to other PGM based RL algorithms[27]. Moreover, PPO requires comparatively little hyper-parameter tuning for use in a variety of different problems. The specific implementation of the algorithm that we used, PPO2, is available as part of the OpenAI stable-baselines library[44]. The ergodic and unsteady nature of chaotic dynamics necessitates the use of a version of PPO2 where the policy is defined by deep recurrent neural networks comprised of Long Short-Term Memory (LSTM) cells[45], instead of traditional feed-forward neural networks.

The environment for the Lorenz system is written in an OpenAI gym[46] -compatible python

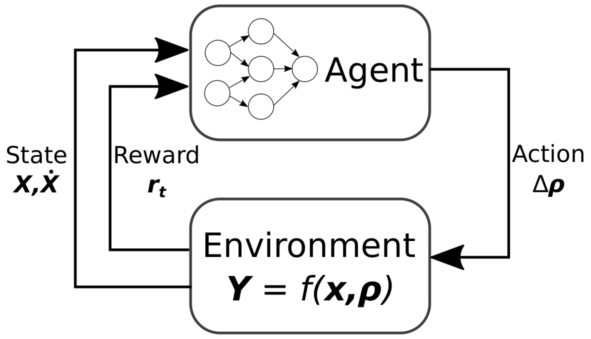

Agent
State
$\boldsymbol{X},\dot{\boldsymbol{X}}$
Reward
r_t
Action
$\Delta\boldsymbol{\rho}$
Environment
$\boldsymbol{Y} = f(\boldsymbol{x},\boldsymbol{\rho})$

Figure 4.1: Schematic illustrating the basic framework of a reinforcement learning problem. An agent continually perturbs the environment (which is the Lorenz system in our case) by taking an action, and records the resulting states. The agent is rewarded when a desirable state is reached, and punished otherwise.

format. The relevant equations are given as,

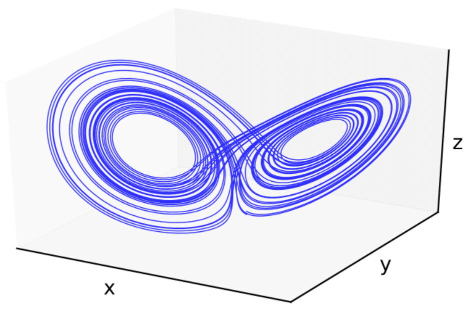
$$\frac{dx}{dt} = \sigma(y - x) \tag{4.2a}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{4.2b}$$

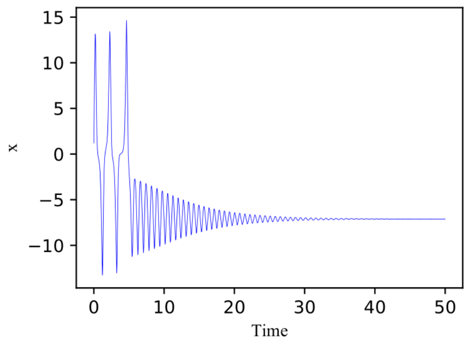
$$\frac{dz}{dt} = xy - \beta z \tag{4.2c}$$

With $\sigma = 10$ and $\beta = 8/3$, $\rho = 28$ gives rise to chaotic trajectories, whereas transient chaos is found in the interval $\rho \in [13.93, 24.06]$ [47]. Without any control implemented, the solution will converge to specific fix-points after a short transient, as shown in Fig. 4.2. The two fix-points in our case are given by $\mathrm{P}_+ = (7.12, 7.12, 19)$ and $\mathrm{P}_- = (-7.12, -7.12, 19)$.

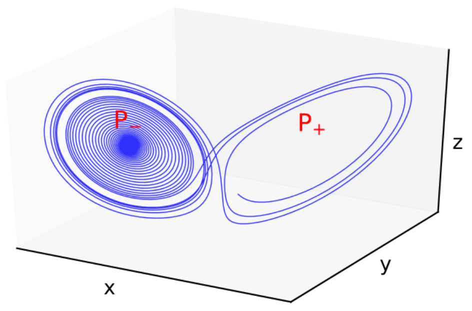
We use reinforcement learning to prevent such a transient from chaotic- to fix-point solutions. This is done by perturbing the parameters in Eqs. 4.2 ($\boldsymbol{\rho} = (\sigma, \rho, \beta)$) by $\Delta\boldsymbol{\rho} = (\Delta\sigma, \Delta\rho, \Delta\beta)$, with $\Delta\boldsymbol{\rho} \in [-\boldsymbol{\rho}/10, \boldsymbol{\rho}/10]$. We clarify that these limits do not change with the 'current' value of $\boldsymbol{\rho}$, but instead remain fixed at the initial value. The instantaneous value of the solution vector $\boldsymbol{X}(t) = (x, y, z)$ and its time-derivative (velocity) $\dot{\boldsymbol{X}}(t) = (V_x(t), V_y(t), V_z(t)) = (\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt})$ constitute the state space S for the RL algorithm. For training the RL agent to retain a chaotic trajectory, we utilize the fact that $|\boldsymbol{V}(t)|$ will decrease consistently as the solution converges to one of the fix-points, eventually becoming zero. On the other hand, $|\boldsymbol{V}(t)|$ will have a non-zero average value when the solution traces the chaotic attractor. Thus, whenever the agent determines suitable action values $\Delta\boldsymbol{\rho}$ for which $|\boldsymbol{V}(t)|$ is maintained above the predefined threshold value $V_0 = 40$,



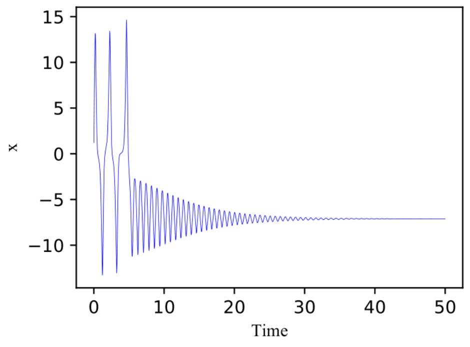
(a)



(b)



(c)



(d)

Figure 4.2: Solution of the Lorenz system of equations in (a,b) the chaotic regime with $\rho = 28$, and (c,d) the transiently-chaotic regime with $\rho = 20$. Panels (a) and (c) depict the solutions in phase space, and the corresponding time-variation of the x coordinate is shown in panels (b) and (d). Note that the solution traverses a chaotic trajectory for $\rho = 28$, whereas it converges to P_- after a few chaotic transients for $\rho = 20$.

it is rewarded, otherwise it is punished. In doing so over several iterations, the agent eventually learns to keep the trajectory chaotic. We remark that the method is robust against the choice of V_0 unless a very small value of V_0 is chosen. The strategy used here for selecting an appropriate V_0 is to use a value close to the ensemble average of velocity magnitude sampled from various instances of chaotic-transients.

The reward allocated to the agent consists of two parts: a stepwise reward r_t provided at each time step, and a one-time terminal reward $r_{terminal}$ given at the end of each episode. The two terms take the following form,

$$r_t = \begin{cases} 10 & V(t) > V_0 \\ -10 & V(t) \leqslant V_0 \end{cases} \tag{4.3a}$$

$$r_{terminal} = \begin{cases} -100 & \bar{r}_t < -2 \\ 0 & \bar{r}_t > -2 \end{cases} \tag{4.3b}$$

The average $\bar{r}_t$ is defined over the last 2000 time steps of an episode, and facilitates learning to keep the trajectory chaotic over long periods of time. The training of the agent is divided into episodes of 4000 time steps each, with time step size $dt = 2e{-}2$. The RL agent is expected to learn suitable action values $\Delta\boldsymbol{\rho}$ for any state permissible by the system environment, such that the long-term reward accumulated is maximized.

The neural network architecture used for training the RL agent is shown in Fig. 4.3. The network consists of an input layer, an output layer, and three hidden layers in between. Fig. 4.4 illustrates the training of the agent with time. The underlying neural network is trained for 2×10^5 time steps, which corresponds to 50 independent episodes in total, with each episode beginning with random values of the state variables $\boldsymbol{X}$ between -40 and 40; the corresponding values for $\dot{\boldsymbol{X}}$ are determined using the Lorenz equations (Eqs. 4.2). Initially, the solution keeps converging to the fix-points, since the network is unable to provide optimal action-decisions. After the network has trained for some time, it successfully learns the optimal actions for keeping the value of $|\boldsymbol{V}(t)|$ above V_0. As a consequence, the agent learns that the best way of maximizing reward is by maintaining the dynamics over the chaotic-attractor, which, although non-attracting for the given set of parameters, is a coexisting attractor of the system, along with the fix-points. In Fig. 4.4(b), we observe that

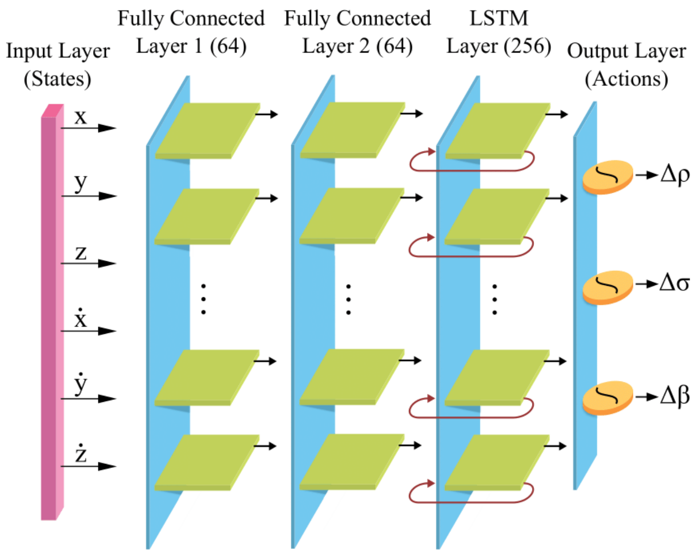
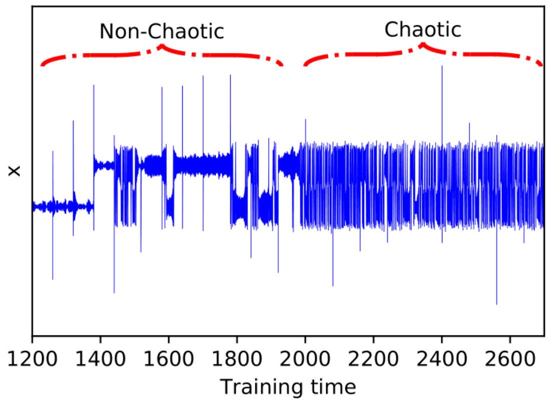


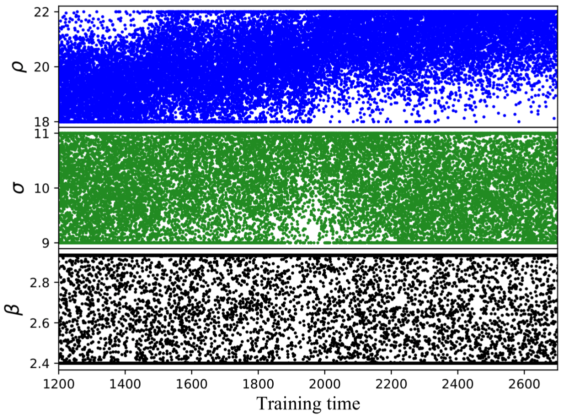
Figure 4.3: A schematic diagram of the neural network architecture used in our study. The 6 state variables feed in to a network with 2 fully connected layers consisting of 64 nodes each, followed by an LSTM layer comprised of 256 cells. The output layer has 3 nodes corresponding to the three possible actions. Hyperbolic tangent ($tanh$) activation is used throughout the network.

the autonomous controller learns a distinct trend for ρ which eventually suppresses the transiently-chaotic behavior beyond $t = 2000$. However, the other two parameters σ and β do not exhibit a notable pattern. This indicates that the controller focuses primarily on the parameter that is best able to restore chaotic dynamics.

We note that the training procedure for the present study is not extremely demanding in terms of computational cost; a complete training run requires approximately 10 minutes on a regular laptop computer. The main difficulty arises when deciding the most appropriate form of the reward function, since variations in the formulation can lead to notably different outcomes.

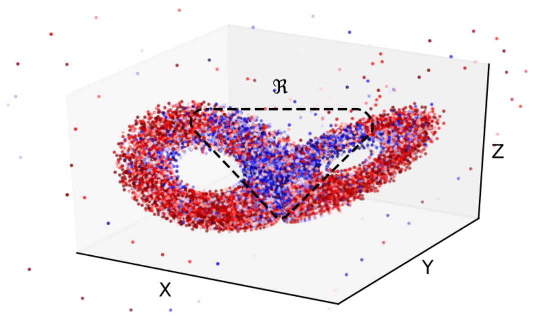


(a)

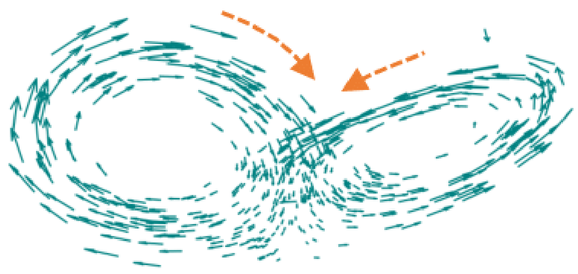


(b)

Figure 4.4: (a)Training of the RL agent with time. Note that the trajectory is bluetransiently-chaotic until around $t = 2000$, beyond which the agent learns to take effective decisions to keep the solution-trajectory chaotic for further instances. (b) Time-variation of the controlled parameters ρ, σ, and β. Note the gradual transition of the mean value of ρ, which corresponds with the eventual switch to chaotic behaviour.



(a)



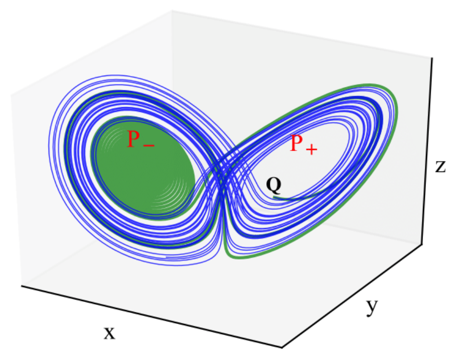
(b)

Figure 4.5: (a) Distribution of the perturbation parameter $\Delta\rho$ learned by the RL agent to keep the dynamics on the chaotic-attractor. The red dots indicate locations where the perturbation values are positive, and the blue dots correspond to negative values. (b) Velocity vector $(\frac{dx}{dt}, \frac{dy}{dt}, \frac{dz}{dt})$ plot shown for the corresponding solution in the state space. Note that $\Delta\rho$ is predominantly negative in the region $\Re$ where $V_z = \frac{dz}{dt} < 0$.

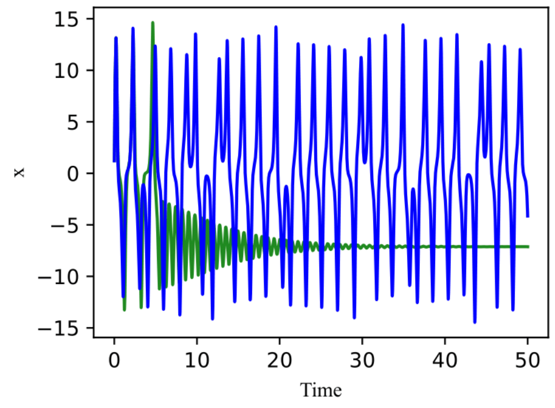
4.4 CONTROL STRATEGY DISCOVERED BY THE AUTONOMOUS AGENT

Figure 4.5 shows the distribution of the perturbation variable $\Delta\rho$ employed by the trained agent, which allows it to keep the dynamics on the chaotic-attractor. This distribution was obtained by plotting the controlled-trajectories for 400 random initial values for the variables x, y and z, lying between -40 and 40. Note that a similar distribution was obtained for the other perturbation variables $\Delta\beta$ and $\Delta\sigma$. However, we find that an execution of the converged RL control-policy with $\Delta\beta$ and $\Delta\sigma$ explicitly set to zero does not make a difference in the control outcome; the agent is still able to maintain a chaotic trajectory. This may be attributed to the dominating magnitude of the parameter ρ compared to the other two parameters. The significance of ρ is also evident in Fig. 4.4(b) where a transition in the mean value of ρ from approximately 20 to 21 is observed when the RL agent becomes effective at keeping the dynamics chaotic. No such transition is evident for β and σ. We note that a mean value of 21 for ρ does not correspond to the stable chaotic regime of the Lorenz system, and solving the equations with constant $\rho = 21$ would still lead to transiently chaotic behaviour. From the distribution shown in Fig. 4.5(a), we observe that the perturbation values for $\Delta\rho$ are predominantly negative in the region $\Re$, where $V_z = dz/dt < 0$, and positive elsewhere. We observe this correspondence by visualizing the velocity vectors in Fig. 4.5(b), which indicate that the direction of motion within the region $\Re$ almost always results in a decrease of the z coordinate, i.e., $V_z < 0$. The overall effect of this control mechanism is to prevent the trajectory from spiralling in to the fix-points.

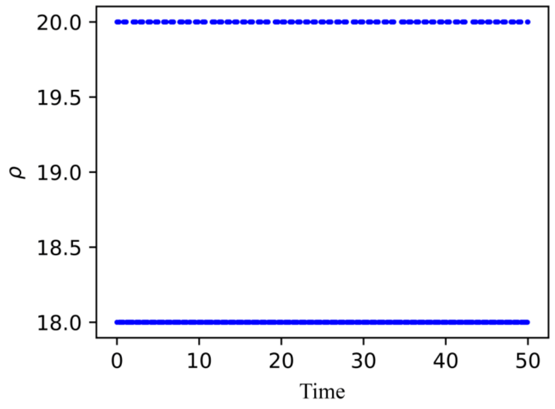
Note that unlike other control-techniques, RL-based control requires no a-priori analytical knowledge about the dynamical system regarding its escape-regions, target-points and safe-sets. The RL agent learns an optimal strategy π^* to prevent the transition from chaotic to fix-point solutions completely autonomously, by continually interacting with the environment defined by the Lorenz system of equations exhibiting transient-chaos.



(a)



(b)



(c)

Figure 4.6: (a,b) Comparison of the trajectory with and without the application of rule-based control. The blue trajectory corresponds to the controlled solution, starting from the initial condition $\mathbf{Q} = (1, 12, 9)$. The green uncontrolled solution starts from the same initial condition, and spirals in to the fix-point P_{-}. (c) Time-variation of ρ for the rule-based control approach.

4.5 REVERSE ENGINEERING A CONTROL LAW

Based on the strategy of the RL controller, we formulate a simple rule-based controller which perturbs the parameter ρ by $-\rho/10$ whenever the trajectory visits the region $\Re$, i.e., whenever $V_z < 0$. All parameters remain unperturbed outside this region. The success of the rule-based binary-control is demonstrated in Fig. 4.6 (Multimedia view), where the uncontrolled trajectory (green) converges to the fix-point P_-, whereas its controlled counterpart (blue) remains chaotic. We remark that in the controlled scenario, ρ takes on two discrete values $\rho = 20.0$ and $\rho = 18.0$ (Fig. 4.6(c)), both of which lie within the range $\rho \in [13.93, 24.06]$ where transient chaos would be observed without active control. This demonstrates that autonomous strategies discovered by RL can be extremely useful for formulating simple control-laws in fairly complex dynamical systems.

CHAPTER 5

CONCLUSIONS AND SCOPE FOR FUTURE WORK

The utility of deep reinforcement learning in restoring chaos for a transiently-chaotic system was exhibited in this thesis. The learning algorithm autonomously discovered an effective strategy for perturbing the parameters of the Lorenz system to achieve its goal. The underlying strategy was analyzed in order to formulate a simple control-law, which is able to sustain the chaotic trajectory in the transiently-chaotic parametric regime. Since transient chaos is a consequence of a catastrophic bifurcation (crisis)[48], which underlie the formation and evolution of a complex topological structure (saddle), the results in this work show the effectiveness of RL in controlling complex dynamical systems undergoing such catastrophic transitions.

A particularly important engineering application of the control of transient chaos arises in wall bounded shear flows, such as the pipe-flows [49], where transient chaos and chaotic saddles play a vital role in governing the fluid dynamics [36, 35]. For instance, regulating transient chaos and evolution of the topology of saddles can minimize intermittent switching between turbulent and laminar states–and delay the onset of turbulence in such flows–thereby reducing drag force. Note that when dealing with transient chaos in pipe-flows, the nature of dynamics is spatio-temporal. Representative models for spatio-temporal chaos, such as coupled-map lattice [50, 51], can serve as appropriate environments for developing reinforcement learning based control algorithms for regulating transient-chaos in wall bounded shear-flows.

BIBLIOGRAPHY

[1] SH Strogatz. Nonlinear dynamics and chaos: With applications to physics, biology, chemistry, and engineering (cambridge, ma: Westview). 1994.

[2] Fred Attneave. Multistability in perception. *Scientific American*, 225(6):62–71, 1971.

[3] Alexander N Pisarchik and Ulrike Feudel. Control of multistability. *Physics Reports*, 540(4):167–218, 2014.

[4] Tamás Tél. The joy of transient chaos. *Chaos*, 25(9):097619, 2015.

[5] Ian Dobson and Hsiao-Dong Chiang. Towards a theory of voltage collapse in electric power systems. *Systems & Control Lett.*, 13(3):253–262, 1989.

[6] Kevin McCann and Peter Yodzis. Nonlinear dynamics and population disappearances. *The American Naturalist*, 144(5):873–879, 1994.

[7] Pierre Simon, Marquis de Laplace, and FW Truscott. *A philosophical essay on probabilities*, volume 166. Dover Publications New York, 1951.

[8] Henri Poincaré. *Les méthodes nouvelles de la mécanique céleste*, volume 3. Gauthier-Villars et fils, 1899.

[9] David Ruelle and Floris Takens. On the nature of turbulence. *Les rencontres physiciens-mathématiciens de Strasbourg-RCP25*, 12:1–44, 1971.

[10] Benoit B Mandelbrot. *The fractal geometry of nature*, volume 173. WH freeman New York, 1983.

[11] Holger Kantz. A robust method to estimate the maximal lyapunov exponent of a time series. *Phys. Lett. A*, 185(1):77–87, 1994.

[12] Eckehard Schöll and Heinz Georg Schuster. *Handbook of chaos control*. John Wiley & Sons, 2008.

[13] Edward Ott and Mark Spano. Controlling chaos. In *AIP Conference Proceedings*, volume 375, pages 92–103. AIP, 1996.

[14] Kestutis Pyragas. Continuous control of chaos by self-controlling feedback. *Phys. Lett. A*, 170(6):421–428, 1992.

[15] Ying-Cheng Lai and Tamás Tél. *Transient chaos: complex dynamics on finite time scales*, volume 173. Springer Science & Business Media, 2011.

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

[17] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[18] Andrew G Barto, PS Thomas, and Richard S Sutton. Some recent applications of reinforcement learning. In *Proceedings of the 18th Yale Workshop on Adaptive and Learning Systems*, 2017.

[19] Marc Peter Deisenroth, Carl Edward Rasmussen, and Jan Peters. Gaussian process dynamic programming. *Neurocomputing*, 72(7-9):1508–1524, 2009.

[20] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer, 2003.

[21] John N Tsitsiklis and Benjamin Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[22] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[23] Vladimir Feinberg, Alvin Wan, Ion Stoica, Michael I Jordan, Joseph E Gonzalez, and Sergey Levine. Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*, 2018.

[24] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[25] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 10571063, Cambridge, MA, USA, 1999. MIT Press.

[26] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

[27] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *ICML*, pages 1889–1897, 2015.

[28] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[29] Ioannis T Georgiou and Ira B Schwartz. The slow invariant manifold of a conservative pendulum-oscillator system. *Int. J. Bifurcation Chaos*, 6(04):673–692, 1996.

[30] Hermann Haken. Chaos and order in nature. In *Chaos and order in nature*, pages 2–11. Springer, 1981.

[31] J Chris Sackellares, Leontdas D Iasemidis, Deng-Shan Shiau, Robin L Gilmore, and Steven N Roper. Epilepsy–when chaos fails. In *Chaos in Brain?*, pages 112–133. World Scientific, 2000.

[32] Ary L Goldberger, David R Rigney, and Bruce J West. Chaos and fractals in human physiology. *Sci. Am.*, 262(2):42–49, 1990.

[33] Weiming Yang, Mingzhou Ding, Arnold J Mandell, and Edward Ott. Preserving chaos: Control strategies to preserve complex dynamics with potential relevance to biological disorders. *Phys. Rev. E*, 51(1):102, 1995.

[34] Celso Grebogi, Edward Ott, and James A Yorke. Crises, sudden changes in chaotic attractors, and transient chaos. *Physica D: Nonlinear Phenomena*, 7(1-3):181–200, 1983.

[35] Kerstin Avila, David Moxey, Alberto de Lozar, Marc Avila, Dwight Barkley, and Björn Hof. The onset of turbulence in pipe flow. *Science*, 333(6039):192–196, 2011.

[36] Tobias Kreilos, Bruno Eckhardt, and Tobias M Schneider. Increasing lifetimes and the growing saddles of shear flow turbulence. *Phys. Rev. Lett.*, 112(4):044503, 2014.

[37] Ira B Schwartz and Ioana Triandaf. Sustaining chaos by using basin boundary saddles. *Phys. Rev. Lett.*, 77(23):4740, 1996.

[38] Mukeshwar Dhamala and Ying-Cheng Lai. Controlling transient chaos in deterministic flows with applications to electrical power systems and ecology. *Phys. Rev. E*, 59(2):1646, 1999.

[39] Rubén Capeáns, Juan Sabuco, Miguel AF Sanjuán, and James A Yorke. Partially controlling transient chaos in the lorenz equations. *Philos. Trans. R. Soc. London, Ser. A*, 375(2088):20160211, 2017.

[40] S. Gadaleta and G. Dangelmayr. Optimal chaos control through reinforcement learning. *Chaos*, 9(3):775–788, 1999.

[41] Siddhartha Verma, Guido Novati, and Petros Koumoutsakos. Efficient collective swimming by harnessing vortices through deep reinforcement learning. *Proc. Natl. Acad. Sci. USA*, 115(23):5849–5854, 2018.

[42] Edward N Lorenz. *The essence of chaos*. University of Washington Press, 1995.

[43] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[44] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. `https://github.com/hill-a/stable-baselines`, 2018.

[45] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[46] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[47] James L Kaplan and James A Yorke. Preturbulence: a regime observed in a fluid flow model of lorenz. *Comm. Math. Phys.*, 67(2):93–108, 1979.

[48] Wen-Xu Wang, Rui Yang, Ying-Cheng Lai, Vassilios Kovanis, and Celso Grebogi. Predicting catastrophes in nonlinear dynamical systems by compressive sensing. *Phys. Rev. Lett.*, 106(15):154101, 2011.

[49] P Cvitanović and JF Gibson. Geometry of the turbulence in wall-bounded shear flows: periodic orbits. *Physica Scripta*, 2010(T142):014007, 2010.

[50] Hu Gang and Qu Zhilin. Controlling spatiotemporal chaos in coupled map lattice systems. *Phys. Rev. Lett.*, 72(1):68, 1994.

[51] Kunihiko Kaneko. Coupled map lattice. In *Chaos, Order, and Patterns*, pages 237–247. Springer, 1991.