

**LEAST SQUARE REINFORCEMENT LEARNING FOR
SOLVING CART-POLE BALANCING PROBLEM**

SA-NGAPONG PANYAKAEW

**MASTER OF SCIENCE
IN COMPUTER SCIENCE**

**GRADUATE SCHOOL
CHIANG MAI UNIVERSITY
MAY 2019**

**LEAST SQUARE REINFORCEMENT LEARNING FOR
SOLVING CART-POLE BALANCING PROBLEM**

SA-NGAPONG PANYAKAEW

**A THESIS SUBMITTED TO CHIANG MAI UNIVERSITY IN PARTIAL
FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN COMPUTER SCIENCE**


**GRADUATE SCHOOL
CHIANG MAI UNIVERSITY
MAY 2019**

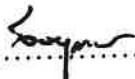
LEAST SQUARE REINFORCEMENT LEARNING FOR SOLVING CART-POLE BALANCING PROBLEM


SA-NGAPONG PANYAKAEW

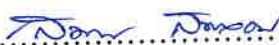
THIS THESIS HAS BEEN APPROVED TO BE A PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN COMPUTER SCIENCE

Examination Committee:


 Chairman
(Dr. Sanparith Marukatat)


 Member
(Assoc. Prof. Dr. Jeerayut Chaijaruwanich)

 Member
(Asst. Prof. Dr. Jakramate Bootkrajang)

 Member
(Asst. Prof. Dr. Samerkae Somhom)

Advisory Committee:

 Advisor
(Assoc. Prof. Dr. Jeerayut Chaijaruwanich)

 Co-advisor
(Asst. Prof. Dr. Jakramate Bootkrajang)

29 May 2019

Copyright © by Chiang Mai University

ACKNOWLEDGMENT

At the moment of time, I used to think that my thesis will not accomplish. I was going to quit several times but, that never happened because there are many people support me. I would like to show my gratitude to my parent Tongwan and Sangla those who support me everything and respect all of my decision. My brother Jarinya and my sister-in-law Thanutchta who took care of me for years of my study.

I extend my respect to most of my teachers who are my role model and conduct me to study computer science. I wish to honor my advisor Jeerayut Chaijaruwanich, for every support, encouragement, merciful and opportunity for all years of my study. I appreciate for trust me all the way through. I wish to thank my co-advisor Jakramate Bootkrajang, for good advice, improve my writing skill and made me fascinated reinforcement learning since I was an undergrad student. I would extend my respect to my early school teachers especially, Mullika Tawonatiwas who made me interested in science and sowed curiosity to the kid that day. I would like to mention Yossawin Fagpan who introduce programming basic and inspired me to interest in programming stuff.

Special thanks to Kitimapond Rattanadoung, Papangkorn Inkeaw and my friends for friendship and goodwill.

Sa-ngapong Panyakaew

Thesis Title	Least Square Reinforcement Learning for Solving Cart-Pole Balancing Problem	
Author	Mr. Sa-ngapong Panyakaew	
Degree	Master of Science (Computer Science)	
Advisory Committee	Assoc.Prof. Dr. Jeerayut Chaijaruwanich	Advisor
	Asst. Prof. Dr. Jakramate Bootkrajang	Co-advisor

ABSTRACT

Cart-pole balancing is a classic control problem that can be solved by reinforcement learning approach. Most of previous work consider the problem in discrete state space which rather unnatural. In this work, we consider the problem in continuous state space with conditions that track length is limited and period of time considering the task success is extended. Least square policy iteration algorithm is adopted for learning process. A new reward function is proposed. Moreover, various factor that influent the success of the learning are studied. The empirical result validate the effectiveness of our method.

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright© by Chiang Mai University
All rights reserved

หัวข้อวิทยานิพนธ์	การเรียนรู้เสริมกำลังแบบกำลังสองน้อยที่สุดเพื่อแก้ปัญหาคู่สมการคาร์ท-โพล	
ผู้เขียน	นายสง่าพงศ์ ปัญญาแก้ว	
ปริญญา	วิทยาศาสตรมหาบัณฑิต (วิทยาการคอมพิวเตอร์)	
คณะกรรมการที่ปรึกษา	รศ.ดร.จิรยุทธ ไชยจรรูณิช	อาจารย์ที่ปรึกษาหลัก
	ผศ.ดร.จักรเมธ บุตรกระจำง	อาจารย์ที่ปรึกษาร่วม

บทคัดย่อ

ปัญหาคู่สมการคาร์ท-โพลจัดเป็นปัญหารูปแบบหนึ่งที่สามารถแก้ไขได้โดยใช้การเรียนรู้แบบเสริมกำลัง จากงานในอดีตที่ผ่านมาปัญหาคู่สมการคาร์ท-โพลถูกพิจารณาแบบเป็นสถานะที่ไม่ต่อเนื่อง ซึ่งไม่มีความเป็นธรรมชาติ ดังนั้นในงานนี้จึงพิจารณาปัญหาคู่สมการคาร์ท-โพลในรูปแบบที่เป็นสถานะที่ต่อเนื่อง และประยุกต์ใช้การเรียนรู้แบบเสริมกำลังในการแก้ปัญห โดยได้มีการเพิ่มข้อกำหนดให้รางวัลของรถเลื่อนให้มีระยะที่จำกัด และเพิ่มระยะเวลาเงื่อนไข เพื่อให้สามารถทำการกิจได้ยาวนานขึ้น รวมทั้งงานนี้ประยุกต์ใช้อัลกอริทึมการวนซ้ำแผนงานแบบกำลังสองที่น้อยที่สุดสำหรับกระบวนการเรียนรู้ นอกจากนี้ยังศึกษาปัจจัยที่ส่งผลต่อการทำการกิจให้สำเร็จ พร้อมทั้งนำเสนอผลลัพธ์เชิงประจักษ์เพื่อตรวจสอบประสิทธิภาพของกระบวนการ

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright © by Chiang Mai University
All rights reserved

CONTENTS

	Page
ACKNOWLEDGEMENT	c
ABSTRACT IN ENGLISH	d
ABSTRACT IN THAI	e
CONTENTS	f
LIST OF FIGURES	h
LIST OF ABBREVIATIONS	i
LIST OF SYMBOLS	j
CHAPTER 1 Introduction	Error! Bookmark not defined.
1.1 Motivation	Error! Bookmark not defined.
1.2 Research objectives	Error! Bookmark not defined.
1.3 The usefulness of the research	Error! Bookmark not defined.
1.4 Research scope	Error! Bookmark not defined.
1.5 Research methodology	Error! Bookmark not defined.
CHAPTER 2 Background	Error! Bookmark not defined.
2.1 Decision	Error! Bookmark not defined.
2.2 Agent environment interaction	Error! Bookmark not defined.
2.3 Markov decision process	Error! Bookmark not defined.
2.4 Value function	Error! Bookmark not defined.
2.5 Policy	Error! Bookmark not defined.
2.6 Action selection method	Error! Bookmark not defined.
2.7 Policy iteration	Error! Bookmark not defined.
2.8 Reinforcement learning	Error! Bookmark not defined.
2.9 State approximation	Error! Bookmark not defined.

CHAPTER 3 Reinforcement Learning for Solving Cart-Pole Balancing Problem **Error!**

Bookmark not defined.

3.1 Cart-Pole Balancing Problem Simulation **Error! Bookmark not defined.**

3.2 State and action definition **Error! Bookmark not defined.**

3.3 Reward function **Error! Bookmark not defined.**

3.4 Experiment **Error! Bookmark not defined.**

CHAPTER 4 Experimental Result **Error! Bookmark not defined.**

4.1 Comparison between fixing and updating sample collection method **Error!**

Bookmark not defined.

4.2 Effect of Initial Sample on Performance of the Agent **Error! Bookmark not defined.**

4.3 Effectiveness of proposed reward function **Error! Bookmark not defined.**

4.4 Effect of λ on the Performance of the Policy **Error! Bookmark not defined.**

CHAPTER 5 Conclusion and Discussion **Error! Bookmark not defined.**

5.1 Discussion **Error! Bookmark not defined.**

5.2 Conclusion **Error! Bookmark not defined.**

REFERENCES **Error! Bookmark not defined.**

APPENDICES

APPENDIX A Additional reward function **Error! Bookmark not defined.**

APPENDIX B Stability and reproducibility of reinforcement learning **Error! Bookmark not defined.**

CURRICULUM VITAE **Error! Bookmark not defined.**

LIST OF FIGURES

Page

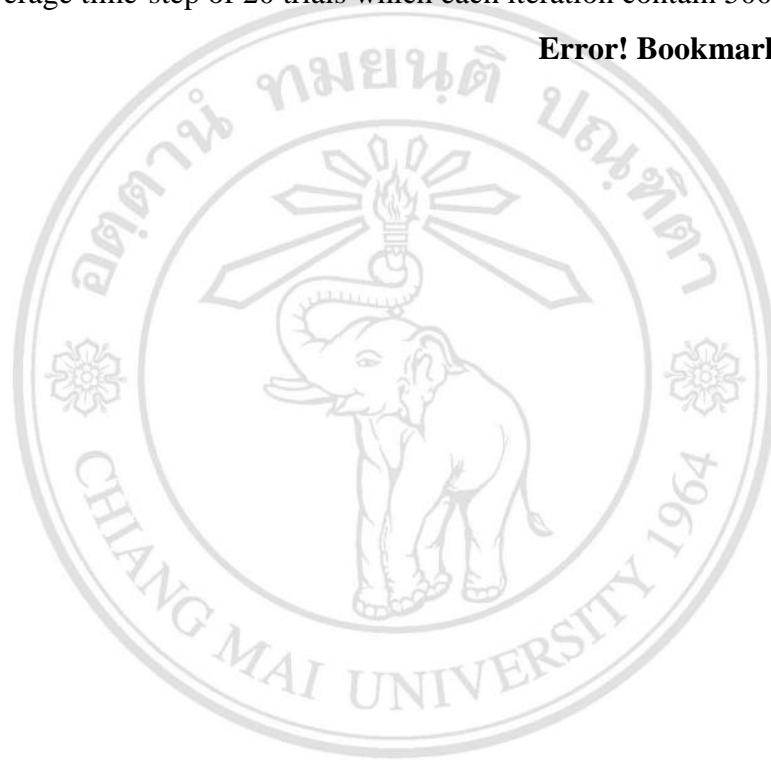
Figure 2.1 Policy iteration algorithm	Error! Bookmark not defined.
Figure 2.2 TD(0) algorithm	Error! Bookmark not defined.
Figure 2.3 TD(λ) algorithm	Error! Bookmark not defined.
Figure 2.4 LSPI algorithm	Error! Bookmark not defined.
Figure 2.5 LSTDQ algorithm	Error! Bookmark not defined.
Figure 3.1 A flowchart illustrating how the sample gets updated.	Error! Bookmark not defined.
Figure 4.1 Average time-step of 100 episodes in each iteration for two agents learning with fixed sample and continuously updated sample	Error! Bookmark not defined.
Figure 4.2 Reward per decision in each iteration for two agents learning with fixed sample and continuously updated sample	Error! Bookmark not defined.
Figure 4.3 Average time-step of 100 episodes in each iteration for initial samples.	Error! Bookmark not defined.
Figure 4.4 Reward per decision in each iteration for three initial samples.	Error! Bookmark not defined.
Figure 4.5 Average time-step of 100 episodes in each iteration for reward functions	Error! Bookmark not defined.
Figure 4.6 Reward per decision in each iteration for reward functions	Error! Bookmark not defined.
Figure 4.7 Average time-step of 100 episodes in each iteration for lambda values	Error! Bookmark not defined.
Figure A.1 Average time-step of 100 episodes in each iteration for R_6	Error! Bookmark not defined.
Figure A.2 Reward per decision in each iteration for R_6	Error! Bookmark not defined.

Figure A.3 percent of episode ending in each iteration for R_6 **Error! Bookmark not defined.**

Figure B.1 Average time-step of 100 episodes in each iteration for 20 trials **Error! Bookmark not defined.**

Figure B.2 Average time-step of 20 trials which each iteration contain 100 episodes **Error! Bookmark not defined.**

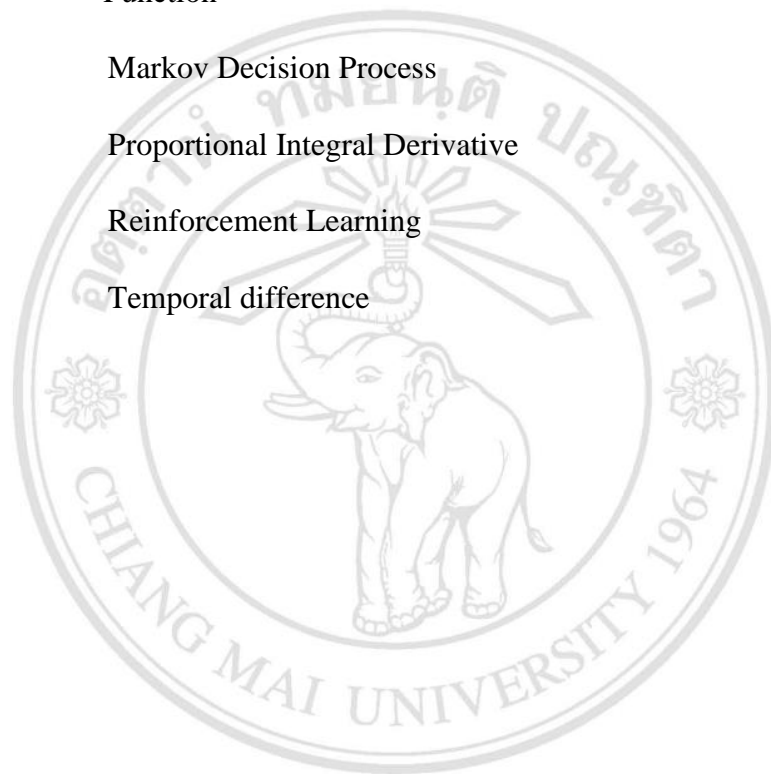
Figure B.3 Average time-step of 20 trials which each iteration contain 500 episodes **Error! Bookmark not defined.**



ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright© by Chiang Mai University
All rights reserved

LIST OF ABBREVIATIONS

LSPI	Least Square Policy Iteration
LSTDQ	Least Square Temporal Difference for State-Action Value Function
MDP	Markov Decision Process
PID	Proportional Integral Derivative
RL	Reinforcement Learning
TD	Temporal difference



ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright© by Chiang Mai University
All rights reserved

LIST OF SYMBOLS

π	Pi, Policy
ϕ	Phi, Basis function
Q	Action value function, State-action Value function
V	State value function, vale function
a	Action
r	Reward
s	State
w	Policy weight vector
\mathbb{E}	Expected value
Σ	Sigma, Summation
\forall	For all
\in	Set membership
\wedge	And
\vee	Or

ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright © by Chiang Mai University
All rights reserved

CHAPTER 1

Introduction

1.1 Motivation

Inverted pendulum is a classic problem in control theory. It is an unstable condition that pivot placed under the center of mass which could be found in daily life, such as two parallel wheeled vehicles (dicycle). The task is to maintain a condition that mass remains above pivot as long as possible. For research, inverted pendulum is usually simplified to cart-pole balancing problem which the cart is placed on that track that limited movement in one dimension, forward and backward and the pole is mounted on the cart with one degree joint. The objective for the cart-pole balancing problem is to prevent the pole from falling down as long as possible by moving the cart back and forth. In practical, a control strategy is considered successful if the pole stays upright longer than some predefined threshold. Some might add the condition that the cart must not move too far from where it started.

Cart-pole balancing problem is a popular benchmark task which has been solved by various algorithms such as PID [1], fuzzy control [2]. However, these algorithms require prior knowledge for processing. For example, PID, stand for proportional-integral-derivative, is a control loop feedback mechanism which calculates the output from a linear combination of proportional, integral and derivative term of error. Three coefficients needs to be tuned manually while the output is not optimal for some application and do not react to changing behavior of process.

Another method to solve the cart-pole problem is Reinforcement learning (RL) [3]. RL is a kind of machine learning that able to learn to perform some tasks without knowledge of the task and exact supervision. It can learn by itself by trial-error at first then exploit its own knowledge later. RL is promised to be adopted for most of the tasks, especially optimal control, it then competent for learning a controller. For certain state, the RL-based controller or agent interacts with the environment by choosing an action

and receives feedback as a reward or penalty (negative reward). The reward is the guidance of RL. It influent agent to learn mapping states-actions that leads to maximum long-term reward, the ultimate goal.

RL was able to solve the cart-pole problem with discrete state space [3], [4]. However, treating the problem in discrete domain is quite restrictive. The work in [5] and [6] then reconsider the problem in a more relaxed continuous state space setting. Still, most previous work only studied the learning in the situation where the length of the track is unlimited which is rather unrealistic.

In this work, using reinforcement learning to learn the cart-pole controller will be studied, with the condition that track length is limited. The problem will be considered in continuous state space using a least square reinforcement learning approach. A new reward function which is competent for influencing the agent towards optimal control will be proposed.

1.2 Research objectives

To propose a new reward function for cart-pole balancing problem in which track length is limited and state space is continuous.

To empirically investigate effect of various factors, namely, updating sample method, quality of initial sample, lambda value and reward function on agent's performance.

1.3 The usefulness of the research

Able to apply reinforcement learning for solving cart-pole balancing problem and to understand its limitation in practice which crucial for implementation of other tasks that state space is continuous.

1.4 Research scope

In this study, we will solve cart-pole balancing problem using LSPI algorithm. The condition for algorithm are continuous state and discrete action space. Experiments are conducted via simulation provided by OpenAI [7] in which the length of cart track is limited and the amount of time-step for considering the task success is set to 6000. The performance is measured by amount of time-step that agent survived.

1.5 Research methodology

1.5.1 Study theory of reinforcement learning and cart-pole balancing problem

1.5.2 Define major component of reinforcement algorithm such as state, basis function, action and reward

1.5.3 Implement LSPI algorithm using python programming language

1.5.4 Conduct experiment comparing various factors, namely, updating sample method, quality of initial sample, lambda value and reward function on agent's performance.

1.5.5 Documentation and Publication

This thesis is organized as the follows, the basic concept of reinforcement learning in chapter 2. The proposed methodology is described in chapter 3, Experimental results are in chapter 4 and chapter 5 concludes the study.



ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright© by Chiang Mai University
All rights reserved

CHAPTER 2

Background

This chapter will explain the basic concept of reinforcement learning which relates to our work. Most of the content refers to Sutton's book [8].

2.1 Decision

Decision making is selecting an action, based on situation, belief or preference of decision maker, that lead to alternate consequences. Some problems required multiple decisions to complete the objective, for examples chess, inverted pendulum control.

Chess is one of the most well-known board game, play by two players, each player have 16 pieces which consist of 6 types, each type able to move differently. The objective is to checkmate the opponent's king. During that player have to move and capture the opponent's pieces for a better position. For every turn, the player selects a move as well action based on board position that considered as situation or state and player's insight. For a certain move, there are various choices for the opponent to respond, as a result, the next position (that player expect) occurs by chance.

Inverted pendulum is another problem that involves decision making. Considerate the problem as simplified as cart-pole which the cart, placed on the track, limited movement only back and forth, vertically mounted by a pole with one degree of freedom joint. Movement of the cart considered as action while state is a combination of the pole's angle and pole's angular velocity. Some also limited distance of the track, so the cart's position is needed to be included to state. For every moment of state, the controller must decide the action to maintain the pole to upright position and to maintain the cart to still in the track, in case, the track length is limited.

All mentioned problems require multiple decisions to complete the task. Chess is needed to play until win or lost. Cart-pole is needed to be controlled as long as possible. For every state, the best action is needed to be decided based on information of that moment, then the new state occurs. The process is repeated until the task completed, that would be called sequential decision making.

2.2 Agent environment interaction

Regularly, the task is an interaction between decision maker, called agent, and the environment. Agent can be player (chess player), controller or software (cart controller), while environment is the world that facing agent (opponent of chess player, physical cart pole). The agent sensed situation as state that represents environment status, and decide to choose action then, the state change and feedback is sent to agent. The feedback that reflects consequence of action is considered a reward. The scenario that agent on certain state then select an action and transit to the new state is called time-step.

For the task such as chess, there are states that terminate decision, for example, checkmate, these state is called terminal state. The transitions from initial state to terminal state is called episode.

2.3 Markov decision process

Sequential decision-making problem could be modeled as Markov Decision Process (MDP) which defined as a tuple (S, A, T, R) where

S is a finite set of state

A is a finite set of action

$T: S \times A \rightarrow S$ is a transition function that returns next state s' for given current state $s \in S$ and action $a \in A$. The next state s' might be occurred by probability.

$R: S \times A \rightarrow R$ is reward function for given current state s and action a

The objective of MDP is to find the optimal policy π^* that maximize long term/future reward or return.

2.4 Value function

When agent performs, there are transitions from state to state. The essence of agent is to decide the best action for current state while the action should transfers agent to a

better state. How good of being on the certain state is measured by potential (expected) of return which called value function (state value function). For given policy π , value function underlying the policy V^π for certain state s is defined as

$$V^\pi(s) = \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s \right] \quad (1)$$

Where γ is discount factor that $0 < \gamma < 1$, t is time-step that $t \in \{0,1,2,3, \dots\}$, r_t is reward on time-step t . When agent perform, there are transitions from state to state. It is obvious that transitions returning high reward are preferred. The reward from a transition of state is called immediate reward. However, one immediate reward might not lead agent to ultimate goal. For example, chess game, when agent capture opponent's queen which is the most important one, it gain high reward (in case reward function is returned follow on rank of opponent piece), but it might not win the game, due to capturing the queen cause too much causality. Return thus relate both immediate and future reward. Since current state s is given and an immediate reward r is known. Now value function is defined as

$$V^\pi(s) = r + \gamma \mathbb{E} \left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} \mid s_0 = s \right]$$

$$V^\pi(s) = r + \gamma \mathbb{E}[V^\pi(s_1) | s_0 = s] \quad (2)$$

where s_1 is next state.

There is another value function that shows potential of action which is state-action value function or action value function as called as Q-function. For given policy π , Q-function underlying policy π is defined as

$$Q^\pi(s, a) = r + \gamma \mathbb{E}[Q^\pi(s_1, \pi(s_1)) | s_0 = s, a_0 = a] \quad (3)$$

2.5 Policy

The agent act to environment by discrete time-step. At each time-step, agent chooses the best possible action, based on its knowledge, to maximize reward in long run. Agent's knowledge is summarized as policy $\pi: S \rightarrow A$ that mapping state to action. The good policy is very important for agent to archive the goal.

Since value function is reflected potential of reward so, the policy should return the action that maximizes value function. Suppose V^* is the optimal state value function, the optimal policy π^* is defined as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} V^*(s) \quad (4)$$

However, to evaluate the best action from a state value function, it requires expected value function of next state that also require state transition probability. As a result, it is uncomfortable if agent does not have perfect knowledge of environment that will be discussed in a later section. In the same way, a policy can also be defined from an action value function. Suppose Q^* is the optimal action-value function, the optimal policy π^* is defined as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}} Q^*(s, a) \quad (5)$$

With action value function, policy for given state s is simultaneously derived by the action that maximizes value of every state-action for the given state.

2.6 Action selection method

ε -greedy is the action selection method employed in this work. It makes the agent explore the environment at first and then exploit its own knowledge later. The method is selecting the best action most of the time but randomly select the action with a small probability (ε). It is defined as follows

$$\pi(s) = \begin{cases} \underset{a}{\operatorname{argmax}} V(s) \\ \text{random } a \in A_s \end{cases} \quad (6)$$

where π is policy function and A_s is set of available action for state s . To implement ε -greedy, one can set ε as high as 1 on start and reduce its value over time.

2.7 Policy iteration

If we know the optimal value function then the optimal policy is also known. Policy iteration is one of procedure to find the optimal policy. Starting from arbitrary policy, the procedure evaluates the policy and obtain value function then improve policy and repeat process until the new policy unchanged. The final policy had been proved that optimal [9].

$$\pi_0 \rightarrow V^{\pi_0} \rightarrow \pi_1 \rightarrow V^{\pi_1} \rightarrow \pi_2 \rightarrow V^{\pi_2} \rightarrow \pi_3 \rightarrow V^{\pi_3} \rightarrow \dots \rightarrow \pi_*$$

2.7.1 Policy evaluation

Policy evaluation is a method for calculating value function from a policy. In this work, we will stick with iterative approach. For given arbitrary policy π , the value function is updated as follows equation.

$$V'(s) = r + \gamma \sum_a P^\pi(a|s) \sum_{s_{t+1}} T(s_{t+1}|s, a) V(s_{t+1}) \quad (7)$$

Policy Iteration

Policy evaluation

Input : current value function V , current policy π

Output : derived value function underlying policy V^π

- 1 **Repeat**
- 2 **For each** $s \in S$
- 3 $V^\pi(s) \leftarrow r + \gamma \sum_a P^\pi(a|s) \sum_{s_{t+1}} T(s_{t+1}|s, a) V(s_{t+1})$
- 4 **Until** $\max |V^\pi(s) - V(s)| < \text{threshold}$ **for all** $s \in S$
- 5 **Return** V^π

Policy improvement

Input : current policy π , current value function V

Output : improved policy π'

- 1 $\text{policy stable} \leftarrow \text{true}$
- 2 **For each** $s \in S$
- 3 $\pi'(s) \leftarrow \operatorname{argmax}_a V(s)$
- 4 **If** $\pi(s) \neq \pi'(s)$ **then** $\text{policy stable} \leftarrow \text{false}$
- 5 **If** policy stable **then**
- 6 **Return** π'
- 7 **else**
- 8 **goto** *Policy evaluation*

Figure 2.1 Policy iteration algorithm

where $P^\pi(a|s)$ is selection probability of action a for given state s underlying the policy π . $T(s'|s, a)$ is occurrence probability of next state s_{t+1} for given state s and action a . This method is iterative approach, a new value function V' is estimated based current value function underlying the policy V , so there are approximation of value function $V_0, V_1, V_2, \dots, V_k$ that $V_k \approx V^\pi$ when $k \rightarrow \infty$. Practically, the algorithm continues until value function converges or slightly change. The procedure is depicted in figure 2.1

2.7.2 Policy improvement

To improve policy, it needs to be sure that value function of the new policy better than the old one for all state as follows inequation

$$V^{\pi'}(s) \geq V^\pi(s), \forall s \in S \quad (8)$$

where V^π is value function underlying current policy π and $V^{\pi'}$ is value function underlying new policy π' . The new policy can be obtained greedily as equation

$$\pi'(s) = \operatorname{argmax}_a V^\pi(s) \quad (9)$$

However, obtaining a policy from state-value function is quite challenging due to V -function do not describe utility of any action so, it is more comfortable to obtaining policy from action-value function instead.

$$\pi'(s) = \operatorname{argmax}_a Q^\pi(s, a) \quad (10)$$

2.8 Reinforcement learning

In MDP context, agent completely knows mechanism of the environment. For example, state transition probability is known. While practically, agent just experiences environment without any knowledge and improve decision from existing data, as called as, sample which is sequence of state-action and reward. Learning to solve MDP from sample is reinforcement learning.

The main subject is still finding the optimal value function to calculating the optimal policy. Policy iteration is the procedure to achieve the optimal policy, but without knowledge of environment, policy evaluation method which using only sample is required.

2.8.1 Temporal difference learning

Temporal difference (TD) is a set of policy evaluation algorithms based on sample (collected by certain policy π). It estimates value function underlying policy π iteratively, as a result $V \approx V^\pi$. The value function is updated incrementally iteratively as a summation of current estimated value function and step-sized error as follows equations.

$$\begin{aligned}\delta &= \text{Target Value} - V_t \\ V_{t+1}(s) &= V_t + \alpha_t \delta\end{aligned}\quad (11)$$

where α_t is step-size parameter or learning rate which satisfies the follows conditions.

$$\sum_{t=0}^{\infty} \alpha_t = \infty, \quad \sum_{t=0}^{\infty} \alpha_t^2 < \infty$$

The ideally *Target Value* is actual $V^\pi(s)$. However, it is implicit due to limitation of environment knowledge so that lead to various version of TD.

1) TD(0)

The *Target Value* suppose to be $V^\pi(s)$ which is expected return so, one can assume that $V^\pi(s)$ equal to $r + \gamma V_t(s')$. The value function is updated by follows equation

$$V'(s) = V(s) + \alpha_t [r_t + \gamma V(s_{t+1}) - V(s_t)] \quad (12)$$

According to the equation, the estimation base on only value function of the successor state, so it can be executed right after a transition of state. The incremental updating rule depict in figure 2.2

TD(0)

Input : policy to be evaluated π
Output : derived value function V^π

- 1 initialize V arbitrarily (e. g. , $V(s) \leftarrow 0$ for all $s \in S$)
- 2 initialize s_0
- 3 **Repeat**
- 4 action $\leftarrow \pi(s_t)$
- 5 take action get *reward* r and next state s_{t+1}
- 6 $V^\pi(s) \leftarrow V(s) + \alpha_t [r_t + \gamma V(s_{t+1}) - V(s_t)]$
- 7 **Until** s_{t+1} is absorbing state
- 8 **Return** V^π

Figure 2.2 TD(0) algorithm

2) Monte-Carlo

Another way to estimate value function is considerate *Target Value* as actual return denoted as G .

$$V_{t+1}(s) = V_t(s) + \alpha_t[G - V_t(s)] \quad (13)$$

Collecting data entire episode is required to finding return. Assuming the sample consists an episode of data length k , the return for each state is calculated backward from terminal to start state as follows equation.

$$G = \sum_{i=k-1}^0 \gamma^{i-(k-1)} r_i \quad (14)$$

Monte-Carlo requires data entire episode so, the value function is strict with batch update.

3) TD(λ)

TD(0) is versatile for any application but sometimes updating value function based on one time-step of data is not sensible while Monte-Carlo requires data entire episode which is actual value but not good for long episode task. To trade off these, the algorithm that requires k -time-steps of data is presented. TD(λ) is an algorithm which generalizes both TD(0) and Monte-Carlo. Considerate TD(0)'s updating rule, which requires a time-step of data, as an estimator \mathbb{E}^1 . We know that

$$V(s_{t+1}) = r_{t+1} + \gamma V(s_{t+2}) \quad (15)$$

substitute $V(s_{t+1})$ to equation 11

$$V'(s) = V(s) + \alpha_t[r_t + \gamma r_{t+1} + \gamma^2 V(s_{t+2}) - V(s_t)] \quad (16)$$

The above equation requires r_t and r_{t+1} , which are 2 time-steps of data, for updating value function so, it is considered as an estimator \mathbb{E}^2 . In the same way, estimator \mathbb{E}^k is defined as

$$V'(s) = V(s) + \alpha_t \left[\sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}) - V(s_t) \right] \quad (17)$$

where $k \in \mathbb{Z}^+$. The *Target Value* of TD(λ) is a mixture of every estimator (\mathbb{E}^k for all $k \in \mathbb{Z}^+$) by weight each term of them by $\lambda^{k-1}(1 - \lambda)$ where $0 \leq \lambda \leq 1$

***TD*(λ)**

Input : policy to be evaluated π , *lambda value* λ where $0 \leq \lambda \leq 1$

Output : derived value function $V^{\pi'}$

- 1 initialize V arbitrarily (e.g., $V(s) \leftarrow 0$ for all $s \in S$)
- 2 initialize s_0
- 3 initialize state trace e as zero ($e(s) \leftarrow 0$ for all $s \in S$)
- 4 **Repeat**
- 5 $e(s) \leftarrow 1$
- 6 action $\leftarrow \pi(s_t)$
- 7 take action get *reward* r and next state s_{t+1}
- 8 **For each** $s \in S$
- 9 $V^{\pi'}(s) \leftarrow V(s) + \alpha_t [r_t + \gamma V(s_{t+1}) - V(s_t)] e(s)$
- 10 $e(s) \leftarrow \lambda \gamma e(s)$
- 11 **Until** s_{t+1} is absorbing state
- 12 **Return** $V^{\pi'}$

Figure 2.3 TD(λ) algorithm

$$\text{Target Value} = \sum_{k=1}^{\infty} \lambda^{k-1} (1 - \lambda) \mathbb{E}^k \quad (18)$$

The incremental updating rule depicts in figure 2.3. In case $\lambda = 0$, the algorithm equal to TD(0) while $\lambda = 1$, the algorithm equal to TD(1).

2.9 State approximation

The conventional reinforcement learning operates in discrete state space. It works well in the task which state is discrete but, for continuous state task, state space is must be discretized. However, it is not easy to split the state space, which is continuous in nature. With too many states, more experience is needed to learn the value function because the agent needs to at least visit each state once. In contrast, with too few states, the controller might not be able to perform correctly.

To adopt continuous state space, the value function can be approximated by weight vector w . The value function, instead of presented as tubular, is now represented by dot product of feature vector ϕ and weight vector w as a follows equation

$$\hat{V}^w(s) = w^T \phi(s) \quad (19)$$

where feature vector $\phi(s)$ is a basis function representing state s . By the way, there are various ways to define feature vector with condition that each component $\phi_i(s)$ of $\phi(s)$ is the value of a function $\phi_i(s): s \rightarrow \mathcal{R}$.

In the case of state-action value function, the feature vector ϕ represent state-action instead of state only. If there are few actions, one just composes every state-action into ϕ as follows equation.

$$\phi(s, a) = \begin{bmatrix} I(a, a_1) \cdot s \\ \vdots \\ I(a, a_n) \cdot s \end{bmatrix} \quad (20)$$

when there are n actions that $\{a_1, \dots, a_n\} \in A$ and I is the mask that

$$I(a, action) = \begin{cases} 1 & a = action \\ 0 & otherwise \end{cases} \quad (21)$$

According to equation 9, a policy can be quickly calculated as follows

$$\pi^w(s) = \underset{a}{\operatorname{argmax}} w^T \phi(s, a) \quad (22)$$

Generally, the dimension of vector w is much lesser than the dimension of value function $V(s)$, changing one weight affect value of many states. The weight is considered as generalization parameter of value function which smoothly merges states together that make learning potentially more efficient.

2.9.1 Least square policy iteration

The learning objective in the continuous case is to estimate the weight parameter w using past experience in the form of sample. Least Square Policy Iteration (LSPI) is a global algorithm for improving the policy [10]. The LSPI operates by repeatedly calling policy evaluation algorithm Least square temporal difference for state-action value function (LSTDQ), feeding new sample to LSTDQ and updating the policy weight until convergence, e.g., $\|w_t - w_{t-1}\| < \varepsilon$, where $\|\cdot\|$ denotes some vector norm, or the maximum number of iterations has been reached. LSPI is described in figure 2.4.

According to [10], there are several options for obtaining the sample for the next iteration of LSPI. One way is to freeze the sample and reuse the sample for the estimation of w in the subsequence iterations. The rational behinds this is to extract every bit of

information out of the sample. On the other hand, one could collect a new sample from the updated policy π' and use the new sample for learning.

2.9.2 Least square temporal difference for state-action value function

Least square temporal difference for state-action value function [10] is a learning algorithm for approximating the policy weight w from sample. Given a finite set of examples $\{s_t a_t r_t s'_t\}_{t=1}^n$, a matrix A is defined as

$$A = \frac{1}{n} \sum_{t=1}^n z \cdot [z - \gamma z']^T \quad (23)$$

where $z = \phi(s_t, a_t)$, $z' = \phi(s'_t, \pi^T(s'_t))$ and γ is the discount factor. Here, P^T denotes a transposition of matrix P . In the case that s' is an absorbing state, z' will be a zero vector. Further, a matrix B is defined as

$$B = \frac{1}{n} \sum_{t=1}^n z \cdot r_t \quad (24)$$

LSPI: Least Square Policy Iteration

Input : $\gamma, \phi, \varepsilon, w, n$ where

γ is discount factor

ϕ is basis function

ε is stopping threshold

w is initial policy weight

n is maximum of iteration

Output : policy weight w

1 $w' \leftarrow w$

2 $sample \leftarrow \text{collectSample}(w)$

3 $iteration \leftarrow 1$

4 **while** $iteration < n$ **and** $\|w - w'\| > \varepsilon$ **do**

5 $sample \leftarrow \text{updateSample}(w)$

6 $w \leftarrow w'$

7 $w' \leftarrow \text{LSTDQ}(\gamma, \phi, w, sample)$

8 $iteration \leftarrow iteration + 1$

9 **Return** w

Figure 2.4 LSPI algorithm

where r_t is a reward at time-step t . According to the work in [7], it has been shown that A, B , and w are related through $Aw^T = B$. The matrix A is always positive-definite so that we can solve for w by inverting A . The LSTDQ procedure is described in figure 2.5

We note that the original LSTDQ considers z that captures the dependency of the previous state and the current state.

It is possible, however, to extend the dependency towards states far back in the past [11]. The idea is to calculate z as:

$$z \leftarrow z \cdot \lambda + \phi(s, a)$$

where $0 \leq \lambda \leq 1$ is a parameter that indicates how much experience from the previous states will be accumulated.

LSTDQ: Least Square Temporal Difference for State – Action Value Function

Input : γ, ϕ, w , *sample* **where**

γ is discount factor

ϕ is basis function

w is initial policy weight

sample is historical data that each tuple consist $[s, a, r, s']$

Output : policy weight w

```

1   $A \leftarrow 0_{k,k}$ 
2   $B \leftarrow 0_{k,1}$ 
3  For each sample do
4     $z \leftarrow \phi(s, a)$ 
5    If  $s'$  is non – absorbing then
6       $\text{action} \leftarrow \pi^w(s')$ 
7       $z' \leftarrow \phi(s', \text{action})$ 
8    else
9       $z' \leftarrow 0_{k,1}$ 
10    $A \leftarrow A + z \cdot [z - \gamma z]^T$ 
11    $B \leftarrow B + z \cdot r$ 
12    $w \leftarrow A^{-1}B$ 
13 Return  $w$ 

```

Figure 2.5 LSTDQ algorithm

CHAPTER 3

Reinforcement Learning for Solving Cart-Pole Balancing Problem

This chapter will explain experiment setting for the LSPI algorithm, such as definition of state, reward function design, algorithm configuration, and experiment details.

3.1 Cart-Pole Balancing Problem Simulation

Reinforcement learning theory is converged by infinite iteration [12]. So, Agent need unlimited amount of data using for simulation. This paper uses “cart-pole balancing problem” for simulation data. It is provided by OpenAI [7]. The cart pole balancing problem is balancing the pole to not fall down and control the cart as middle point of a track.

A cart moves on a frictionless track with a pole is mounted by an unactuated joint. It is controlled by applying a force direction left or right to itself. The cart-pole starts with random position which the cart is almost at the center of the track and the pole is almost upright. The goal is to prolong episode as long as possible. The episode ends by the pole vertically diverges more than 12 degrees, or the cart moves more than 2.4 units from the center.

Originally, the task is considered success when agent made the decisions in an episode more than 195 time-steps for 100 consecutive trials. However, 195 time-steps seem too short to be the success condition for the good agent so, the minimum time-steps per episode should be extended. Practically, we cannot wait forever so, the time-steps per episode is defined as 6000. Information of cart pole provided by simulation are X, X', θ, θ' when X is horizontal position of the cart on the track, X' is the rate of change of the cart's position, θ is the pole's angle and θ' is the pole's angular velocity.

3.2 State and action definition

To apply the LSPI algorithm solving the cart-pole balancing problem, approximation of the action-value function and the reward function are needed to construct for the setting of the algorithm. First, the raw data which is obtained from simulation is defined as a state vector: $s = \{X, X', \theta, \theta'\}$. The action is defined as $a \in \{Left, Right\}$. Then, we define the state-action feature $\phi(s, a)$ as equation 19

$$\phi(s, a) = \begin{bmatrix} I(a, Left) \cdot V^T(s) \\ I(a, Right) \cdot V^T(s) \end{bmatrix} \quad (1)$$

Note that, there are two actions in this work so, dimensions of $\phi(s, a)$ and w are 8.

3.3 Reward function

The reward is feedback that reflects the consequence of action. For cart-pole balancing problem, the reward values can be either 1, 0, or -1. The reward is positive when the agent's behavior is appropriate or the action leads to a preferred state, in contrast to the negative reward. While zero reward is obscure. The preferred state for this task is pole's angle does not exceed the predefined range and cart still in track, as called as non-absorbing states. The mentioned idea leads to the definition of R_1 and R_2 as follows

$$R_1(s) = \begin{cases} 1 & s' \in \eta \\ -1 & otherwise \end{cases} \quad (2)$$

$$R_2(s) = \begin{cases} 1 & s' \in \eta \\ 0 & otherwise \end{cases} \quad (3)$$

where η is set of non-absorbing states. R_1 and R_2 is considered sparse rewards. However, by these reward functions, agent will get the same signal even the next state worse than current one. To encourage agent to do more preferred behavior, the reward should announce difference feedback between action that makes state better or worse. The new reward function is proposed as follows

$$R_3(s) = \begin{cases} 1 & s' \in \eta \wedge [(\theta > 0 \wedge \theta' < 0) \vee (\theta < 0 \wedge \theta' > 0)] \\ 0 & s' \in \eta \\ -1 & otherwise \end{cases} \quad (4)$$

R_3 is the balancing by making direction of θ' counter to the direction of θ . This kind of reward function is shaped reward. Unlike sparse reward that agent gains same portion of reward for most of the time and then occasionally gain difference value of reward when

it reached the crucial state, e.g., absorbing state that indicates success or failure of the task.

The limitation of R_3 , it does not take cart's position into account. However, the objective of this task is to maintain both of the pole's angle and the cart's position. So, the ideal reward function supposes to guide the agent to satisfy both objectives. We thus minimize non-preferred state such as the pole diverge from upright and the cart moves far from the center. The reward function is defined as follows

$$R_4(s) = \begin{cases} \text{Min}(1 - \hat{\theta}, 1 - \hat{X}) & s' \in \eta \\ -1 & \text{otherwise} \end{cases} \quad (5)$$

when $\hat{\theta}$ is the normalized degree of pole's angle for non-absorbing states and \hat{X} is the normalized cart's distance. R_4 returns maximum value when the pole's angle and the cart's position are zero.

This work proposed another reward function R_5 . The agent gain the reward from pole's angle minus by the cart's position. R_5 is inversely proportional to pole's angle. So when the pole is upright, agent gain the highest reward. However, when the cart goes too far from the center, the agent is punished. R_5 is defined as follows

$$R_5(s) = \begin{cases} e^{-(\theta^2)} - e^{-(100(x-2.4)^2)} & s' \in \eta \\ -1 & \text{otherwise} \end{cases} \quad (6)$$

when e is a natural number. The term of $e^{-(\theta^2)}$ represents as reward and the term of $e^{-(100(x-2.4)^2)}$ represents as punishment. Note that, 2.4 is the diatance from the center to the edge of the track.

3.4 Experiment

This section described the four experiments namely, 1) comparison between fixing and updating sample collection method, 2) effect of initial sample on performance of the agent, 3) effectiveness of proposed reward function, 4) impact of λ on the performance of agent.

For all experiments, we need to define various parameters. There are initial policy weight, stopping threshold, maximum of data per episode, amount of episode for each iteration, amount of iteration and discount factor. The policy weight (w) is initialized to zero vector and updated throughout the learning process. We investigate the result in the

long run so, we run LSPI algorithm for 200 iterations without the stopping threshold. We defined maximum of data per episode as 6000 time-step. The discount factor γ was set to 0.9.

Due to the simulation start with a slightly random position of cart-pole, the quantity of data per iteration should sufficient to eliminate the uncertainty. And because of success condition of the task required agent to meet the minimum time-steps for 100 consecutive trials so, amount of episode for each iteration is defined as 100 episodes.

Performance of the agent is measured in terms of time-steps and rewards that agent made. The performance is measured for every iteration. The time-steps measure as average amount of time-step that agent made per episode. The reward measure as average amount of reward that agent made per decision due to the data for each iteration may contains difference amount of time-step. This measure imply effectiveness for each decision of agent. Specific detail for each experiment is described as follows.

3.4.1 Comparison between fixing and updating sample collection method

LSPI algorithm is claimed that able to work with arbitrary sample collection method. The sample can be either fix or update over time. In addition, the size of sample can also be changed when required. This work interested in both of mentioned methods. The fixing sample method requires initial policy or any procedure for collecting data. The data is used for the entire learning process. And, the updating sample method required updating for each iteration. The method is collecting the sample with the current policy. Then agent learn a new policy and replace the old sample by the new one collecting by the new learned policy.

Propose of this experiment is to determine the essential of updating the sample. It is a comparison between two agents that have different sample updating method. Both agents get initial sample collected from the random policy. The first agent updates the sample every iteration. Updating method of the first agent is depicted in Figure 3.1. The second agent fixes the sample for the entire learning process.

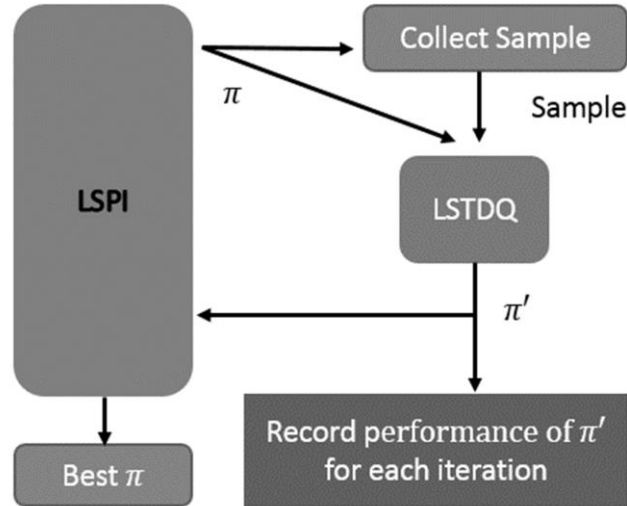


Figure 3.1 A flowchart illustrating how the sample gets updated.

3.4.2 Effect of initial sample on performance of the agent

This experiment investigate the effect of quality of initial sample on the learning of the agent. There are three sets of initial sample namely, s_1 , s_2 and s_3 . s_1 is the set of sample collected from the bad policy. s_2 is the set of sample collected from the good policy. And the s_3 is the mixed of s_1 and s_2 . Each set of initial samples contains 100 episodes of data. The good policy is the policy that made around 2300 time-step per episode. We defined the good policy weight w_1 as $w_1^T = [-1, -1, -1, -1, 1, 1, 1, 1]$. w_2 is a bad policy. It is completely opposite to w_1 . w_2 is defined as $w_2^T = [1, 1, 1, 1, -1, -1, -1, -1]$. In this experiment, the agent updates sample for every iteration.

3.4.3 Effectiveness of proposed reward function

This experiment is evaluating the effectiveness of proposed reward function. We compare the performance of three agents which using five reward functions which are R_1, R_2, R_3, R_4 , and R_5 respectively. All agents start with zero policy weight, the sample is updated every iteration. There is no initial set of sample for each agent.

3.4.4 Impact of λ on the performance of agent

This experiment will investigate the impact of λ on the learning of the agent. In general, the value of λ is between 0 and 1. The value of 1 implies that the current state affects the entire predecessor states. While the value of 0 implies that the current state

affects only one previous state. In this experiment, we compare the λ value of the set {0.0, 0.2, 0.4, 0.6, 0.8, 1.0}



ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright© by Chiang Mai University
All rights reserved

CHAPTER 4

Experimental Result

This chapter express the result of experiments mentioned in the previous chapter.

4.1 Comparison between fixing and updating sample collection method

The result shows the performance of two agents that utilized different sample updating method. There are two terms of measured performance which are average time-step per episode and reward per time-step. The blue line represents the agent that update sample every iteration, while the orange line represents the agent that fix sample for the entire learning process. In figure 4.1, both agents start with low capability. Then, the blue line growth exponentially until it reaches the goal, 6000 time-steps, around iteration 130th. In contrast, there is no improvement for the orange line. In figure 4.2, the blue line shows higher average reward than the orange line. According to figure 4.1 and figure 4.2, the agent that update sample are a lot better than the other one. It is obvious that fixing sample

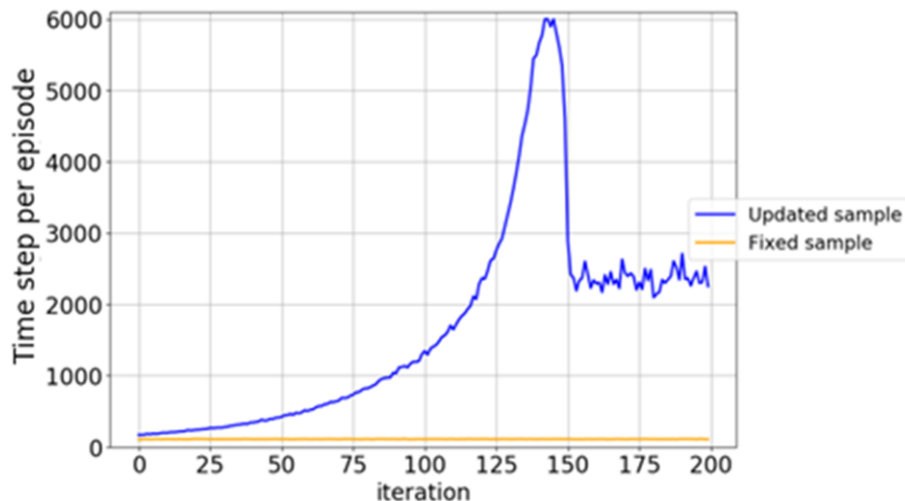


Figure 4.1 Average time-step of 100 episodes in each iteration for two agents learning with fixed sample and continuously updated sample

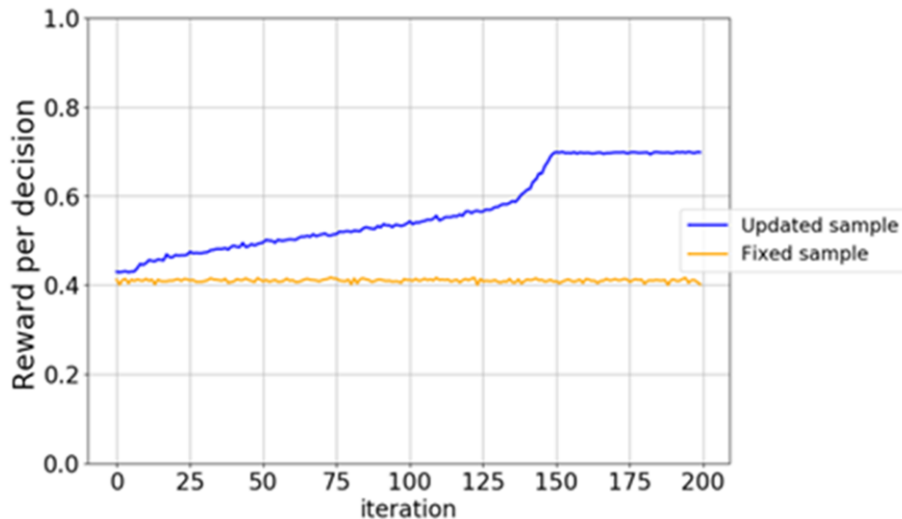


Figure 4.2 Reward per decision in each iteration for two agents learning with fixed sample and continuously updated sample

does not help agent to learn or make any progress so, updating sample for every iteration is necessary.

4.2 Effect of Initial Sample on Performance of the Agent

The result shows the performance of three agents that utilized the different initial set of sample. There are two terms of measured performance which are average time-step per episode and reward per time-step. The red line represents an agent that utilized the bad set of initial sample. The blue line represents an agent that utilized the good set of initial sample. The green line represents an agent that utilized the mix set of initial sample. In figure 4.3, the blue and green lines start with 6000 time-steps on very early iteration then decay to 2300 time-steps and keep fluctuating in a small range. The red line starts at low time-step then it improves exponentially until reach 6000 time-steps around the iteration 125th. It keeps steady for 30 iterations then decays to 2300 time-steps and fluctuates in a small range. In figure 4.4, the blue and green line converged on early iteration while the red line starts on low reward then improve and converge after the iteration 125th. The agents that utilized good and mix set of initial sample show good performance and converge on early iteration. While the agent that utilized the bad set of initial sample show bad performance at first but, it still able to improve and converge.

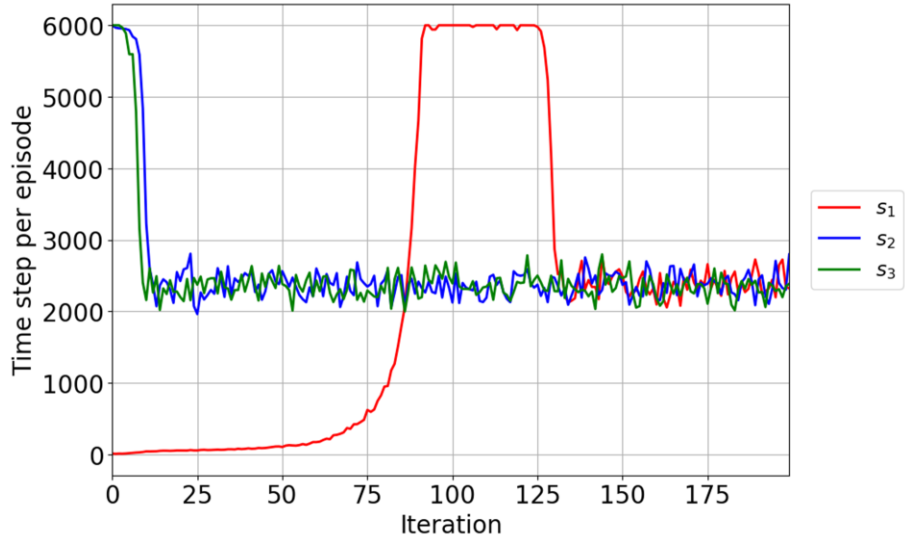


Figure 4.3 Average time-step of 100 episodes in each iteration for initial samples.

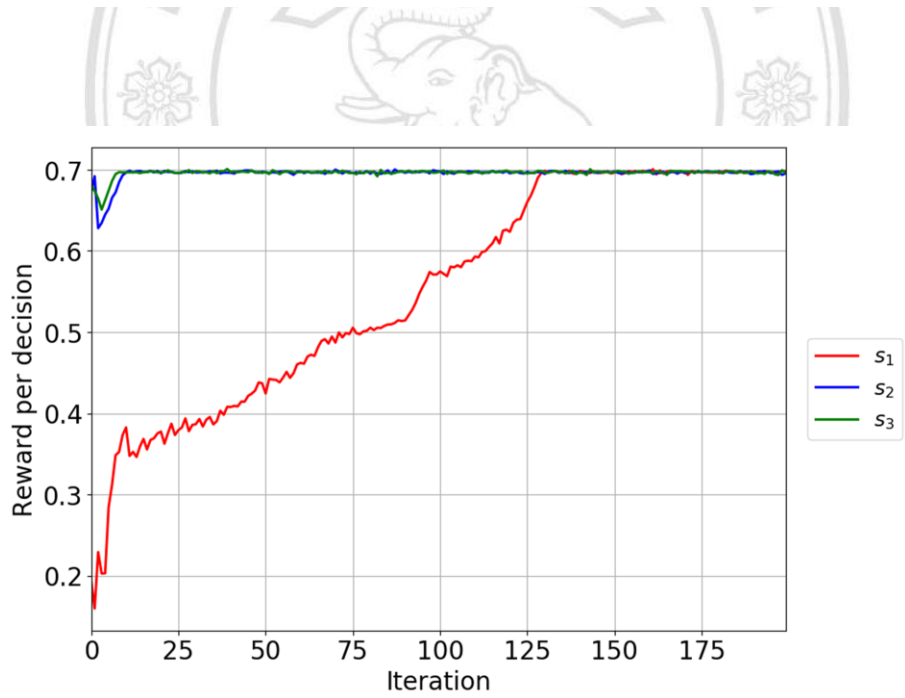


Figure 4.4 Reward per decision in each iteration for three initial samples.

4.3 Effectiveness of proposed reward function

The result shows the performance of five agents that utilized different reward functions. There are two terms of measured performance which are average time-step per episode and reward per time-step. The red and blue lines represent agent that utilize reward functions R_1 and R_2 , respectively. The green, orange and purple lines represent agent that utilize proposed reward functions R_3 , R_4 and R_5 , respectively

In figure 4.5, the green line is the only one that reaches 6000 time-step while other lines show very low time-step through learning process. The green line start at low time-step on early iteration. Then, it growth exponentially until reaches the goal, 6000 time-steps, around iteration 110th. After that, it keeps steady for 30 iterations then decays to 2300 time-steps and fluctuates in small range.

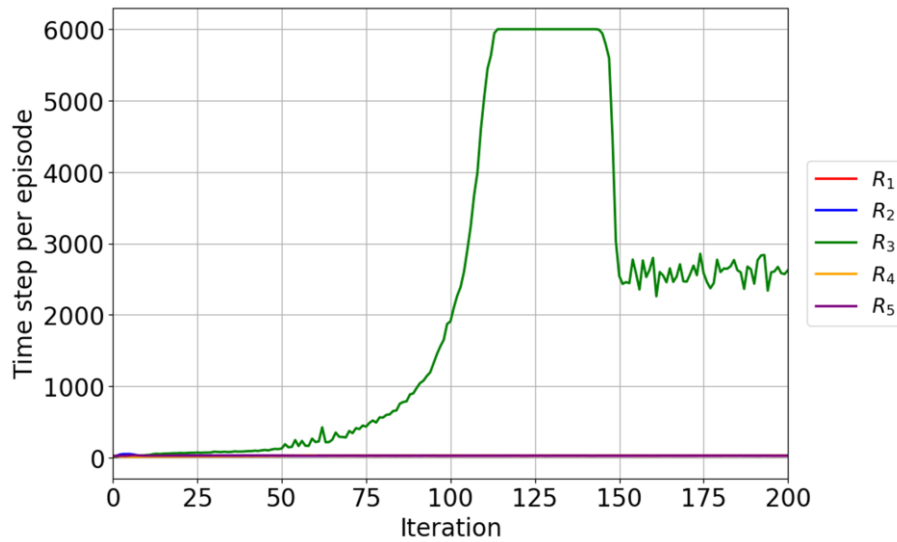


Figure 4.5 Average time-step of 100 episodes in each iteration for five reward functions

In figure 4.6, the green line is the only one that improve then converge on late iteration. It reaches the highest average reward per time-step after iteration 150th, which the performance has already reduced, instead of the iteration that agent made 6000 time-steps. The orange line shows improvement at first but decay and converge on later iteration. Other lines show no improvement through learning process. As can be observed from the result, reward function plays an important role in the success of the learning. Since R_3 does not take the cart position into account, it will not respond to event when the cart going to break the track. This may be the reason why the performance reduced to

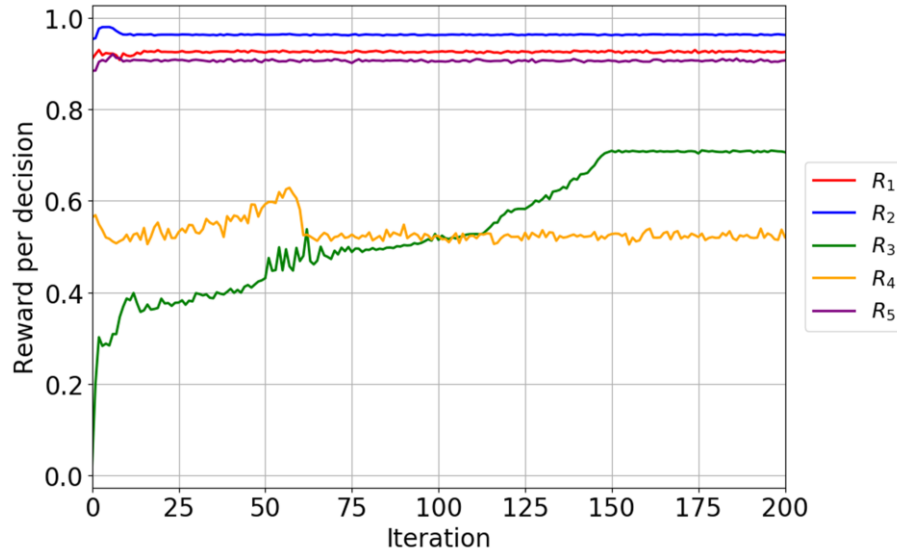


Figure 4.6 Reward per decision in each iteration for five reward functions

2300 time-steps on late iteration. By the way, although R_3 is not the perfect one, it still able to lead agent to archive the goal.

4.4 Effect of λ on the Performance of the Policy

Performance of agents that use difference λ , namely, 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0, is shown in Figure 4.6. The graph show performance in terms of average time-step per episode. Agents that use a small value of λ , e.g., 0.0, 0.2 and 0.4 can reach 6000 time-steps while agents that use other larger values converged just below 120 time-steps. According to theory, competent λ value is specific to learning Configuration. In this case, $\lambda = 0.2$ is the best value because agent that utilize this value is the fastest one that can reach the goal. As can be seen, utilizing λ can speed up learning process, comparing to the default setting that considerate $\lambda = 0$. However, the value needs to be carefully defined, due to it ruins the learning when agent uses inappropriate value.

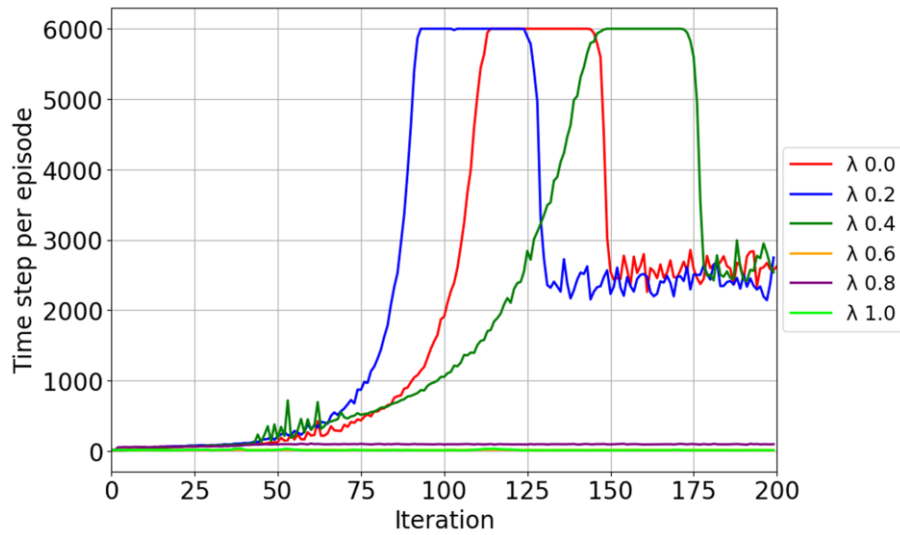


Figure 4.7 Average time-step of 100 episodes in each iteration for lambda values of 0.0, 0.2, 0.4, 0.6, 0.8 and 1.0



ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
 Copyright© by Chiang Mai University
 All rights reserved

CHAPTER 5

Conclusion and Discussion

This chapter express discussion and conclusion of this work.

5.1 Discussion

In this section, there are discussion of four result of experiments that mentioned in the previous chapter.

According the experiment result 4.1, we found that updating sample due iteration obviously help agent to learn to archive the task. In contrast, the agent that learn from fix sample has no improvement, considering from both performance measures.

According the experiment result 4.2, we found that the good and mixed quality set of initial sample help agent to archive the goal on early iteration. While the bad set of initial sample policy does not boost the learning process. This lead to awareness of mix quality set of initial sample that it should not contain too much of data collected from bad policy. However, the agent that utilized the bad set of initial sample still able to archive the goal. For this problem, initial sample is a safe option with none of negative effect

According the experiment result 4.3, we suggest that reward function is the major factor leading agent to archive the goal. Only the proposed reward function R_3 able to guide agent to achieve the goal, while other reward functions are failure. R_1 and R_2 always reward agent with the same value, as long as the next state is non-absorbing state is the reason that make R_1 and R_2 inferior to the R_3 . They return the same value even though the agent made a bad decision that transfers itself to a worse state, i.e., state that the pole diverge further from upright position. On the other hand, reward function R_3 treat agent in a difference way. It returns positive value only if the action leads to preference state. It ignores if the next state is not the preferred one and punishes if the task failure. This seems encourages agent to take the right decision according to our common sense. The proposed reward function R_3 work well but not perfect due to it not take cart position into account

so, it does not respond to event when the cart going to break the track. This may be the reason why the performance reduced to 2300 time-steps on late iteration. Anyway, this still acceptable because policy has been recorded for every iteration so, one can choose the high-performance policy from the record. Considering from the performance in term of reward per time-step, R_4 seems unlikely to be the right idea because it show reducing during the learning process. While the idea of R_5 are more likely to employ. We adapt the idea of R_3 and R_5 that can be seen in appendix section.

According the experiment result 4.4, the result confirms that lambda value can speed up learning process. Comparing to the lambda value 0, some of the lambda value help agent to archive the task faster. However, some of them slow down or even ruin the learning process when the lambda value is too high. High lambda value fail to speed up the learning process because they make the current event that agent facing affects too much of the predecessor states which may not relate to the current one anymore. This lead to awareness of defining the lambda value.

5.2 Conclusion

In this work, we studied reinforcement learning for solving cart-pole balancing problem which the track length is limited by using LSPI algorithm. The problem is considered in continuous state space. We proposed new rewards function which is competent for guiding the agent towards optimal control strategy. We employ the LSPI algorithm with various reward functions. According to experiments, we found that agent that utilize updated sample with the reward function R_3 achieve that goal. In addition, we confirmed that lambda value can speed up that learning process. The initial sample are the safe option when it possible to provided.

REFERENCES

- [1] R. C. Dorf and R. H. Bishop, *Modern Control Systems* (12th Edition), Pearson, 2010.
- [2] K. M. Passino and S. Yurkovich, *Fuzzy Control*, Addison-Wesley, 1997.
- [3] A. G. Barto, R. S. Sutton and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Transactions on Systems, Man, and Cybernetics*, Vols. SMC-13, pp. 834-846, 9 1983.
- [4] B. D. Nichols, "A comparison of action selection methods for implicit policy method reinforcement learning in continuous action-space," in *2016 International Joint Conference on Neural Networks (IJCNN)*, 2016.
- [5] A. Weinstein and M. Botvinick, "Structure Learning in Motor Control: A Deep Reinforcement Learning Model," *CoRR*, vol. abs/1706.06827, 2017.
- [6] L. Busoniu, D. Ernst, B. D. Schutter and R. Babuška, "Online least-squares policy iteration for reinforcement learning control," in *Proceedings of the 2010 American Control Conference*, 2010.
- [7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba, *OpenAI Gym*, 2016.
- [8] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning), A Bradford Book, 1998.
- [9] R. A. Howard, *Dynamic Programming and Markov Processes*, Cambridge, MA: MIT Press, 1960.
- [10] M. G. Lagoudakis and R. Parr, "Least-squares Policy Iteration," *J. Mach. Learn. Res.*, vol. 4, pp. 1107-1149, 12 2003.
- [11] J. A. Boyan, "Technical Update: Least-Squares Temporal Difference Learning," *Machine Learning*, vol. 49, pp. 233-246, 01 11 2002.

- [12] E. Patricia Brunskill, "Compact parametric models for efficient sequential decision making in high-dimensional, uncertain domains," 3 2010.



ลิขสิทธิ์มหาวิทยาลัยเชียงใหม่
Copyright© by Chiang Mai University
All rights reserved

APPENDIX A

Additional reward function

R_3 is the only proposed reward function that able to lead agent to archive the goal. However, it is not sensible by ignore the cart's position. To improve the reward, we alter the R_3 by punishing agent when it going to make the cart move out of track. The improved reward function R_6 is defined as follows

$$R_6(s) = \begin{cases} 1 - e^{-(100(X-2.4)^2)} & s' \in \eta \wedge [(\theta > 0 \wedge \theta' < 0) \vee (\theta < 0 \wedge \theta' > 0)] \\ 0 - e^{-(100(X-2.4)^2)} & s' \in \eta \\ -1 & \text{otherwise} \end{cases} \quad (1)$$

when e is a natural number. Like R_5 , the highest vale of term $e^{-(100(x-2.4)^2)}$ is 1 when the cart's position: $X = 2.4$.

An experiment is conducted to validate effectiveness of R_6 . Setting of the experiment is as same as the experiment in section 3.4.3. In addition, ending of episode is measured in this experiment. There are three kind ending in this work namely, "pole down", "out of track" and "time". "Pole down" and "out of track" represent the event

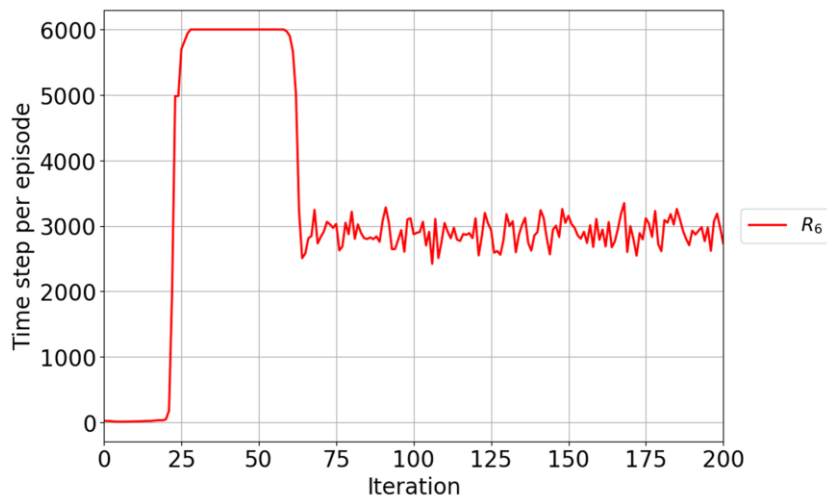


Figure A.1 Average time-step of 100 episodes in each iteration for R_6

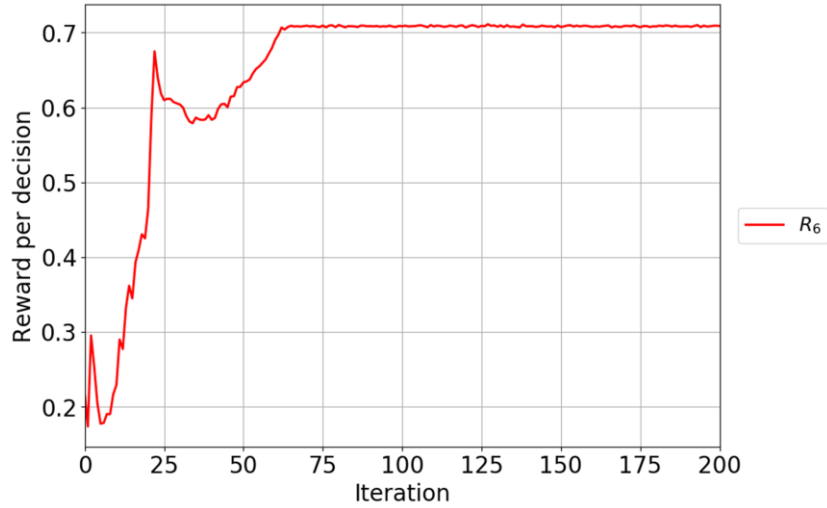


Figure A.2 Reward per decision in each iteration for R_6

when the pole's angle and the cart's position exceed the limit, respectively. While "time" represent the event when the agent reach 6000 time-step.

According to figure A.1, the agent archive the goal on early episode. However, time-step remain still and decay on iteration 60th. Reward per decision in figure A.2 confirmed that the learning converge after the performance reduced. Figure A.3 show percent for each kind episode ending. The graph confirms that the cart breaking the track cause reducing of the performance after iteration 60th.

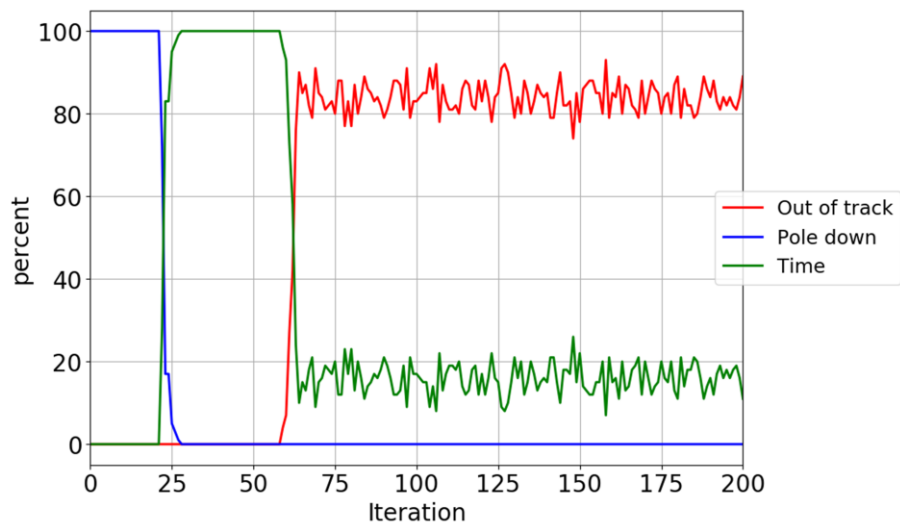


Figure A.3 percent of episode ending in each iteration for R_6

APPENDIX B

Stability and reproducibility of reinforcement learning

Steady of reproducing is an issue in reinforcement learning. We used to conduct an experiment with the same setting for 20 trials. There are 100 episodes of data for each iteration. The result is shown in figure B.1. There are 20 lines that represent each trial. The graph shows that the agent does not performs stably. Overall performance of the agent is shown figure B.2. The line represents average time-step per episode of 20 trials. The 25th percentile and the 75th percentile are boundary of the shadow area. The graph shows high variance since the 120th iteration.

We also conducted another experiment which increased amount of data per iteration up to 500 episodes. There are 20 trials for this experiment. Overall performance of the agent is shown figure B.3. Comparing to figure B.2, the shadow area is significantly reduced. However, this experiment consumed a lot more time than the previous one.

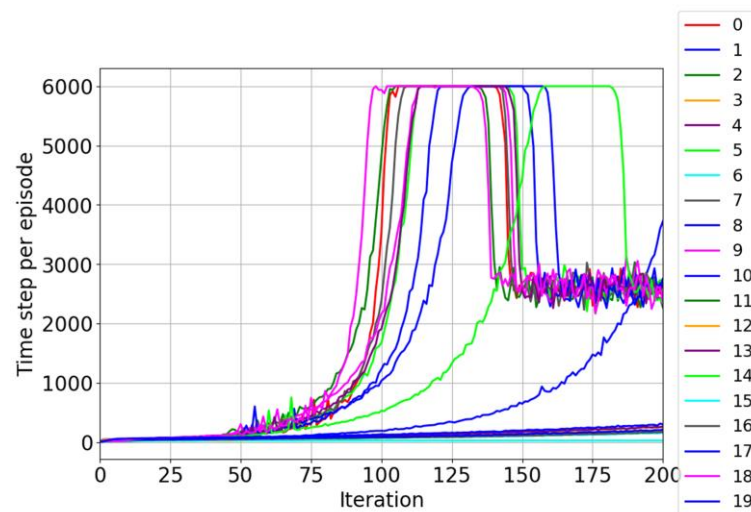


Figure B.1 Average time-step of 100 episodes in each iteration for 20 trials

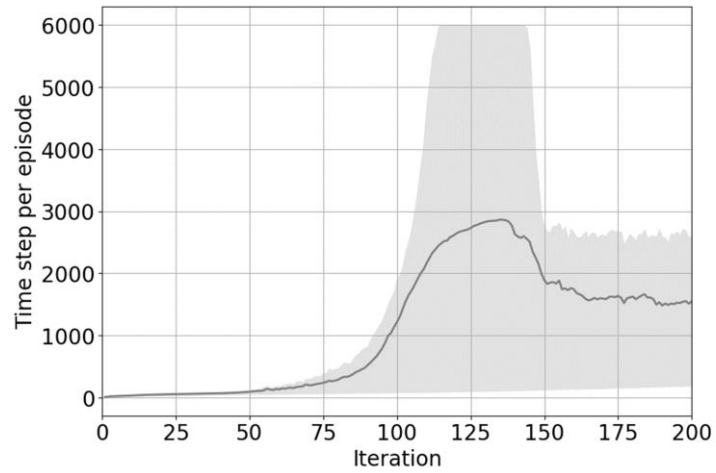


Figure B.2 Average time-step of 20 trials which each iteration contain 100 episodes of data

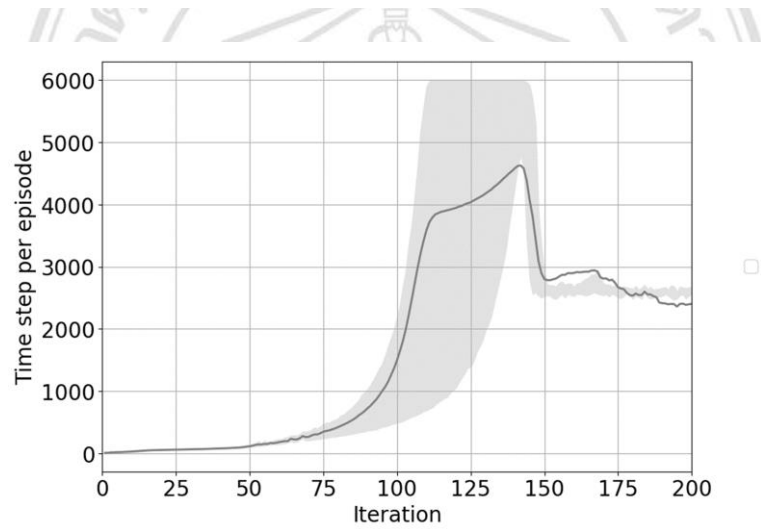


Figure B.3 Average time-step of 20 trials which each iteration contain 500 episodes of data

Copyright © by Shanghai Jiao Tong University
All rights reserved

CURRICULUM VITAE

Name Mr.Sa-ngapong Panyakaew

Date of birth 19th November 1992

Education 2014 Bachelor of Science, Computer Science, Faculty of Science, Chiangmai University

Publication S. Panyakaew, P. Inkeaw, J. Bootkrajang, & J. Chaijaruwanich, “Least Square Reinforcement Learning for Solving Inverted Pendulum Problem”, 2018 3rd International Conference on Computer and Communication Systems (ICCCS 2018), Nagoya, Japan



มหาวิทยาลัยเชียงใหม่
Copyright © by Chiang Mai University
All rights reserved