BALANCE AND GAIT

A Project Report

submitted by

SHRITEJ CHAVAN (BE14B004)

in partial fulfilment of the requirements
for the award of the degree of

Dual Degree (B.Tech and M.tech)
in Biological Engineering

DEPARTMENT OF BIOTECHNOLOGY
BHUPAT AND JYOTI MEHTA SCHOOL OF BIOSCIENCES
INDIAN INSTITUTE OF TECHNOLOGY MADRAS.

June 2019

THESIS CERTIFICATE

This is to certify that the thesis titled **BALANCE AND GAIT**, submitted by **Shritej Chavan** , to the Indian Institute of Technology, Madras, for the award of the degree of **Dual Degree (B.Tech and M.Tech) in Biological Engineering**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. V. Srinivasa Chakravarthy
Research Guide
Professor
Dept. of Biotechnology
IIT-Madras, 600 036

Place: Chennai

Date: 12th July 2019

ACKNOWLEDGEMENTS

I would first like to thank Prof. V. Srinivasa Chakravarthy for his patience and support. I am extremely grateful to him for providing his valuable guidance and motivating me every step of the way. I am grateful for the confidence he instilled upon me during the course of the project.I learnt a great deal both academically and personally from him and I would be grateful for that forever. He is a great role model and his passion and sincerity towards this field has inspired many people including me.His constant support helped me stay on the track and proceed forward with right enthusiasm.

Secondly, I am grateful to Dipayan Biswas who with his constant feedback and support kept my project moving in the right direction.

Also, I would take this opportunity to thank my batch mates Rahul Ramesh, Saurabh Mathur and Abhinav Saradana for providing valuable inputs and insightful discussions throughout the course of my project.

Any acknowledgement will be incomplete without a mention of my parents . I am extremely grateful for their unceasing support, encouragement and love. They have always stood by me through my years of study. I would also like to place on my record my sense of gratitude for everyone who has helped me either directly or indirectly during the course of my degree. I apologise for being unable to mention all of them by name as without them this thesis would not have been possible.

ABSTRACT

KEYWORDS: Postural Sway, RL, DDPG, Actor, Critic

Implementation of a function approximator using Deep Reinforcement Learning algorithm like DDPG to design the neuronal controller which takes the low dimensional proprioceptive signal as input and produces desired torque at the joint of a 1DOF inverted pendulum, a crude approximation of the mechanical model of human being, to make it vertical with respect to ground and produce postural sway characteristics as seen in the healthy adults

TABLE OF CONTENTS

ACKNOWLEDGEMENTS i

ABSTRACT iii

LIST OF TABLES vii

LIST OF FIGURES x

ABBREVIATIONS xi

NOTATION xiii

1 INTRODUCTION 1

1.1 Sense of Balance 1

1.1.1 Sensory Input 2

1.1.2 Processing of conflicting sensory inputs 4

1.1.3 The coordinated balance system 5

1.2 Postural Sway in Humans 6

1.2.1 Quantification of Postural Sway during standing 8

2 Background 13

2.1 Motivation 13

2.2 Setting 14

2.3 Episodic and Continuing Tasks 15

2.4 Expected Discounted Return 16

2.5 Value Functions 16

2.6 Temporal Difference 17

2.6.1 One-Step Temporal Difference 18

2.6.2 Multi-Time Temporal-Difference 19

2.6.3 On-Policy, Off-Policy Methods 20

2.7 Actor - Critic Methods 20

2.7.1 Optimal Policy 20
2.7.2 Deterministic Actor-Critic 21

3 Deep Reinforcement Learning **23**
3.1 Neural Networks 23
3.1.1 Multilayer Perceptron 24
3.1.2 Regularization 26
3.1.3 Mini-Batch Methods 27
3.1.4 Batch Normalization 27
3.2 Experienced Relay 28
3.3 Prioritized Experienced Relay 28
3.4 Deep Deterministic Policy Gradient 29
3.4.1 Target Networks 30
3.4.2 Deterministic Policy Gradient 31

4 Methods **33**
4.1 RL model 33
4.1.1 Environment 34
4.1.2 Agent 34
4.2 Implementation 35
4.2.1 OpenAI Gym 35
4.2.2 Stable Baselines 36
4.3 Experimental Data 37

5 Results **39**
5.1 Model 1 39
5.2 Model 2 41

6 Future Work and Limitations **43**

LIST OF TABLES

1.1 Usual variables used in the global analysis of COP displacement and examples on how to compute these variables by using Matlab software 10

LIST OF FIGURES

1.1 Mean values for the limits of the base of support (solid line), limits of stability individuals can voluntarily reach during standing (dashed line), center of pressure (COP) area of sway during prolonged unconstrained standing for 30 min (dotted line), and COP area of sway during standing still for 40 s (filled area in the center). 7

1.2 ML : Mediolateral - Frontal Plane , AP : Anterioposterior - Sagittal Plane 8

1.3 Examples of statokinesigram (A) and stabilogram (B) of center of pressure (COP) displacement during standing as still as possible on a force plate. 9

1.4 Example of the power spectral density estimation of center of pressure (COP) displacement during quiet standing. The peak (Fpeak), mean (Fmean), and median (F50) frequencies and the frequency band that contains up to 80 10

1.5 Single inverted pendulum model for the representation of a human standing. COG , center of gravity; COG v , COG vertical projection in relation to the ankle joint; COP , center of pressure in relation to the ankle joint; m, g , body mass, acceleration of gravity; F x , F y , horizontal and vertical components of the resultant ground reaction force; T a , torque at the ankle joint; d , distance between the COG and ankle joint; h , height of the ankle joint to the ground; Îś , angle of the body. . . 11

2.1 Mario as a RL example 13

2.2 Interaction of Agent and Enviroment 14

3.1 MLP consisting of an input layer with three neurons, two hidden layers each with four neurons respectively, and an output layer with one neuron 24

4.1 Block of the RL model 33

4.2 'Inverted_Pendulum-v0': Direction of arrow shows the direction of torque applied 36

4.3 Sway of 4 Healthy Adults and their respective Frequency Spectrum 37

5.1 Sway Characteristics and Power Spectral Density 39

5.2 Action vs Time 40

5.3 Sway Characteristics and Power Spectral Density 41

5.4 Angular Velocity versus Time 42
5.5 Action vs Time 42

ABBREVIATIONS

RL	Reinforcement Learning
ANN	Artificial Neural Networks
MLP	Multilayer Perceptron
DDPG	Deep Deterministic Policy Gradient
SGD	Stochastic Gradient Descent
RMSProp	Root Mean Square Propagation
Adam	Adaptive Moment Estimation
ML	MedioLateral
AP	Anterioposterior
COM	Centre of Mass
DRL	Deep Reinforcement Learning
COP	Centre of Pressure
PD	Parkinson's Disease
COG	Centre of Gravity
TD	Temporal Difference

NOTATION

E	Environment
S	Continuous State Space
A	Action Space
$\mathbb{R}$	Domain of Real Numbers
s	Arbitrary State
a	Arbitrary Action
p	Probability Distribution
T	State Transition Function
r	Reward
$done$	binary terminal condition
π	Stochastic Policy
μ	Deterministic Policy
γ	Discount factor
R_t	Discounted Return
$\mathbb{E}$	Expectation
V	Value Function
Q	Action Value Function
δ	TD error
Θ	Parameters of Neural Networks
L	Loss function
w_i	Weights of each neuron
b	Bias
$\odot$	Element wise Multiplication

CHAPTER 1

INTRODUCTION

Human walking is unique, and each of us has a " fingerprint " (or footprint) walk unlike any other. Bipedal walking is almost strictly a human skill. The evolution of bipedalism occurred after the chimpanzee-human diversion about 6 million years ago and it evolved over a few million years so that by the time of the skeleton "Lucy" (3.2 million years ago), she was mostly upright, and by 2 million years ago, our ancestors became obligate bipeds. Bipedalism helped free the hands and possibly paved the way towards tool development and the eventual evolution of modern humans (Nielsen, 2003). Currently, we all still belong to 1 species, with our skeletons staying similar over the last 200,000 years, yet evolution is ongoing; for example, blue eyes evolved only few thousand years ago.

Balance is the ability to stand up and remain upright against the force of gravity (equilibrium), and gait is rhythmic stepping movements to advance in space (locomotion). All vertebrates, including humans, have a desired body orientation that they maintain (in animals it is dorsal side up, and in humans it is upright posture)(Grillner, 2008). Balance and gait abilities in man depend on a hierarchal system of muscles, nerves, and interconnected central nervous system structures that is amazingly similar to that seen all the way "down" to the snail (Sten Grillner, 1999). Walking is not an aimless process and is essential for survival of animal and human. For this reason, disorders of balance and gait have serious consequences to the afflicted. Because such disorders almost always coexist, they are discussed together, under the rubric of "balance and gait disorders".

1.1 Sense of Balance

Balance is achieved and maintained by a complex set of sensorimotor control systems that include sensory input from vision (sight), proprioception (touch), and the vestibular system (motion, equilibrium, spatial orientation); integration of that sensory input; and

motor output to the eye and body muscles. Injury, disease, certain drugs, or the aging process can affect one or more of these components. In addition to the contribution of sensory information, there may also be psychological factors that impair our sense of balance.

1.1.1 Sensory Input

Maintaining balance depends on information received by the brain from three peripheral sources: eyes, muscles and joints, and vestibular organs. All three of these information sources send signals to the brain in the form of nerve impulses from special nerve endings called sensory receptors.

Input from Eyes

Sensory receptors in the retina are called rods and cones. Rods are believed to be tuned better for vision in low light situations (e.g. at night time). Cones help with color vision, and the finer details of our world. When light strikes the rods and cones, they send impulses to the brain that provide visual cues identifying how a person is oriented relative to other objects. For example, as a pedestrian takes a walk along a city street, the surrounding buildings appear vertically aligned, and each storefront passed first moves into and then beyond the range of peripheral vision.

Input from the muscles and joints

Proprioceptive information from the skin, muscles, and joints involves sensory receptors that are sensitive to stretch or pressure in the surrounding tissues. For example, increased pressure is felt in the front part of the soles of the feet when a standing person leans forward. With any movement of the legs, arms, and other body parts, sensory receptors respond by sending impulses to the brain. Along with other information, these stretch and pressure cues help our brain determine where our body is in space.

The sensory impulses originating in the neck and ankles are especially important. Proprioceptive cues from the neck indicate the direction in which the head is turned. Cues from the ankles indicate the body 's movement or sway relative to both the stand-

ing surface (floor or ground) and the quality of that surface (for example, hard, soft, slippery, or uneven).

Input from Vestibular system

Sensory information about motion, equilibrium, and spatial orientation is provided by the vestibular apparatus, which in each ear includes the utricle, saccule, and three semicircular canals. The utricle and saccule detect gravity (information in a vertical orientation) and linear movement. The semicircular canals, which detect rotational movement, are located at right angles to each other and are filled with a fluid called endolymph. When the head rotates in the direction sensed by a particular canal, the endolymphatic fluid within it lags behind because of inertia, and exerts pressure against the canal 's sensory receptor. The receptor then sends impulses to the brain about movement from the specific canal that is stimulated. When the vestibular organs on both sides of the head are functioning properly, they send symmetrical impulses to the brain. (Impulses originating from the right side are consistent with impulses originating from the left side.)

Integration of Sensory Input

Balance information provided by the peripheral sensory organs such as eyes, muscles and joints, and the two sides of the vestibular system is sent to the brain stem. There, it is sorted out and integrated with learned information contributed by the cerebellum (the coordination center of the brain) and the cerebral cortex (the thinking and memory center). The cerebellum provides information about automatic movements that have been learned through repeated exposure to certain motions. For example, by repeatedly practicing serving a ball, a tennis player learns to optimize balance control during that movement. Contributions from the cerebral cortex include previously learned information; for example, because icy sidewalks are slippery, one is required to use a different pattern of movement in order to safely navigate them.

1.1.2 Processing of conflicting sensory inputs

A person can become disoriented if the sensory input received from his or her eyes, muscles and joints, or vestibular organs sources conflicts with one another. For example, this may occur when a person is standing next to a bus that is pulling away from the curb. The visual image of the large rolling bus may create an illusion for the pedestrian that he or she rather than the bus is moving. However, at the same time the proprioceptive information from his muscles and joints indicates that he is not actually moving. Sensory information provided by the vestibular organs may help override this sensory conflict. In addition, higher level thinking and memory might compel the person to glance away from the moving bus to look down in order to seek visual confirmation that his body is not moving relative to the pavement.

Motor Output

As sensory integration takes place, the brain stem transmits impulses to the muscles that control movements of the eyes, head and neck, trunk, and legs, thus allowing a person to both maintain balance and have clear vision while moving.

Motor output to the muscles and joint

A baby learns to balance through practice and repetition as impulses sent from the sensory receptors to the brain stem and then out to the muscles form a new pathway. With repetition, it becomes easier for these impulses to travel along that nerve pathway a process called facilitation and the baby is able to maintain balance during any activity. Strong evidence exists suggesting that such synaptic reorganization occurs throughout a person's lifetime of adjusting to changing motion environs.

This pathway facilitation is the reason dancers and athletes practice so arduously. Even very complex movements become nearly automatic over a period of time. This also means that if a problem with one sensory information input were to develop, the process of facilitation can help the balance system reset and adapt to achieve a sense of balance again.

For example, when a person is turning cartwheels in a park, impulses transmitted

from the brain stem inform the cerebral cortex that this particular activity is appropriately accompanied by the sight of the park whirling in circles. With more practice, the brain learns to interpret a whirling visual field as normal during this type of body rotation. Alternatively, dancers learn that in order to maintain balance while performing a series of pirouettes, they must keep their eyes fixed on one spot in the distance as long as possible while rotating their body.

Motor output to the eyes

The vestibular system sends motor control signals via the nervous system to the muscles of the eyes with an automatic function called the vestibulo-ocular reflex (VOR). When the head is not moving, the number of impulses from the vestibular organs on the right side is equal to the number of impulses coming from the left side. When the head turns toward the right, the number of impulses from the right ear increases and the number from the left ear decreases. The difference in impulses sent from each side controls eye movements and stabilizes the gaze during active head movements (e.g., while running or watching a hockey game) and passive head movements (e.g., while sitting in a car that is accelerating or decelerating).

1.1.3 The coordinated balance system

The human balance system involves a complex set of sensorimotor-control systems. Its interlacing feedback mechanisms can be disrupted by damage to one or more components through injury, disease, or the aging process. Impaired balance can be accompanied by other symptoms such as dizziness, vertigo, vision problems, nausea, fatigue, and concentration difficulties.

The complexity of the human balance system creates challenges in diagnosing and treating the underlying cause of imbalance. The crucial integration of information obtained through the vestibular, visual, and proprioceptive systems means that disorders affecting an individual system can markedly disrupt a person's normal sense of balance. Vestibular dysfunction as a cause of imbalance offers a particularly intricate challenge because of the vestibular system's interaction with cognitive functioning, and the degree of influence it has on the control of eye movements and posture.

1.2 Postural Sway in Humans

In humans, the postural control of a segment or the whole body about a reference position is achieved by passive and active restoring forces applied to the system under control. Under this rationale, the control of whole body posture during upright standing has been modeled as an inverted pendulum oscillating about a fixed position. This simple representation has been very useful for understanding many aspects of human postural control. However, some behaviors observed during upright standing are not well captured by this representation. For example, we conducted a series of studies on natural (unconstrained) prolonged (several minutes) upright standing and showed that individuals tend to oscillate about a moving reference position.(Duarte M, 1999)

In fact, there are no mechanical or neural constraints requiring that humans regulate their upright posture around a reference position somewhat aligned with the vertical axis. An alternative idea is that humans simply adopt a strategy to maximize the safety margin for falling. For example, Slobounov and colleagues (1997) have proposed that we regulate our upright posture by maximizing the time the body center of pressure (COP) would take to contact the stability boundaries at any instant, given the instantaneous position, velocity, and acceleration of the COP at that instant (this time was termed the virtual time to contact). By maximizing the virtual time to contact, a standing person would avoid a fall. Although it remains to be shown to what extent virtual time to contact is incompatible with posture control around a reference point, this theory has not been disproved.

The concept of COP is very useful for understanding the regulation of postural control. The COP expresses the position of the resultant vertical component of the ground reaction force applied to the body at the ground surface. The COP is a two-dimensional position dependent on the acceleration of the body and its segments (because it is related to the forces applied to the body). A related concept to COP is the center of gravity (COG). The COG is defined as a specific point in a system of particles (or segments for the human body) that behaves as if the weight of all particles were concentrated at that point. From simple mechanics, if the vertical projection of the COG (COGv) steps out of the base of support (the area at the floor that circumscribes the region of contact of our body with the ground, e.g., for the bipedal posture, it is the area circumscribing the feet), the body will be unable to apply restoring forces to maintain the upright posture.

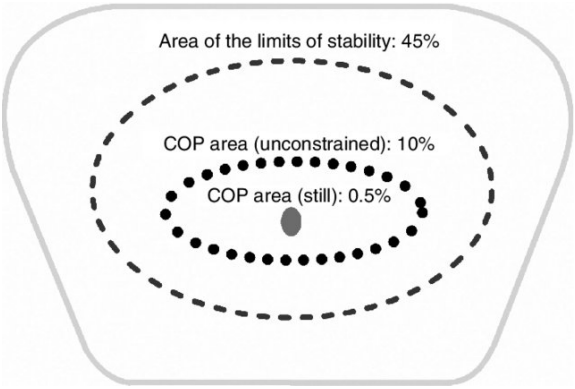


Figure 1.1: Mean values for the limits of the base of support (solid line), limits of stability individuals can voluntarily reach during standing (dashed line), center of pressure (COP) area of sway during prolonged unconstrained standing for 30 min (dotted line), and COP area of sway during standing still for 40 s (filled area in the center).

In this sense, the contour of the base of support can be viewed as the stability boundaries for controlling posture.

In fact, the limits of stability standing humans are able to use are smaller than the physical limits given by the contours of the feet. Figure 1.1 presents mean values for the limits of the base of support, limits of stability the adults can voluntarily reach during standing, COP area of sway during prolonged natural (unconstrained) standing for 30 minutes, and COP area of sway during standing still for 40 seconds. Figure 1.1 shows that, while standing still, humans occupy a very small area of the base of support and that during unconstrained standing this area is much larger. With regards to the amount of sway produced during standing, in general, it is assumed that more sway means more instability and is an indication of a deteriorated posture control system. This rationale is based on many experiments on aging and pathological conditions that showed increased sway in those conditions (see for example, the reviews of Horak et al. 1989 ; Bonnet et al. 2009). However, this is not always the case. Patients with Parkinson disease

in some cases demonstrate reduced postural sway compared to elderly adults, despite the fact that patients with Parkinson disease do present severe problems of postural control (Romero and Stelmach 2003). Another proposition for postural control is that at least part of the sway during upright posture is, in fact, an intentional sway (Riccio et al. 1992 ; Riley et al. 1997 ; Riley and Turvey 2002 ; Stoffregen et al. 2005 ; Bonnet et al. 2009). In this view, more sway does not necessarily imply more instability; neither it is an indication of a deteriorated posture control system. In addition, although elderly

persons, when asked to stand as still as possible for a short period of time, commonly show increased postural sway during standing compared to younger persons, elderly persons show the opposite behavior during prolonged unconstrained standing (Freitas et al. 2005b).

In this chapter, we briefly review the control of equilibrium in humans during quiet standing and findings about prolonged unconstrained standing, and we discuss the implications of these findings for understanding the control of equilibrium in humans. But first we describe how postural sway can be evaluated.

Throughout this chapter, we employ biomechanical principles to understand the control of equilibrium. The use of biomechanics to understand the control of locomotion and the connection between motor control and biomechanics in general are addressed in the other two chapters of this section.

1.2.1 Quantification of Postural Sway during standing

Before 1950, postural sway was studied mainly by recording head oscillation, a method that is called ataxiography . Since then, ataxiography was almost completely forgotten and the recording of ground reaction forces and COP displacement became the prevailing approach. The quantitative evaluation of body sway via force recordings is called posturography , which has been divided into static posturography, when the postural control of a person is evaluated by asking that person to stand as still as possible, and dynamic posturography, when the postural responses to a perturbation applied to the person are evaluated. The most frequent measurement used in posturography is COP displacement, which can be easily measured using a force plate. The most common

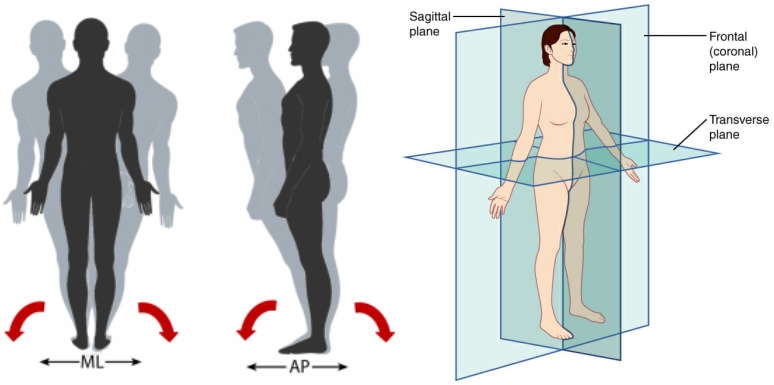


Figure 1.2: ML : Mediolateral - Frontal Plane , AP : Anterioposterior - Sagittal Plane

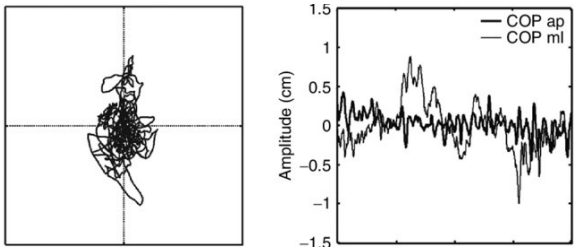


Figure 1.3: Examples of statokinesigram (A) and stabilogram (B) of center of pressure (COP) displacement during standing as still as possible on a force plate.

task used for the evaluation of postural control is the quiet standing task, in which the person is asked to stand "as still as possible," commonly while looking at a fixed target. The COP displacement is then measured and analyzed to quantify the postural sway.

Although the most utilized instrument to evaluate postural control is the force plate and the most commonly measured variable is COP displacement, there is no agreement about which variables derived from the COP signal should be used to evaluate postural sway (see, for example, (Kapetyn, 1983)). Typically, COP displacement during a standing task can be visualized in two ways: in statokinesigram and stabilogram plots (Figure 1.2). The statokinesigram is the map of the COP displacement in the sagittal plane (anteroposterior direction, COP ap) versus the COP displacement in the frontal plane (mediolateral direction, COP ml); whereas the stabilogram is the time series of the COP displacement in each direction. Customarily, posturographic analysis has been divided into global and structural analyses. Global analysis is related to the quantification of the total amount of body sway, whereas structural analysis quantifies particular events or components of body sway.

A large number of measures have been used to describe the amount of postural sway (Winter, 1990)(Duarte M, 1999). Among them, the most common measures are COP spatial displacement (usually standard deviation in each direction or total area), mean speed or velocity, and frequency variables (usually mean or median frequency). Some of the most common variables used in the quantification of body sway in the time and frequency domains are presented in Table 10.1 , and an example of the power spectral density estimation and its outcome variables for COP displacement during quiet standing is presented in Figure 1.3.

In humans, the postural control of a segment or the whole body about a reference position is achieved by passive and active restoring forces applied to the system under

Variable	Matlab code
Standard deviation	std(COP)
RMS (Root Mean Square)	sqrt(sum COP$^2/length(COP)$)
Range of COP displacement	max(COP) - min(COP)
Sway path	sum(abs(COP))
Resultant sway path	sum(sqrt(COPap$^2 + COPml^2$))
Area (95 percent of the COP data inside)	[vec,val] = eig(cov(COPap,COPml));
	Area = pi prod(2.4478 sqrt(svd(val)))
Power spectral density	nfft = round(length(COP)/2);
Peak (Fpeak)	[p,f] = psd(detrend(COP),nfft,
Mean (Fmean)	frequency,nfft,round(nfft/2));
Median (F50) frequency and the	[m,peak] = max(p);
frequency band that contains	area = cumtrapz(f,p);
up to 80 percent of the spectrum (F80)	F50 = find(area > = .50 âĹů area(end));
	F80 = find(area > = .80 âĹů area(end));
	Fmean = trapz(f,f. âĹů p)/trapz(f,p)
	Fpeak = f(peak)
	F50 = f(F50(1))
	F80 = f(F80(1))

Table 1.1: Usual variables used in the global analysis of COP displacement and examples on how to compute these variables by using Matlab software

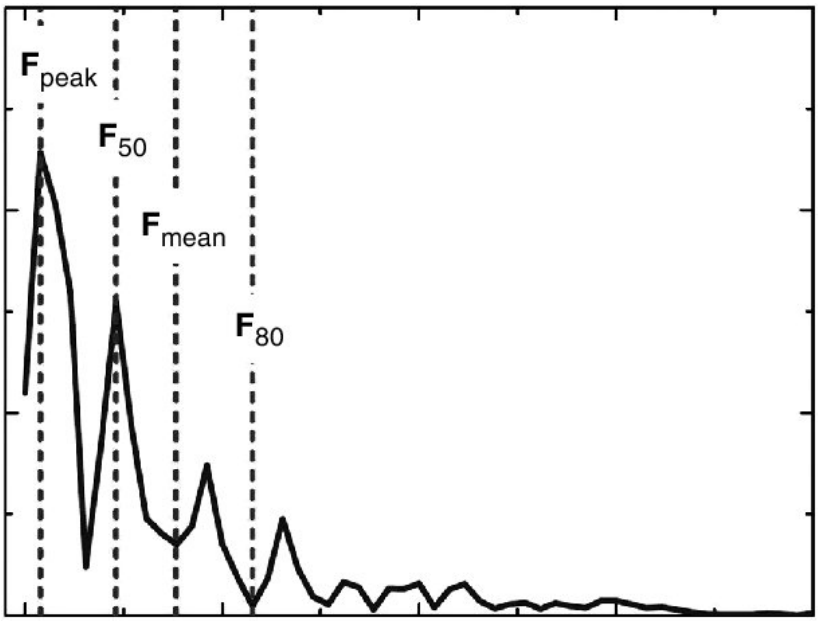


Figure 1.4: Example of the power spectral density estimation of center of pressure (COP) displacement during quiet standing. The peak (Fpeak), mean (Fmean), and median (F50) frequencies and the frequency band that contains up to 80

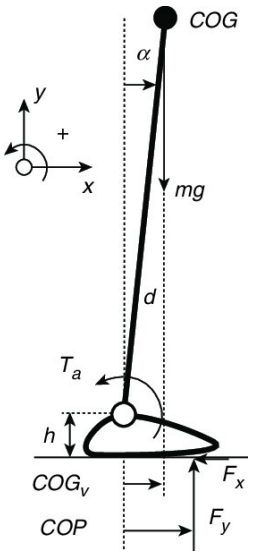


Figure 1.5: Single inverted pendulum model for the representation of a human standing. COG , center of gravity; COG v , COG vertical projection in relation to the ankle joint; COP , center of pressure in relation to the ankle joint; m, g , body mass, acceleration of gravity; F x , F y , horizontal and vertical components of the resultant ground reaction force; T a , torque at the ankle joint; d , distance between the COG and ankle joint; h , height of the ankle joint to the ground; Îś , angle of the body.

control. Under this rationale, the control of whole body posture during upright standing has been modeled as an inverted pendulum oscillating about a fixed position. This simple representation has been very useful for understanding many aspects of human postural control.

And to model the human postural sway and control in we have used something called “Reinforcement Learning”. The next chapter discuss in length what is Reinforcement Learning, it’s basics, advanced alogrithms used, etc.

CHAPTER 2

Background

2.1 Motivation

Reinforcement learning (Richard Sutton, 1998) is a branch of artificial intelligence that deals with learning agent's behavior solely from interaction with an environment. We use a formulation of the reinforcement learning problem where agent interacts with the environment sequentially in discrete time steps. In each step, the agent observes current state of the environment and decides to take an action according to its policy. The environment then transitions to next state and provides feedback in a form of reward signal. Agent 's objective is to maximize cumulative reward received during the entire interaction. The goal of reinforcement learning is finding an action-selection policy that achieves this objective.

Figure 2.1: Mario as a RL example

Reinforcement learning has been combined with deep neural networks into "deep reinforcement learning"(Volodymyr Mnih, 2013). Using advances in deep learning (Ian Goodfellow, 2016), these methods can learn policies directly from raw input without feature engineering. Recently, this approach has been successfully applied to many challenging domains, including playing Atari 2600 games at human level (Mnih, 2015) and defeating human champions in the game Go (Silver, 2016). These games are typically played with discrete actions from a finite domain. On the other hand, many robot

control tasks require continuous actions, e.g. for modulating precise amount of torque applied to joint actuators. In this more general setting, classical optimization techniques are intractable due to the infinite numberof actions. Commonly, these tasks are solved using actor-critic methods, which learn a continuous policy (actor) using feedback from the critic. Deep Deterministic Policy Gradient (DDPG)(Timothy P. Lillicrap, 2015) is an actor-critic method based on the policy gradient which was successfully applied to our inverted pendulum control tasks.

2.2 Setting

We consider a standard reinforcement learning (RL) setting (Richard Sutton, 1998) where agent interacts with the environment sequentiall with discrete time steps t = 0, 1, 2, . . .We model the reinforcement learning problem as an extension of a deterministic Markov Decision Process with a continuous state space $S \subseteq \mathbb{R}^N$ and an action space $A \subseteq \mathbb{R}^M$. An arbitrary state $s \in S$ and action $a \in A$ are real-valued vectors of the dimensions N and M respectively.

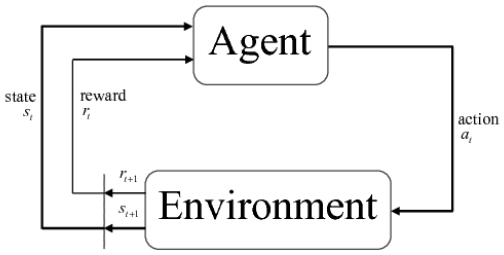


Figure 2.2: Interaction of Agent and Enviroment

Environment In general, environments may be stochastic, but we limit ourselves to the deterministic case which is more suitable for our purposes.We assume that the environment satisfies the Markov property and is fully observable (Richard Sutton, 1998).Therefore, its one-step dynamics can be described by a state transition function $T : S \times A$, a reward function $r : S \times A \to \mathbb{R}$, and a terminal condition $done : S \to \{0, 1\}$ An initial state s_0 of the environment is selected randomly according to a probability distribution $p(s_0)$. In time step t, the agent observes current state s_t of the environment and selects action a_t to perform. The environment and selects action a_t to perform. The environment then transitions to next state $s_{t+1} = T(s_t, a_t)$ and awards the agent with

reward $r_{t+1} = r(s_t, a_t)$. The interaction between the agent and environment ends after reaching a terminal state s where the terminal condition $done(s) = 1$ is satisfied.

Agent The goal of an agent is to select actions so as to maximize the cumulative reward received during the entire interaction with an environment(Richard Sutton, 1998). Agent's behavior is prescribed by a *policy* which selects an action from the action space based on the current state. In general, the policy may be stochastic. Stochastic policy $\pi : S \to P(A)$ maps agent's state space to a probability distribution over actions. In this thesis, we are mainly concerned with learning deterministic policies, a special case of stochastic policies. Deterministic policy $\mu : S \to A$ is a function that maps states directly to actions. In this case, the agent selects action $a = \mu(s)$ in an arbitrary state s. Since many control tasks don't require stochastic policies to be solved, deterministic policies are particularly useful due to their desirable properties in policy gradient estimation an arbitrary state.

2.3 Episodic and Continuing Tasks

In addition to the current state and reward, environments also provide a binary signal $done_t = done(s_t)$ informing the agent whether time step t is final(Richard Sutton, 1998). Upon reaching a terminal state s_t in the final step t, the interaction between the agent and environment is concluded. We call such interaction an $episode$ with duration t and the task $episodic$. On the other hand, tasks that have no terminal states and continue forever are called $continuing$ tasks.

In order to unify notation between both types of tasks, we transform episodic tasks into continuing (Richard Sutton, 1998). We add a special $absorbing\ state$ into the state space. After reaching a terminal state, all further transitions lead to the absorbing state no matter what action is taken. The agent receives zero reward for these transitions. Since the sum of all rewards in the original and the modified task is the same, they can be treated as equivalent for the purposes of reinforcement learning.

2.4 Expected Discounted Return

Reinforcement learning is the process of finding a policy that maximizes some notion of a cumulative reward. Commonly, the concept of discounted return is introduced(Richard Sutton, 1998). Discount factor $\gamma \in \langle 0, 1)$ is used to to scale down the importance of future rewards. Typical values of γ are approaching 1. We define discounted return R_t as the discounted sum of all rewards from time step t till infinity:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ... = \sum_{i=t}^{\infty} \gamma^{i-t} r_{i+1} \tag{2.1}$$

We assume that the rewards are bounded and discount factor $\gamma < 1$, therefore R_t exists and is finite(Richard Sutton, 1998). Notably, the discounted return depends on the environment and the policy used to select actions. In general, they may be stochastic and for this reason, RL typically searches for policies that maximize the expected discounted return $\mathbb{E}_{E,\pi}[R_t]$ dependent on the environment E and the policy π.

2.5 Value Functions

To an agent that maximizes its cumulative reward, some states are more valuable than others.We formulate this value of a state as the expected discounted return that the agent expects to get by following some policy from this state onwards (Richard Sutton, 1998). We define this relationship as a value function $V : S \to \mathbb{R}$. Value function maps each stateValue function maps each state s_t to the expected discounted return R_t, assuming that the agent follows a policy m in an environment E :

$$V^{\mu}(s_t) = \mathbb{E}_{E,\mu}[\, R_t | s_t \,] \tag{2.2}$$

Alternatively, value function can be defined recursively as a sum of the immediate reward r_{t+1} and discounted value of the next state st+1, also known as the Bellman equation

$$V^{\mu}(s_t) = \mathbb{E}_{E,\mu}\left[\, r_{t+1} + \gamma V^{\mu}(s_{t+1}) \,\right] \tag{2.3}$$

Similarly, we can estimate value of an arbitrary state-action pair (s_t, a_t) as the ex-

pected discounted return of taking the action a_t (independent of agent 's policy) in the state s_t and then following the agent's policy μ from the next state s_{t+1} onward. Formally, action-value function $Q : S \times A \rightarrow \mathbb{R}$ is defined as

$$Q\mu(s_t, a_t) = \mathbb{E}_{E,\mu}\left[\, R_t | s_t, a_t \,\right] \tag{2.4}$$

Analogously to the value function, action-value function Q^μ can be defined recursively, too:

$$Q^\mu(s_t, a_t) = \mathbb{E}_{E,\mu}\left[\, r(s_t, a_t) + \gamma Q^\mu(s_{t+1}, \mu(s_{t+1}))\,\right] \tag{2.5}$$

Optimal value function V^* maps states to maximum expected discounted return obtainable by any policy (Richard Sutton, 1998)

$$V^*(s_t) = \max_\mu V^\mu(s_t) \tag{2.6}$$

Optimal action-value function Q^* assigns maximum expected discounted return of any policy to state-action pairs

$$Q^*(s_t, , a_t) = \max_\mu Q^\mu(s_t, a_t) \tag{2.7}$$

$$Q^*(s_t, a_t) = \mathbb{E}_E\left[\, r(s_t, a_t) + \gamma \max_\mu Q^*(s_{t+1}, a_t)\,\right] \tag{2.8}$$

2.6 Temporal Difference

In reinforcement learning, an agent is trying to optimize its policy to maximize the expected discounted return. Typically, the agent also learns a value or an action-value function which are used to improve its policy. Since we are dealing with continuous state and action spaces, we use function approximators to estimate these functions. We parametrize such approximators with parameters Θ which determine the function that the approximator computes. The goal of the learning process is to find parameters of

the approximator that closely estimate the real function (Richard Sutton, 1998).

Considering that we are dealing with continuing tasks, sampling the real value function from an infinite sequence of future rewards is infeasible. Temporal-difference (TD) learning (Sutton, 1998) is a method that solves this problem by bootstrapping a previously learned estimate to learn a new better estimate of a function. TD can be used to incrementally improve the estimate of a value function from past experience of interactions with the environment.

2.6.1 One-Step Temporal Difference

TD learning (Richard Sutton, 1998) requires experience of a policy μ interacting with an environment E. In the simplest one-step version called TD(0), the experiences are one-step transitions between two consecutive steps $t, t+1$. We would like the estimated value $V^{\mu}_{\Theta}(s_t)$ of state s_t to approach the true expected discounted return $\mathbb{E}_{E,\mu}[R_t|s_t]$ from this state. As was already discussed in the previous section, this return is unavailable. Instead, TD(0) approximates it with one-step target return backup R^1_t consisting of the immediate reward r_{t+1} and the bootstrapped value estimate $V^{\mu}_{\Theta}(s_{t+1})$ of the next state 's value.

$$R^1_t = r_{t+1} + \gamma V^{\mu}_{\Theta}(s_{t+1}) \tag{2.9}$$

TD error δ_t is the difference between the one-step return backup and the old value estimate.

$$\delta_t = R^1_t - V^{\mu}_{\Theta}(s_t) \tag{2.10}$$

The same ideas can be applied to estimation of an action-value function Q^{μ} with a parametrized approximator Q^{μ}_{Θ}

$$R^1_t = r_{t+1} + \gamma Q^{\mu}_{\theta}(s_{t+1}, a_{t+1}) \tag{2.11}$$

$$\delta_t = R_t^1 - Q_\Theta^\mu(s_t, a_t) \tag{2.12}$$

Once the TD error is known, we can change the parameters of the approximator to reduce it. We define a loss function L_t for the current transition as a mapping from the approximator 's parameters to squared value of TD error

$$L_t(\Theta) = \frac{1}{2}\delta_t^2 \tag{2.13}$$

Assuming that the approximator is differentiable, we can use gradient descent methods to update its parameters in the direction that minimizes loss $L_t(\Theta)$. We ignore the fact that the target return R_t^1 in TD error also depends on parameters Θ and treat it as a constant.

2.6.2 Multi-Time Temporal-Difference

TD learning is not limited to its one-step variant, which updates the value estimate based on only one transition. Let 's consider an experience of n transitions from steps $t...t+n$. Discounted sequence of received rewards $r_t...r_{t+n}$ together with the bootstrapped value estimate $V_\Theta^\mu(s_{t+1})$ of the last state s_{t+n} gives us a better estimate of the true value function (Richard Sutton, 1998). We define the n-step target return R_t^n as

$$R_t^n = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + ...\gamma^{n-1} rt + n + \gamma^n V_\Theta^\mu(s_{t+1}) \tag{2.14}$$

$$R_t^n = \sum_{i=1}^{n} \gamma^{i-1} r_{t+i} + \gamma^n V_\Theta^\mu(s_{t+1}) \tag{2.15}$$

Analogously to one-step TD error, n-step TD error δ_t^n is defined as the difference between the target return and the old value estimate

$$\delta_t^n = R_t^n - V_\Theta^\mu(s_t) \tag{2.16}$$

In the most general case, a return backup can be an average of many n- step returns

(Richard Sutton, 1998). Algorithm TD (λ) uses λ-return R_t^λ defined as the average of all n-step backups discounted by factor $\lambda \in \langle 0, 1 \rangle$

$$R_t^\lambda = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^n \tag{2.17}$$

2.6.3 On-Policy, Off-Policy Methods

There is an important distinction between how different RL methods learn from experience.We distinguish $on - policy$ and $off - policy$ methods. Let's suppose that we use a policy π to interact with an environment and generate one-step transitions $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$ from two consecutive steps. Off-policy methods are able to use these transitions to learn an approximator of value (or action-value) function of any policy, while on-policy methods are limited to the policy π.

2.7 Actor - Critic Methods

2.7.1 Optimal Policy

In reinforcement learning, we are trying to find a policy that maximizes the expected discounted return. Policy μ^* is optimal (Richard Sutton, 1998) if value function V^μ of following this policy has the same values as optimal value function $V^{\mu*}(s) = V^*(s)$ in every state $s \in S$. Let 's suppose that the optimal action-value Q^* was known, then the optimal policy could be acquired by greedily choosing action that maximizes action-value in current state s_t.

$$\mu^*(s_t) = \arg\max_a Q^*(s_t, a) \tag{2.18}$$

This is a central idea of RL techniques for solving tasks with discrete actions from finite action spaces, such as Q-learning(Watkins, 1992) These techniques typically don't require explicit representation of a policy, since it is implicitly specified by Q function. However, a naive application of Q-learning to continuous action domain by discretization of actions suffers from combinatorial explosion (Timothy P. Lillicrap, 2015)and

doesn't yield good results. To solve this issue, actor-critic methods are typically used

2.7.2 Deterministic Actor-Critic

An actor-critic architecture(Richard Sutton, 1998) consists of two components: an actor that selects actions and a critic which criticizes them. These methods maintain an explicit parametrized representation of a deterministic policy μ_θ, called actor, and an action-value function Q^μ_Θ, called critic.

After the actor selected an action $a_t = \mu_\theta(s_t)$ in a state s_t, we let the critic Q^μ_Θ criticize it by assigning it an action-value $Q^\mu_\Theta(s_t, \mu_\theta(s_t))$. Since finding the optimal action $a^* = \arg\max_a Q^\mu_\Theta(s_t, a)$ w.r.t. current action-value approximator Q^μ_Θ is intractable in a continuous action space, we modify actor's parameters in the direction that increases the action-value. We use deterministic policy gradient theorem(Sutton, 1998) to modify actor 's parameters θ alongside gradient $\bigtriangledown_\theta Q^\mu_\Theta(s, \mu_\theta(s_t))$

$$\theta \leftarrow \theta + \alpha \bigtriangledown_\theta Q_\Theta(s_t, \mu_\theta(s_t)) \tag{2.19}$$

where constant α is a step size (also called learning rate) in the direction of the gradient. By applying chain rule to the above equation, we can split this gradient into the gradient of policy w.r.t. its parameters and the gradient of action-value w.r.t. to the action

$$\theta \leftarrow \theta + \alpha \bigtriangledown_\theta \mu_\theta(s_t) \bigtriangledown_{\mu_\theta} (s_t) Q^\mu_\Theta(s_t, \mu_\theta(s_t) \tag{2.20}$$

After actor's parameters θ have been updated, the action-value function is no longer up-to-date with the latest policy and its parameters Θ need to be updated using TD learning with TD error

$$\delta_t = r_{t+1} + \gamma V^\mu_\Theta(s_{t+1}) - Q^\mu_\Theta(s_t, a_t) \tag{2.21}$$

Usually, the process of policy improvement and learning its value function approximator alternates in a so called policy iteration (Richard Sutton, 1998).

CHAPTER 3

Deep Reinforcement Learning

Deep Reinforcement Learning(Volodymyr Mnih, 2013) is the combination of reinforcement learning and deep learning (Ian Goodfellow, 2016). Deep neural networks are typically used as approximators of a policy and an action-value function. Recently, deep RL has successfully learned to play Atari 2600 games directly from raw input(Mnih, 2015)and beaten human champions in the game Go (Silver, 2016). Deep neural networks are particularly interesting nonlinear function approximators for their universal approximation ability. (Hornik, 1989). However, using deep neural networks brings its own set of challenges (Tsitsiklis, 1997) and requires modifications of the standard RL algorithms.

3.1 Neural Networks

Artificial neural networks (ANN) are a set of machine learning models loosely similar to biological neural networks (Ian Goodfellow, 2016). The basic processing unit of an ANN is a neuron. Neuron processes its input vector x by multiplying each input element x_i by its respective weight w_i and adding bias term b

$$z = \sum_i w_i x_i + b \tag{3.1}$$

A differentiable activation function f is then applied to the weighted sum z to calculate activation y of the neuron

$$y = f(z) \tag{3.2}$$

3.1.1 Multilayer Perceptron

Multilayer perceptron (MLP), sometimes referred to as deep feedforward network or deep neural network, is a type of a feedforward ANN, whose neurons form an acyclic directed graph and are organized into layers (Ian Goodfellow, 2016). MLPs are often used as nonlinear function approximators.

A layer is a set of neurons with the same inputs. MLP consists of an input layer, one or more hidden layers, and an output layer. The input layer is a special kind of layer that has no inputs, but rather stores the network's input in its activations. We say that a layer is fully-connected if the layer, its input, and the connections between them form a complete bipartite graph. Hidden layers and output layer are all fully-connected.

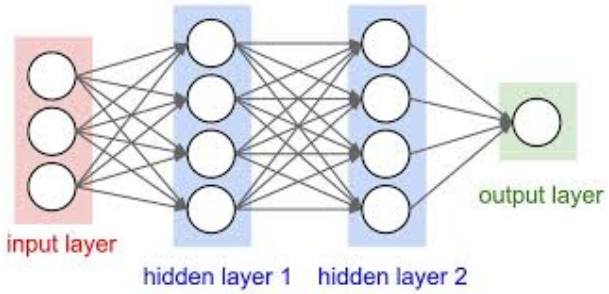


Figure 3.1: MLP consisting of an input layer with three neurons, two hidden layers each with four neurons respectively, and an output layer with one neuron

Forward Propagation Let's us suppose we have an MLP with M hidden layers. Instead of describing behavior of individual neurons, we will use more convenient vector notation to describe entire layer(Ian Goodfellow, 2016). We denote inputs of the network as vector y_0, activations of neurons in ith hidden layers as vector y_i, and output layer activations as y_{M+1}. We also denote bias terms as vector b_i and connection weights as weight matrix W_i, where element in j th row and k th column is a weight of connection from input j to neuron k of the respective layer.

Each non-input layer $i = 1, ..., M + 1$ computes its weighted sum of inputs z_i and activation y_i based on the activation of previous layer y_{i-1} according to

$$z_i = y_{i-1}W_i + b_i \tag{3.3}$$

$$y_i = f(z_i) \tag{3.4}$$

The output of a network is computed in a process called forward propagation. Firstly, activations of the input layer y_0 are set to the input of the network. Then, activations are propagated from the input layer through hidden layers to the output layer in a cascading fashion (figure 3.1) following equations 3.3 and 3.4. Finally, activations of the output layer become the output of the network.

Backpropagation ANNs are usually trained using gradient descent methods (Ian Goodfellow, 2016). These methods are trying to minimize some loss function L of network parameters θ (weights and biases) by modifying parameters in the direction of gradient $\bigtriangledown_\theta L(\theta)$. To compute this gradient for parameters in every layer, we use backpropagation, or backward propagation of errors.

We start the process by computing $\bigtriangledown_{y_{M+1}} L(\theta)$, the gradient of the loss function w.r.t. to the network's output (Ian Goodfellow, 2016). This gradient will depend on the chosen loss function. As an example, the squared error loss $L(\theta) = \frac{1}{2}\|y_{M+1} - target\|^2$ is commonly used in supervised learning where the goal is to bring the network 's output closer to the $target$ vector. By differentiating this loss w.r.t. the network's output, we get $\bigtriangledown_{y_{M+1}} L(\theta) = y_{M+1} - target$.

Each layer then computes its error vector $\delta_i = \bigtriangledown_{zi} L(\theta)$, which is the gradient of the loss function w.r.t. the layer's weighted sum of inputs.With the gradient $\bigtriangledown_{y_{M+1}} L(\theta)$ that we already calculated, we can directly compute the error vector of the output layer δ_{M+1} as

$$\delta_i = \bigtriangledown_{y_{M+1}} L(\theta) \odot f*'(z_i) \tag{3.5}$$

where $\odot$ is element-wise multiplication operator and $f_0$is derivative of the activation function. We require activation function to be differentiable for this reason. For every hidden layer $i = M...1$, we use the chain rule to propagate the errors backward according to equation

$$\delta_i = (W_{i+1}^T \delta_{i+1}) \odot f'(z_i) \tag{3.6}$$

where W_{i+1}^T is transposed weight matrix of the next layer. Similar to forward propagation, backpropagation propagates the error in a cascading fashion. However, propagation happens in the opposite direction from the output layer to the first hidden layer. After that, we compute the desired gradient of the loss w.r.t. the parameters $\bigtriangledown_\theta L(\theta)$ Since parameters of layer i consist of both weight matrix W_i and bias b_i, we compute their gradients as follows

$$\bigtriangledown_{W_i} L(\theta) = y_{i-1}^T \delta_i \tag{3.7}$$

$$\delta_{b_i} L(\theta) = \delta_i \tag{3.8}$$

Finally, parameters θ are updated alongside their respective gradients depending on our gradient descent algorithm of choice. In case of the standard stochastic gradient descent (SGD) (Ian Goodfellow, 2016) with learning rate parameter α, the update looks as follows

$$\theta_i \leftarrow \theta_i + \alpha \bigtriangledown_\theta L(\theta) \tag{3.9}$$

One may also choose a more advanced technique such as momentum [21], root mean square propagation (RMSProp) [22], or Adaptive Moment Estimation (Adam) [23]. They typically use information about gradients from past updates to heuristically improve training speed. The same technique described here can be used for the gradient ascent, instead of descent, by simply flipping the sign of the loss.

3.1.2 Regularization

When searching for parameters of a neural network, it is not only important to find ones that minimize the loss on the training data, but ones that also generalize well to new unseen data. One approach to this problem is limiting network 's capacity using parameter norms. L^2 parameter regularization is an approach to regularization, that introduces weight decay (Ian Goodfellow, 2016). We modify the original loss function L of parameters θ to also minimize the magnitude of the weights W (biases are unchanged),

the assumption being that smaller weights generalize better. This gives us a new loss function L_R

$$L_R(\theta) = L(\theta) + \frac{1}{2}\|W\|^2 \tag{3.10}$$

3.1.3 Mini-Batch Methods

In the previous sections, we have described online (or stochastic) gradient descent algorithm that uses single input sample to form an estimate of the gradient. In general, this doesn't have to be the case. We distinguish a spectrum of methods ranging from online methods to batch methods, that use the entire dataset to estimate the gradient. Mini-batch methods are in the middle and estimate gradient from multiple samples, called mini-batch (Ian Goodfellow, 2016). Justification for using mini-batches is achieving more accurate gradient estimate by reducing the variance of individual samples, which may improve training time. Thanks to the parallelism of modern graphics processing units (GPUs), the performance penalty of using mini-batches is sublinear w.r.t. batch size. Mini-batch is an input matrix whose rows are individual input vectors. Equations for forward propagation (section 3.1.1) and backpropagation (section 3.1.1) require no modification thanks to our notation. One change that's typically done is scaling the gradients by factor $\frac{1}{m}$, where m is batch size, in order to average the gradients instead of summing them.

3.1.4 Batch Normalization

Different dimensions of network's input may have different scales or be shifted, which unnecessarily slows down learning. Generally, it is a good practice to normalize each dimension of network's input vector to have zero mean and unit variance across the whole dataset. However, this only helps in training the first hidden layer of an MLP. Input distributions of layers that are deeper in the network may still change uncontrollably as the network is trained. Batch normalization(Ioffe, 2015) is a technique for accelerating training of neural networks. It performs normalization per each mini-batch and can be applied to inputs of all layers. During training, inputs are normalized by mean and variance computed from the mini-batch. Moving averages of mean and variance of

inputs are maintained and used for normalization during testing. Batch normalization is differentiable and backpropagates errors through itself. For exact derivations of how it is trained, we refer the reader to the original paper (Ioffe, 2015).

3.2 Experienced Relay

An experienced relay $x_t = (s_t, a_t, r_{t+1}, s_{t+1})$ An is created by observing interaction of an agent and an environment in two consecutive steps (Mnih, 2015) . Deep neural networks typically require a lot of training data to learn. Training them with online stochastic gradient descent using only the most recent experience doesn 't lead to satisfactory results for two reasons. Firstly, it is wasteful to train on each experience only once. Secondly, consecutive experiences are typically highly correlated. This breaks assumptions of gradient descent methods and causes catastrophic forgetting of old experiences in neural networks. One solution to this problem is experience replay. After every interaction with the environment, we store the current experience xt in a replay memory. Replay buffer is a data structure used as replay memory that can store a limited number of experiences N. When it's full, the oldest sample is removed to make space for a new one. When a neural network is trained with experience replay, it uses mini-batches of experiences sampled uniformly from the replay buffer. This helps to both reuse experiences and to uncorrelate consecutive samples. Experience replay has demonstrated its usefulness in Deep Qlearning (Volodymyr Mnih, 2013), a variant of Q-learning with a neural network approximator of the action-value function, which has successfully learned to play Atari 2600 games.

3.3 Prioritized Experienced Relay

Experience replay samples experiences uniformly from the replay memory, which means they are replayed with the same frequency as they are experienced. However, not all experiences are equally important for learning, especially if rewards are very sparse, e.g. they are non-zero only at the end of an episode.

Prioritized experience replay (Tom Schaul)addresses this issue by assigning a priority p_t to each experience sample x_t. Experience x_t is sampled from a prioritized replay

buffer R stochastically according to their priority $p_t > 0$. Probability of selecting sample x_t is defined as

$$P(t) = \frac{p_t^\alpha}{\sum_{i \in R} p_i^\alpha} \tag{3.11}$$

where exponent α determines the amount of prioritization. In general, priorities may be arbitrary. In RL tasks, it is quite suitable to prioritize samples by magnitude of their TD error, which describes how surprising, or informative, the sample is. For experience x_t we set its priority $p_t = |\delta_t| + \epsilon$, where ϵ is a small positive constant to make sure $p_t > 0$. Priorities of samples may be updated at any time. A convenient time to update them is right after the experiences are replayed, as we can use TD errors that have already been computed.

Prioritized replay changes the distribution of experiences away from the real distribution (Tom Schaul). This negatively affects the learning of expected values depending on this distribution. In order to correct for this bias, importance-sampling weight w_t of experience x_t is used

$$w_t = \left(\frac{1}{|R|} . \frac{1}{P(t)} \right)^\beta \tag{3.12}$$

Exponent $\beta \in \langle 0, 1 \rangle$ is a constant determining amount of compensation for this bias (β = 1 is a full compensation, β = 0 none). Weights are normalized by $1/max_i w_i$ for stability reasons. TD error δ_t is scaled using normalized weight to $w_t \delta_t$ and then used for training.

3.4 Deep Deterministic Policy Gradient

Deep Deterministic Policy Gradient (DDPG) (Timothy P. Lillicrap, 2015) is a model-free offpolicy actor-critic RL algorithm applicable to problems with continuous action spaces. It is based on the deterministic policy gradient method and uses neural networks as function approximator of a deterministic policy μ_θ with parameters θ and an action-value function Q^μ_Θ with parameters Θ.

Similar to Deep Q-learning (Volodymyr Mnih, 2013; Mnih, 2015), DDPG also uses

experience replay to train its neural networks by replaying random mini-batches of uncorrelated transitions.

3.4.1 Target Networks

Straightforward application of neural networks to deterministic policy gradient (section 2.6.2) doesn 't lead to stable training, as action-value approximator is trained recursively using bootstrapped action-value estimated by the same approximator. Nonlinear function approximators such as neural networks are known to diverge under such conditions in TD learning (Tsitsiklis, 1997).

DDPG solves this problem by using another set of networks for creating bootstrapped backups (Timothy P. Lillicrap, 2015). Target policy network μ_θ, is initialized with parameters θ', a copy of original policy parameters θ. Likewise, target action-value parameters Θ. Target networks then slowly track original parameters. Every time the original parameters are updated, target parameters move closer to them

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \tag{3.13}$$

$$\Theta \leftarrow \tau\Theta + (1-\tau)\Theta' \tag{3.14}$$

according to parameters $\tau\langle 0, 1\rangle$. Smaller τ leads to slower tracking of original parameters, but more stable learning.

Action-value parameters Θ are updated according to one-step TD error δ_t with bootstrapped return backup estimated by target networks

$$\delta_t = r_t + \gamma Q^\mu_\Theta(s_{t+1}, \mu_\theta(s_t)) - Q^\mu_\Theta(s_t, a_{t+1}) \tag{3.15}$$

3.4.2 Deterministic Policy Gradient

Policy parameters θ are updated in line with the deterministic policy gradient

$$\bigtriangledown_\theta Q^\mu_\Theta(s, \mu_\theta(s_t)) \tag{3.16}$$

Although, the use of neural networks trained by backpropagation offers a more intuitive view of the policy gradient computation. Let us consider an actor-critic architecture as shown in figure 3.2 on the next page. Policy and action-value networks are interlinked (Matthew J. Hausknecht, 2015) so as to compute action-value $Q^\mu_\Theta(s_t, \mu_\theta(s_t))$, where s_t is an arbitrary state. The goal of the training process is to increase the action-value $Q^\mu_\Theta(s_t, \mu_\theta(s_t))$ by modifying policy parameters θ. This leads to a straight-forward definition of a loss function

$$L_t(\theta) = Q^\mu_\Theta(s_t, \mu_\theta(s_t)) \tag{3.17}$$

Since the approximators Q^μ_Θ and μ_θ are differentiable, so is the loss function. We use gradient ascent to modify policy parameters in the direction of gradient $\bigtriangledown_\theta L_t(\theta)$. Application of chain rule to this gradient and computation of parameter updates is taken care of by the backpropagation algorithm. In this process, parameters of action-value approximator remain constant and only the policy parameters are updated.

CHAPTER 4

Methods

4.1 RL model

The following block diagram of the RL model can be seen in the figure below.

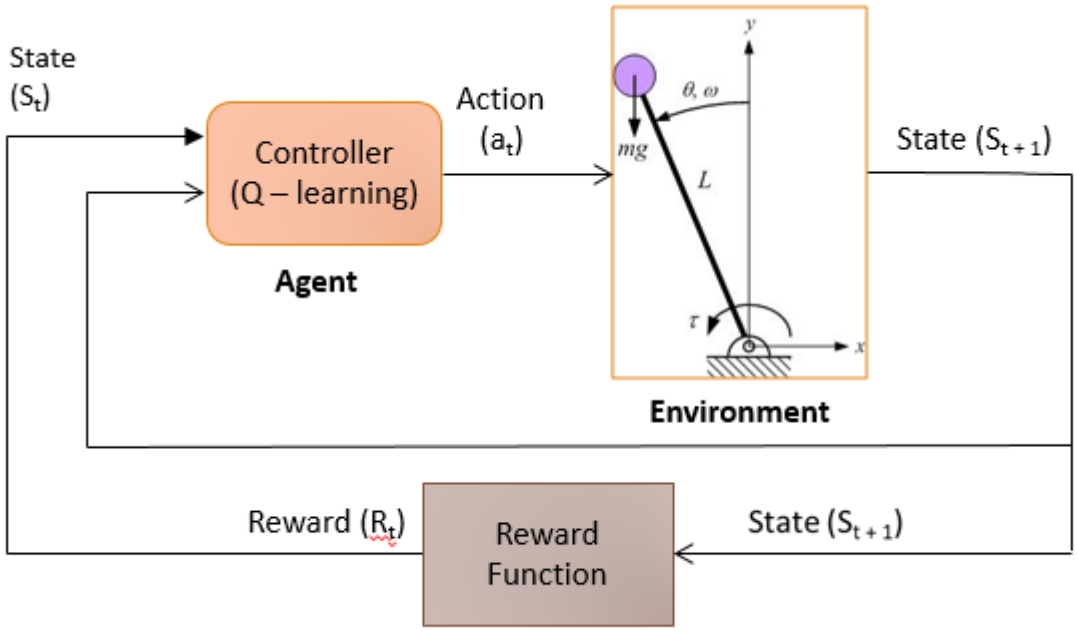


Figure 4.1: Block of the RL model

In our Reinforcement Learning setting, the Angular Displacement (θ) and Angular Velocity ($\dot{\theta}$) are the states of the agent, which is defined by the Inverted Pendulum and Neuronal Controller combined and the Torque applied by the neuronal controller is the action that drives the agent towards the next state. Environment is defined as the space around the inverted pendulum which constitutes the action of forces such as gravity, damping, stiffness, etc.

4.1.1 Environment

Dynamical Equation :

$$I\frac{d^2\theta}{dt^2} = \tau(t) + mglsin(\theta) - b\dot{\theta} - k\dot{\theta} \tag{4.1}$$

where,

- I - Moment of Inertia = ml^2
- l - Length of Center of Mass (i.e. Position of 2nd Lumbar Vertebrae) = 110 cm
- m - mass of healthy adult human = 65 kgs
- k & b - pivot joint properties of the inverted pendulum characterizing ankle joint, k - stiffness constant = 8 kg/s and b - damping constant = 0.8 kg/s

Reward :

$$reward = -(\theta^2 + 0.1\dot{\theta}^2) \tag{4.2}$$

4.1.2 Agent

$$\tau = \tau_{nc} + c\mathcal{N}(0,1) \tag{4.3}$$

$$\tau(t) = \lambda\tau(t) + (1-\lambda)\tau(t-1) \tag{4.4}$$

where,

- τ_{nc} - torque applied by the neuronal controller
- $\mathcal{N}(0,1)$ - white gaussian noise of mean 0 and variance 1
- c - noise amplitude
- λ - filtering factor

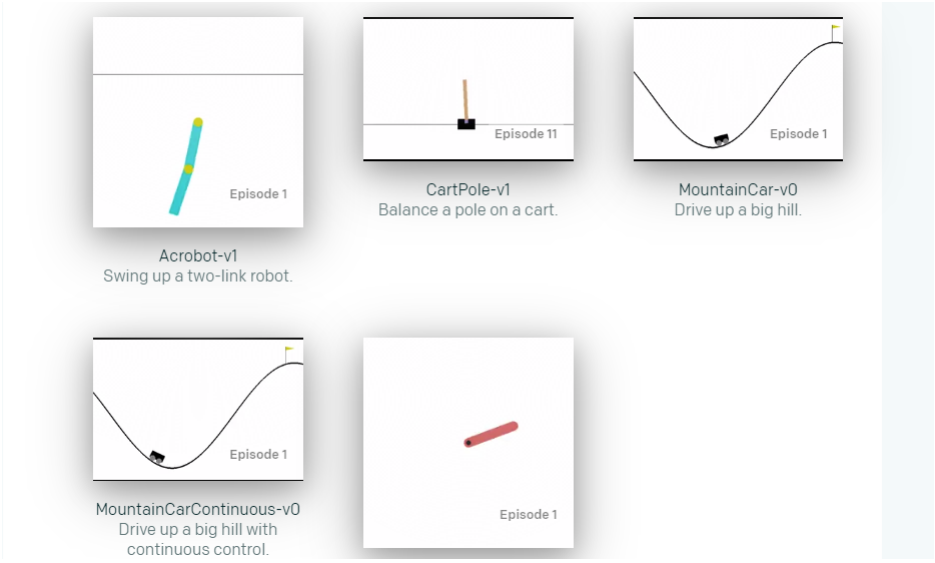
The parameters of the above equation are subject to change for the fine tuning of the model including the reward function

4.2 Implementation

As a part of this thesis, we provide an efficient implementation of the tested deep RL methods (DDPG) written in Python using TensorFlow library, Stable Baselines and Keras RL. The agents are tested in environments from OpenAI Gym, a collection of RL tasks. Github was heavily used during the during the course of the project. Over 50 models were trained by tuning parameters such as filtering factor, noise amplitude, learning rate, training time etc. to find optimal set of parameters for the model. Each model took over 150 minutes and was then tested over the environment.

4.2.1 OpenAI Gym

OpenAI Gym (Greg Brockman, 2016) is a reinforcement learning toolkit. It includes a collection of various environments for comparing performance of different RL algorithms. Each environment has the same interface for interacting with the agent. We can customize our own environment with the existing environments as a reference and then implement our own agent to achieve goals in the complex uncertain environment.



The customized environment developed for the problem statement is registered as 'Inverted_Pendulum-v0' and figure below shows how it is rendered in a video.

The following are values of parameters taken to derive results from the model.

For implementation of DDPG algorithm, the state and action were continuous hence Box function was used from gym.spaces to initialize the state and action space.

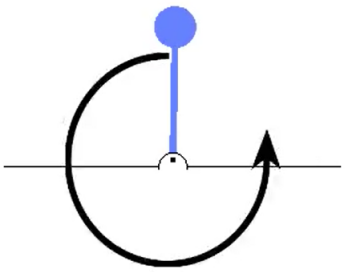


Figure 4.2: 'Inverted_Pendulum-v0': Direction of arrow shows the direction of torque applied

The action was 1dimensional. The bounds of states and action were defined in the __init__ function. Maximum torque was taken to be 300 N-m. The maximum allowed θ and $\dot{\theta}$ are 22.5 radians and 0.5 sec^{-1}. The timestep dt was taken to be 0.01. The values of the dynamical equation were as follows :

g = 9.8 acceleration due to gravity, m = 65 Mass, l = 1.1 length, a = 0.83 Filtering factor, b = 0.8 damping constant, k = 8 stiffness constant, c = np.sqrt(81) noise amplitude

Terminal condition was true if the pendulum crosses the maximum θ i.e 22.5 radians.

This Project can be found on GitHub here : `https://github.com/Shritej24c/Q_Pendulum`

4.2.2 Stable Baselines

Stable Baselines is a set of improved implementations of Reinforcement Learning (RL) algorithms based on OpenAI Baselines.

Github repository:`https://github.com/hill-a/stable-baselines`

4.3 Experimental Data

The data was collected by us from TCS.

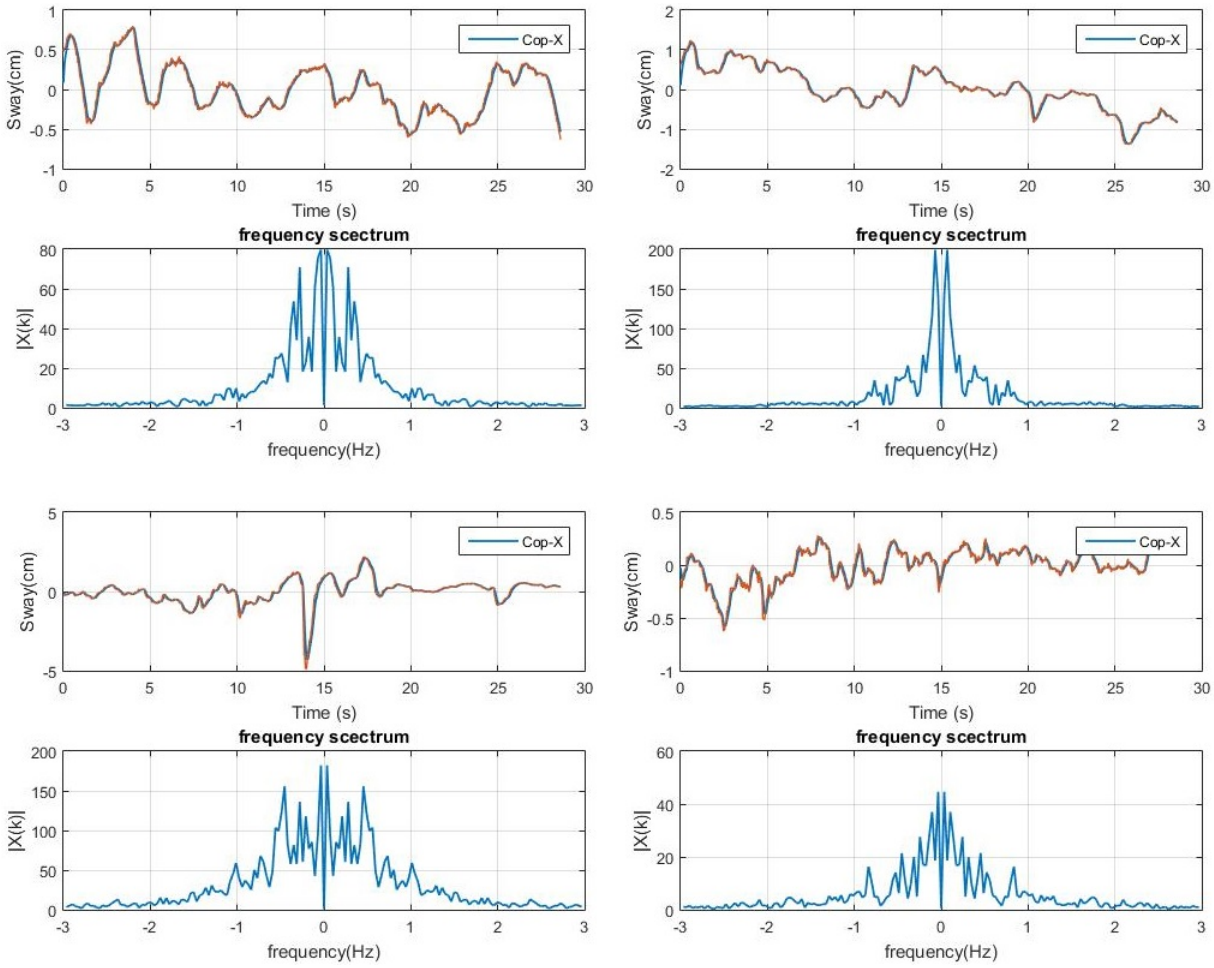


Figure 4.3: Sway of 4 Healthy Adults and their respective Frequency Spectrum

CHAPTER 5

Results

In the plot of sway versus time the negative values show the sway towards the left side.

5.1 Model 1

In this model the actor learning rate is changed from 0.0001 to 0.001 and everything is kept the same as discussed in section The graphs are as follows :

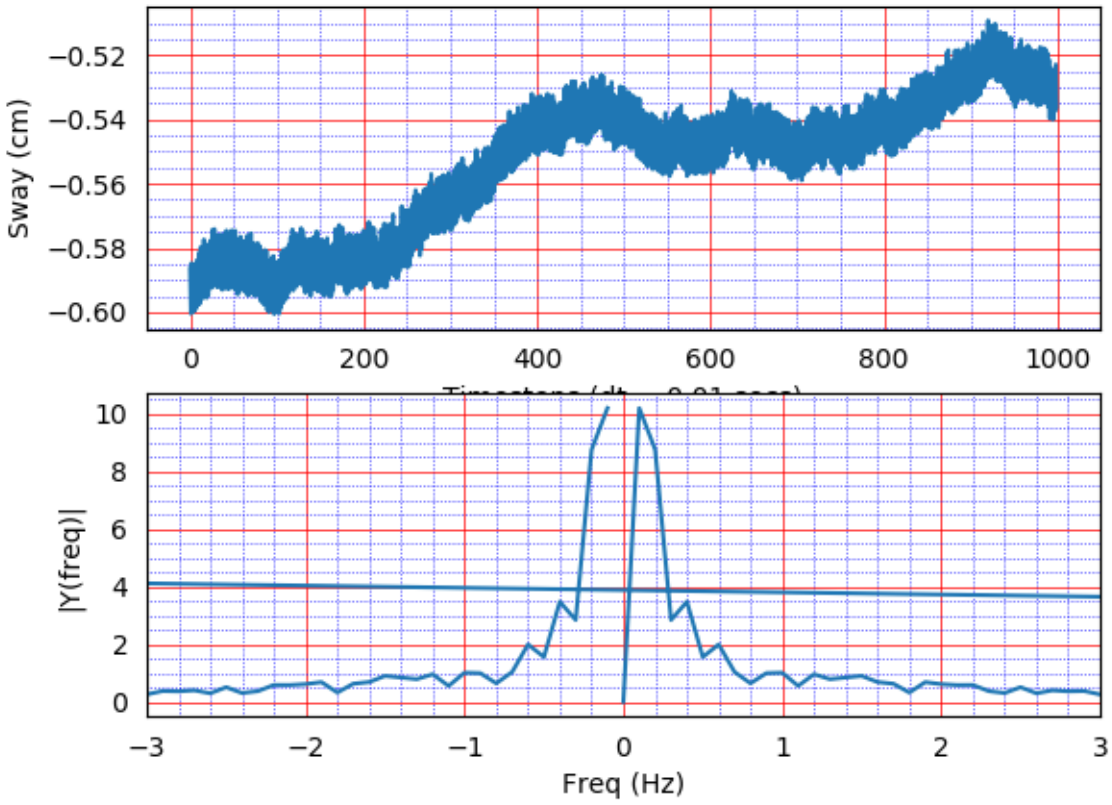


Figure 5.1: Sway Characteristics and Power Spectral Density

As discussed in the 1.4 the most area under PSD lies between frequencies -1 to 1 and above the plot satisfies this characteristic.

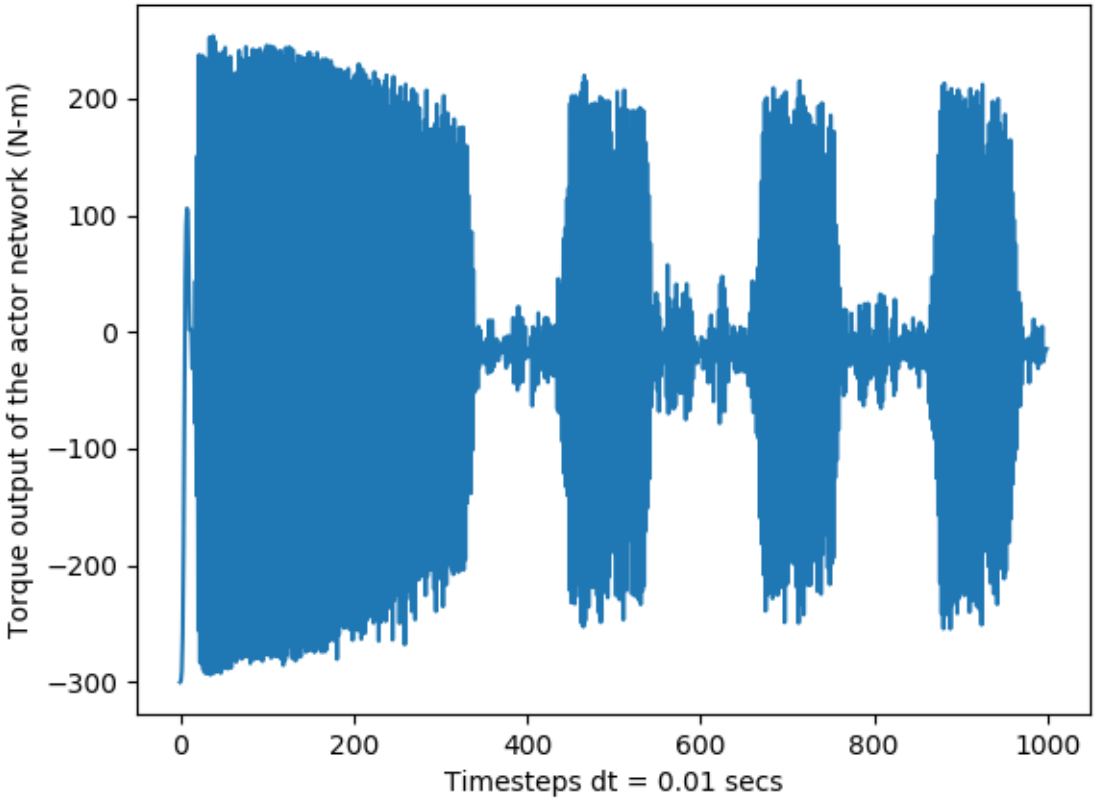


Figure 5.2: Action vs Time

5.2 Model 2

In this model we have kept the filtering factor = 0.35

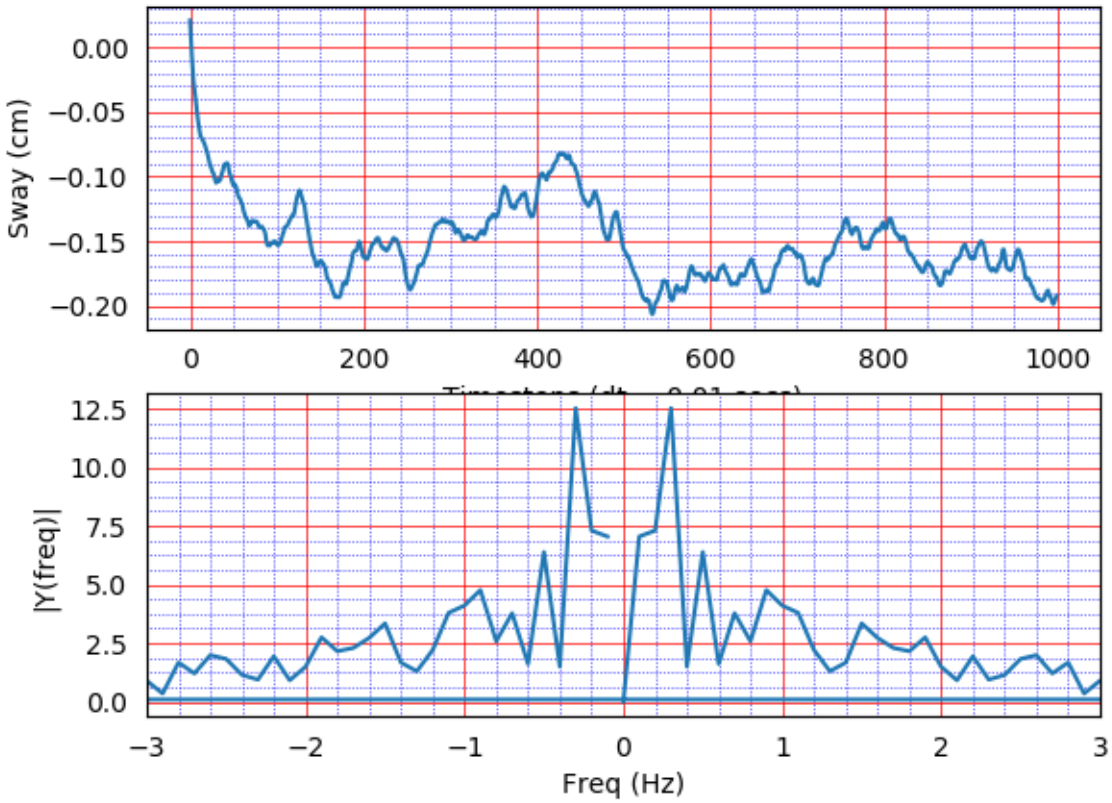


Figure 5.3: Sway Characteristics and Power Spectral Density

The pendulum as seen from the above graph is falling towards the negative x-direction but frequency spectrum specifies some significant characteristics.

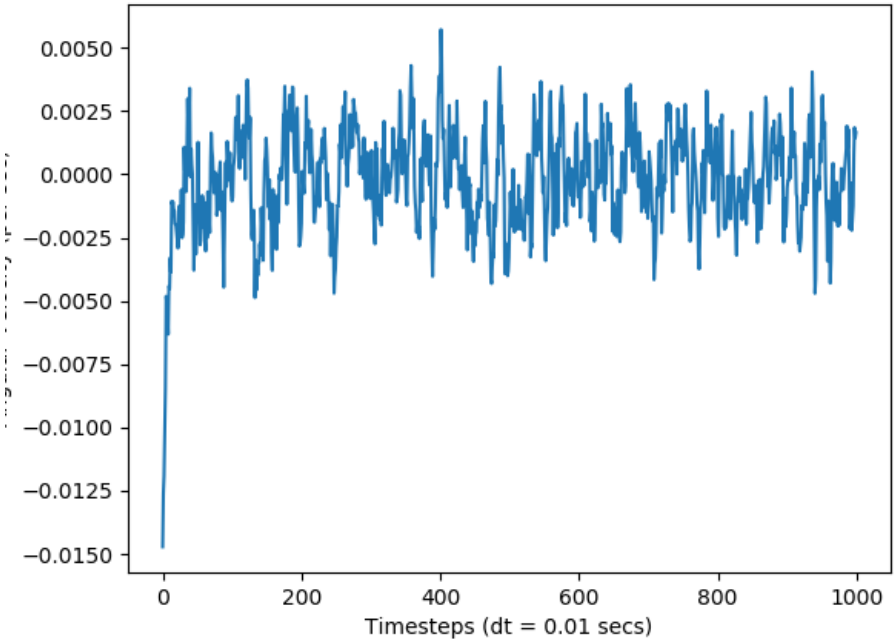


Figure 5.4: Angular Velocity versus Time

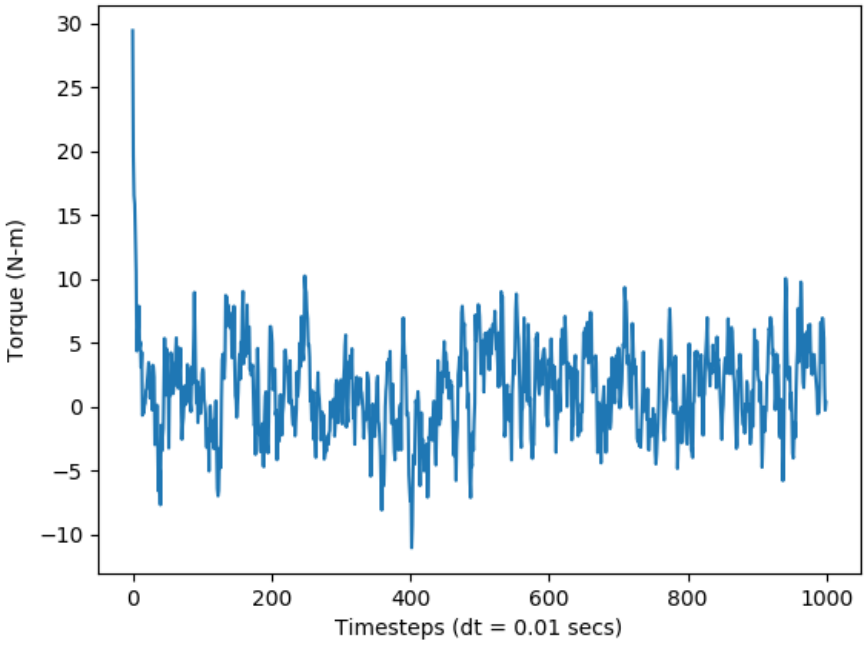


Figure 5.5: Action vs Time

CHAPTER 6

Future Work and Limitations

- Model can be developed to fit the experimental data more accurately by proper tuning of parameters
- Also introduction of scaling factor or the use of Genetic Algorithms to reduce the search space for the actor network could be tried
- Model can be further fitted for pathological conditions, specifically Parkinson's Disease
- Model can be extended to 2-dimension
- Visual feedback could be incorporated

REFERENCES

1. **Duarte M, V. M. Z.** (1999). Patterns of center of pressure migration during prolonged unconstrained standing. *Motor Control*, **3**, 12–27.

2. **Greg Brockman, L. P. J. S. J. S. J. T., Vicki Cheung** (2016). Openai gym. URL `arXiv:1606.01540.`

3. **Grillner, K., Wallen** (2008). Neural bases of goal direction and locomotion vertebrates. *Brain*, 2–12.

4. **Hornik, K.** (1989). Multilayer feedforward networks are universal approximators. *Neural Netw*, **2**(5), 359–366.

5. **Ian Goodfellow, Y. B.**, *Deep Learning*. MIT Press, 2016. URL `http://www.deeplearningbook.org`.

6. **Ioffe, S.** (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Corr*, **1502**. URL `http://arxiv.org/abs/1502.03167`.

7. **Kapetyn, B.** (1983). Standardization in platform stabilometry being a part of posturography. *Agressologie*, **24**, 321 –26.

8. **Matthew J. Hausknecht, P. S.** (2015). Deep reinforcement learning in parameterized action space. *CoRR*, **1511**. URL `http://arxiv.org/abs/1511.04143`.

9. **Mnih, V.** (2015). Human-level control through deep reinforcement learning. *Nature*, (7540), 529–533.

10. **Nielsen** (2003). How we walk: central control of muscle activity in human walking. *Neuroscientist*, **9**, 195–204.

11. **Richard Sutton, A. G. B.**, *Introduction to Reinforcement Learning*. MIT Press, Cambridge, Massachusetts, London, England, 1998.

12. **Silver, D.** (2016). Mastering the game of go with deep neural networks and tree search. *Nature*, **529**(7587), 484–489.

13. **Sten Grillner, T. D., G.N. Orlovsky**, *Neuronal Control of Locomotion: From Mollusc to Man*. Oxford University Press, 1999.

14. **Sutton, R. S.** (1998). Learning to predict by the methods of temporal differences. *Machine Learning*, **3**(1), 9–44.

15. **Timothy P. Lillicrap, D. S.** (2015). Continuous control with deep reinforcement learning. *CoRR*, **1509**. URL `http://arxiv.org/abs/1509.02971`.

16. **Tom Schaul, J. Q., David Silver** (). Prioritized experienced relay. *CoRR*, **1511**. URL `http://arxiv.org/abs/1511.05952`.

17. **Tsitsiklis, J.** (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control*, **42**(5), 674–690.

18. **Volodymyr Mnih, D. S.** (2013). Playing atari with deep reinforcement learning. *Cornell University*. URL `https://arxiv.org/pdf/1312.5602.pdf`.

19. **Watkins, C. J.** (1992). Q-learning. *Machine Learning*, **8**(3), 279–292.

20. **Winter, A. P. J. F., D.A.** (1990). Assessment of balance control in humans. *Medical Progress through Technology*, **16**, 31–51.