Automatic generation of fuzzy neural networks via reinforcement learning with applications in path planning of mobile robots

Zhou, Yi

2007

Zhou, Y. (2007). Automatic generation of fuzzy neural networks via reinforcement learning with applications in path planning of mobile robots. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/13267>

<https://doi.org/10.32657/10356/13267>

Automatic Generation of Fuzzy Neural Networks via Reinforcement Learning with Applications in Path Planning of Mobile Robots

Zhou Yi

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirements for the degree of

Doctor of Philosophy

2007

Statement of Originality

I hereby certify that the work embodied in this thesis is the result of original research and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

Zhou Yi

Acknowledgments

It is an unforgettable and deeply cherished experience in my life to have pursued my Ph.D degree at the School of Electrical and Electronic Engineering, Nanyang technological University. I would like to take this opportunity to express my sincere gratitude to numerous people who have assisted me in one way or another in my research work.

First and foremost, I am most indebted to my supervisor, Associate Professor Er Meng Joo, for his professional guidance, constant support and trust. His insightful comment has been an inspiration for my work. It has been a great pleasure to have professor Er as my advisor.

I am grateful to Profs. Narasimhan Sundararajan, Wlodzislaw Duch, Cheah Chien Chern, Ponnuthurai Nagaratnam Suganthan, Eric Sung, Wang Han, Song Qing, and et al. for their great teaching, guidance and advice for the postgraduate courses.

I would like to thank Drs. Gao Yang, Deng Chang, Zhou Changjiu, Hu Qinglei, Sun Shuli, Xie Li and et al for their help and numerous valuable ideas obtained from discussion with them.

I would like to express my gratitude to the School of Electrical and Electronic Engineering, Intelligent Systems Center and Process Instrumentation Lab for their technical and financial support.

The last but not the least, my sincere thanks go to all the members of my family.

This thesis is dedicated to my parents and girl friend for their love and support.

Table of Contents

Acknowledgments	i
Table of Contents	ii
Summary	vii
Abbreviations	x
List of Figures	xii
List of Tables	xvii
1 Introduction	1
1.1 Motivation	1
1.1.1 Motivation based on fuzzy neural networks	1
1.1.2 Motivation based on machine learning	3
1.1.3 Motivation based on robot learning	6
1.2 Major Contributions of the Thesis	7
1.3 Outline of Thesis	12
2 Fuzzy Systems and Neural Networks	15

TABLE OF CONTENTS

iii

2.1	Fuzzy Inference Systems	16
2.1.1	Fuzzy logic and fuzzy systems	16
2.1.2	Fuzzy membership functions	17
2.1.3	Fuzzy intersection and union	20
2.1.4	Fuzzy If-Then rules	22
2.1.5	Mechanism of Fuzzy Inference Systems (FISs)	24
2.1.6	Three types of Fuzzy Inference Systems (FISs)	26
2.2	Neural Networks and Fuzzy Neural Networks	30
2.3	Combination of Fuzzy and Neural Systems	31
2.4	Design Issues in Fuzzy Neural Networks	33
3	Machine Learning	35
3.1	Overview of Machine Learning	35
3.2	Supervised Learning	36
3.3	Unsupervised Learning	37
3.4	Reinforcement Learning	39
3.4.1	Basic elements of reinforcement learning system	39
3.4.2	Reinforcement learning versus supervised learning	41
3.4.3	Temporal-difference learning and Q-learning	42
3.4.4	Reinforcement learning with fuzzy systems	45
4	Dynamic Self-Generated Fuzzy Neural Networks	48
4.1	A Typical Structure of Fuzzy Neural Networks	49
4.2	Previous Approaches of Generating the FNN	51

TABLE OF CONTENTS

iv

4.3	Fuzzy Q-Learning	54
4.3.1	Fuzzy structure for Q-learning	55
4.3.2	Local action exploration and selection	56
4.3.3	Generation of global actions	57
4.3.4	Update of Q-value	57
4.3.5	Eligibility traces	58
4.4	Dynamic Self-Generated Fuzzy Q-Learning	59
4.4.1	ϵ -Completeness criterion for input space partitioning	59
4.4.2	Allocation of newly generated rules	60
4.4.3	Global TD error evaluation	62
4.4.4	Local TD error evaluation	64
4.4.5	Participation of fuzzy rules	64
4.4.6	Reinforcement learning at premise-generation	64
4.4.7	Restructure of FNNs via reinforcement learning	65
4.4.8	Pruning of redundant fuzzy rules	67
4.4.9	Gradualism of learning	67
4.5	Incorporation of Expert Knowledge in the DSGFQL System	69
4.6	Summary	71
5	Simulation Studies and Experiments on Mobile Robot Control	74
5.1	The Khepera Robot	74
5.2	Wall Following Task for Khepera Robot	76
5.3	Basic Fuzzy Controller	78

TABLE OF CONTENTS

v

5.4	Fixed Fuzzy Q-Learning	79
5.5	DSGFQL Approach for Wall-following	81
5.6	Adaptation in a New Environment	84
5.7	Other Dynamic FQL Approaches	87
5.8	Dynamically Changing Environments	92
5.9	Obstacle Avoidance Task	95
5.10	Summary	97
6	Extended DSGFQL Approaches I: Premise Generation	99
6.1	Reward Scheme for the DSGFQL Algorithm	100
6.1.1	Global and local reward value functions	100
6.1.2	Reward scheme of the DSGFQL	101
6.1.3	Wall-following task by the reward value scheme	102
6.1.4	Performances with different parameter settings	105
6.1.5	Dynamically changing environments	106
6.2	Enhanced DSGFQL with Extended SOM	108
6.2.1	Hybrid DSGFQL with the SOM	108
6.2.2	Simulation studies of wall following	109
6.2.3	Obstacle avoidance task	113
6.3	Summary	116
7	Extended DSGFQL Approaches II: Consequents Generation	117
7.1	Continuous Action Q-Learning	118
7.2	Hybrid DSGFQL with Continuous Action Q-Learning	120

TABLE OF CONTENTS

vi

7.2.1	Local action selection	120
7.2.2	Global action determination	121
7.2.3	Update Q-values	122
7.2.4	Simulations and comparison studies	123
7.3	Summary	127
8	Conclusions and Recommendations	128
8.1	Conclusions	128
8.2	Recommendations for Further Research	131
8.2.1	Convergence property	131
8.2.2	Evolutionary algorithms	131
8.2.3	Incomplete sensor information and noisy data	132
8.2.4	Swam robots and multi-agent systems	132
	Author's Publications	134
	Bibliography	137

Summary

In this thesis, a novel Reinforcement Learning (RL) methodology, termed Dynamic Self-Generated Fuzzy Q-Learning (DSGFQL) is developed for generating Fuzzy Neural Networks (FNNs).

In the DSGFQL system, RL is adopted for both structure identification and parameters estimation of FNNs. Structure and premise parameters can be dynamically adjusted according to reinforcement evaluations. Besides evaluation signals for system performance, a reinforcement sharing mechanism is adopted for evaluating contributions of each fuzzy rules. Therefore, both system performance and individual contributions of each fuzzy rules can be evaluated through reinforcement signals. Fuzzy rules with good contributions can be reinforced while fuzzy rules with poor contributions will be penalized or eliminated. Therefore, structure and premise parts of FNNs can be determined in an RL manner.

The DSGFQL offers a novel view of generating FNNs. RL methodologies are not only applied for selecting optimal actions (consequent parameters) but also applied in determining the number of rules, pruning and adjusting premise parameters. Similarly as reinforcing good actions and penalizing poor actions in conventional RL approaches, good *rules* are promoted while bad *rules* are demoted or eliminated in the DSGFQL method. Therefore, instead of only focusing on applying RL in training consequent parameters (consequent-generation), RL is adopted at a higher level (premise-generation level).

As structure and premise parameters of FNNs can be adjusted according to reinforcement evaluations, efficient structure can be determined through the DSGFQL

method.

The novel DSGFQL methodology can automatically create, delete and adjust fuzzy rules according to the evaluation of system performance as well as contributions from individual fuzzy rules. The whole learning process is based on evaluative information and it does not required instructive training data or many human efforts.

Besides self-generating FNNs without a *priori* structure, the DSGFQL approach can also be incorporated by domain knowledge from human experts or from previous training. At premise level, initial domain knowledge about tasks can be incorporated as bias into the system by If-Then fuzzy rules. An NN structure for incorporating bias components is proposed according to the confidence in the initial knowledge. Therefore, rapid and safe learning can be achieved. At consequents training level, a sharing mechanism is proposed to initialize Q-values of newly generated rules when applying the Q-learning. Instead of randomly assigning Q-values of new rules, Q-values are initialized according to those existing neighboring values. Therefore, previous knowledge can be learned from those neighboring fuzzy rules and learning speed can be increased.

Furthermore, extended studies for further developing the DSGFQL algorithm are carried out. For non-Temporal Difference (TD)-based RL approaches, a reward function scheme (DSGFQL-reward) is proposed as a general approach for all RL problems. Global and local rewards are adopted as evaluation criteria for system and local performances respectively. As reward function is a basic element for all RL problems, including non-TD-based approaches, the reward scheme offers a general RL methodology for generating FNNs.

Moreover, an enhanced version of the DSGFQL termed Enhanced Dynamic Self-Generated Fuzzy Q-Learning (EDSGFQL) is proposed by combining the DSGFQL with an extended Self-Organizing Map (SOM) algorithm. An extended SOM is proposed and adopted to adjust the center positions of fuzzy neurons for better feature representation. With better allocation of fuzzy neurons, the original DSGFQL

is enhanced and the number of fuzzy rules can be further reduced.

Besides extensional approaches in determining premise parameters of FNNs, continuous action Q-learning is combined with the DSGFQL in generating local continuous actions. Therefore, besides applying fuzzy inference for generating continuous global actions, local continuous actions can also be obtained instead of discrete ones from each local fuzzy rules. In the DSGFQL-CA approach, continuous consequent parameters are estimated instead of discrete ones.

The DSGFQL algorithm and its extended methodologies are applied in robotics tasks for navigation such as wall-following and obstacle avoidance tasks. Comparison studies with other existing fuzzy RL approaches demonstrate the superiority of the proposed methods as more efficient FNNs can be generated. A number of comparative studies are carried out to validate the viability of the proposed approaches in both static and dynamic training environments.

Abbreviations

<i>AI</i>	<i>Artificial Intelligence</i>
<i>COA</i>	<i>Centroid Of Area</i>
<i>COG</i>	<i>Center Of Gravity</i>
<i>CQGAF</i>	<i>Clustering and Q – value based Genetic Algorithm learning schemes for Fuzzy system design</i>
<i>DFNN</i>	<i>Dynamic Fuzzy Neural Network</i>
<i>DFQL</i>	<i>Dynamic Fuzzy Q – Learning</i>
<i>DP</i>	<i>Dynamic Programming</i>
<i>DSGFQL</i>	<i>Dynamic Self – Generated Fuzzy Q – Learning</i>
<i>DSGFQL – CA</i>	<i>Dynamic Self – Generated Fuzzy Q – Learning and Continuous Action Q – learning</i>
<i>EBF</i>	<i>Ellipsoidal Basis Function</i>
<i>EDSGFQL</i>	<i>Enhanced Dynamic Self – Generated Fuzzy Q – Learning</i>
<i>FIS</i>	<i>Fuzzy Inference System</i>
<i>FNN</i>	<i>Fuzzy Neural Network</i>
<i>FQL</i>	<i>Fuzzy Q – Learning</i>
<i>GA</i>	<i>Genetic Algorithm</i>
<i>GDFNN</i>	<i>General Dynamic Fuzzy Neural Network</i>
<i>MF</i>	<i>Membership Function</i>
<i>MOM</i>	<i>Mean Of Maximum</i>
<i>MOMO</i>	<i>Multi – Input Multi – Output</i>
<i>MOSI</i>	<i>Multi – Input Single – Output</i>

Abbreviations

xi

<i>NN</i>	<i>Neural Network</i>
<i>RBF</i>	<i>Radial Basis Function</i>
<i>RL</i>	<i>Reinforcement Learning</i>
<i>SL</i>	<i>Supervised Learning</i>
<i>SOM</i>	<i>Self – Organizing Map</i>
<i>TD</i>	<i>Temporal Difference</i>
<i>TSK model</i>	<i>Takagi – Sugeno – Kan model</i>
<i>UL</i>	<i>Unsupervised Learning</i>

List of Figures

2.1	Fuzzy membership functions	19
2.2	Fuzzy inference systems	24
2.3	Tsukamoto-type fuzzy inference system	27
2.4	Mamdani-type fuzzy inference system	28
2.5	TSK-type fuzzy inference system	29
2.6	RBF neural networks	31
3.1	The interactions among learning, performance, knowledge and the environment.	36
3.2	Agent-environment interaction in reinforcement learning	41
4.1	Structure of the Fuzzy Neural Networks System	50
4.2	Consequent parts of the FQL system	56
4.3	Neural network structure with incorporated knowledge	70
4.4	Feedback controller with fuzzy reinforcement learning	72
4.5	Flowchart of the learning algorithm for DSGFQL	73
5.1	The miniature mobile robot: Khepera	75
5.2	Position and orientation of sensors on the Khepera	76
5.3	Actual environment for wall-following experiments	77

LIST OF FIGURES

xiii

5.4	MATLab simulation environment for wall-following experiments . .	77
5.5	Number of failures for comparison of performances of fuzzy controllers for (a)Basic fuzzy controller with 16 fuzzy rules, (b)16 fuzzy rules based on FQL, (c)81 fuzzy rules based on FQL	80
5.6	Reward values for comparison of performances of fuzzy controllers for (a)Basic fuzzy controller with 16 fuzzy rules, (b)16 fuzzy rules based on FQL, (c)81 fuzzy rules based on FQL	81
5.7	Number of failures for comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) DSGFQL	82
5.8	Reward values for comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) DSGFQL	83
5.9	Number of rules generated by the DSGFQL approach	83
5.10	Membership functions after learning at one run	84
5.11	New training environment for wall-following	85
5.12	Performance comparison of DSGFQL with training directly and re-training in a new environment: Number of failures vs number of episodes.	85
5.13	Performance comparison of DSGFQL with training directly and re-training in a new environment: Reward value vs number of episodes.	86
5.14	Performance comparison of DSGFQL with training directly and re-training in a new environment: Number of fuzzy rules vs number of episodes.	86
5.15	Comparison of performances of DFQL, CQGAF and DSGFQL in the wall-following task	89
5.16	Comparison of performances of DFQL, CQGAF and DSGFQL in first 20 episodes of the wall-following task	90

LIST OF FIGURES

xiv

5.17	Dynamically changing environments for training.	92
5.18	Comparison of performances of DFQL, CQGAF and DSGFQL in dynamically changing environments of the wall-following task: Number of failures vs number of episodes.	93
5.19	Comparison of performances of DFQL, CQGAF and DSGFQL in dynamically changing environments of the wall-following task: Reward value vs number of episodes.	93
5.20	Comparison of performances of DFQL, CQGAF and DSGFQL in dynamically changing environments of the wall-following task: Number of fuzzy rules vs number of episodes.	94
5.21	Training environment for the obstacle avoidance task.	96
6.1	Comparison of performances of reward and TD error schemes of the DSGFQL in the wall-following task: Number of failures vs number of episodes.	103
6.2	Comparison of performances of reward and TD error schemes of the DSGFQL in the wall-following task: Reward value vs number of episodes.	104
6.3	Comparison of performances of reward and TD error schemes of the DSGFQL in the wall-following task: Number of fuzzy rules vs number of episodes.	104
6.4	Comparison of performances of reward and TD error schemes of the DSGFQL in dynamically changing environments of the wall-following task: Number of failures vs number of episodes.	106
6.5	Comparison of performances of reward and TD error schemes of the DSGFQL in dynamically changing environments of the wall-following task: Reward value vs number of episodes.	107

LIST OF FIGURES

xv

6.6	Comparison of performances of reward and TD error schemes of the DSGFQL in dynamically changing environments of the wall-following task: Number of fuzzy rules vs number of episodes.	107
6.7	Flowchart of the learning algorithm for EDSGFQL	110
6.8	Comparison of performances of wall-following by DFQL, CQGAF (without GA) and EDSGFQL (reward) with reward scheme (1-40 episodes)	111
6.9	Comparison of performances of DSGFQL and EDSGFQL (reward scheme) in the wall-following task	114
7.1	Comparison of performances of DSGFQL and Continuous Action Q-learning in the wall-following task: Number of failures vs number of episodes.	119
7.2	Comparison of performances of DSGFQL and Continuous Action Q-learning in the wall-following task: Reward value vs number of episodes.	120
7.3	Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Number of failures vs number of episodes.	123
7.4	Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Reward value vs number of episodes.	124
7.5	Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Number of failures vs number of episodes (1-10runs).	124

LIST OF FIGURES

xvi

7.6	Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Reward value vs number of episodes (1-10runs).	125
7.7	Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning (7 actions) in the wall-following task: Number of failures vs number of episodes.	126
7.8	Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning (7 actions) in the wall-following task: Reward value vs number of episodes.	126

List of Tables

5.1	Basic fuzzy control rules for wall following	78
5.2	Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following	88
5.3	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following	88
5.4	Mean value of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task . . .	94
5.5	Standard derivations of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task	94
5.6	Mean value of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance	96
5.7	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance	96
5.8	Comparison among basic FCL, FQL, DFQL, CQGAF and DSGFQL	98
6.1	Mean value of failures, rewards and fuzzy rules after 20 episodes for the wall following task with TD error scheme	103
6.2	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for the wall following task with TD error scheme	103

6.3	Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following with different threshold settings for the DSGFQL (reward) approach	105
6.4	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following with different threshold settings for the DSGFQL (reward) approach	105
6.5	Different sets of parameters	105
6.6	Mean value of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task . . .	106
6.7	Standard derivations of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task	106
6.8	Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following	112
6.9	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following	112
6.10	Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following	113
6.11	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following	113
6.12	Mean value of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance	115
6.13	Standard derivations of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance	115
6.14	Comparison among DFQL, CQGAF, DSGFQL and EDSGFQL . . .	115

Chapter 1

Introduction

1.1 Motivation

1.1.1 Motivation based on fuzzy neural networks

Fuzzy logic is a mathematical approach to emulate the human way of thinking. It reconciles conventional precise mathematical paradigms and human-like decision making.

Fuzzy logic is conceptually easy to understand, flexible, tolerant of imprecise data and based on natural language. The main reasons for implementing fuzzy logic are two-fold. One is due to the complication of precise descriptions and the other is the incorporation of human knowledge [1]. Therefore, fuzziness is introduced to obtain a reasonable, yet trackable, model for approximation and fuzzy logic is regarded as a mathematical approach of modelling the human way of thinking and learning. It has been shown that fuzzy logic-based modelling and control could serve as a powerful methodology for dealing with imprecision and nonlinearity efficiently in [2, 3].

Conventional approach for designing a fuzzy system is a subjective approach which is dependent much on the expert domain knowledge. This subjective approach usually meets trouble when human beings have little or insufficient knowledge

about the system [4].

If controlling rules are complicated, it is hard for human beings to obtain correct rules for the system. Sometimes, people might be able to specify input-output pairs or express their expectations but they cannot identify a concise relationship between inputs and the desired outputs. It may be difficult or impossible for people to describe the rules. Therefore, the controlling system is expected to be able to adjust the internal structure to produce correct outputs and thus suitably meet the expectations. Instead of a purely subjective design, automatic approaches are highly desired for helping people to generate or improve fuzzy systems. In addition, certain characteristics of the working environment might not be completely known at the design time. Therefore, learning capability is requested to enhance fuzzy system for dynamic changing situations. A strategy which can provide training capability to fuzzy systems and retain the capability of incorporating human expert knowledge is highly desired.

To circumvent the problems of lacking systematic design and learning capability, fuzzy logic is usually combined with Neural Networks (NNs) which were first introduced by McCulloch and Pitts in the work of [5]. The technology of Fuzzy Neural Networks (FNNs) combines the profound learning capability of NNs [6] with the mechanism of explicit and easily interpretable knowledge presentation provided by fuzzy logic. In essence, FNN is able to represent meaningful real-world concepts in comprehensive knowledge bases. With the adaptive nature of NNs, this knowledge can be gained autonomously through learning [3, 7–10].

This thesis centers around methodologies for self-generated FNNs with topics on but not limited to:

Systematic design We are looking for systematic designs of FNNs, where procedures of generating FNNs should be clearly defined and easy to implement.

Structure and parameter learning In this thesis, we look forward to methodologies of both structure identification (e.g. determination of input-space partition, number of Membership Functions (MFs) and number of rules) as

well as parameter estimation (e.g. adjustment of both premise and consequent parameters).

Efficient structure The generated FNNs should be able to consider sufficient rules in critical state space and exclude redundant rules in unimportant or unvisited state space. To reduce computational cost, dormant or unnecessary rules should be eliminated from the system. Therefore, pruning techniques are one of our main interests.

Dynamic generation We are interested in dynamic generation of FNNs which can dynamically create, adjust and delete fuzzy rules according to the performance. The learning algorithm should adapt to dynamically changing environments.

Learning without teacher The learning system should be able to be used based on only some suitable evaluative or critic feedback information but without any detailed correct or instructive actions. By this means, this kind of learning method represents a very flexible tool, as less informative learning source is needed.

Integration of domain knowledge Besides numeric training, human expert knowledge and the *priori* knowledge obtained from previous training should be incorporated into the system in order to reduce the training time and avoid critic failures.

1.1.2 Motivation based on machine learning

Machine learning which concerns both cognitive psychology and Artificial Intelligence (AI) has attracted great attention since the birth of computers. Mechanical inventions and applications on machines have extended human being's physical capabilities (i.e. the abilities to act) while the development of machine learning will surely extend human being's mental capabilities (i.e. the abilities to perceive, reason and decide).

Several important engineering reasons for machine learning are given in [11] as below:

- Sometimes, people might be able to specify input-output pairs or express their expectations but they cannot identify a concise relationship between inputs and the desired outputs. Therefore, machines are expected to be able to adjust their internal structures to produce correct outputs and thus suitably constrain their input-output functions to approximate the relationships implicit in the examples or meet the expectations.
- It is possible that important relationships and correlations can be obtained from training. Machine learning methods can often be used to extract these relationships.
- In fact, certain characteristics of the working environment might not be completely known at the design time. Therefore, machine learning methods can be used for on-the-job improvement of existing machine designs.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans. Machines that learn this knowledge gradually might be able to capture more of it than humans would want to write down.
- As environments change over time, machines that can adapt to a changing environment would reduce the need for constant redesign.
- New knowledge about tasks is constantly being discovered. There is a constant stream of new events in the world. Continuing redesign of AI systems to conform to new knowledge is impractical, but machine learning methods might be able to track much of it.

According to the sources of training information, most of the existing learning algorithms for FNNs can be classified into Supervised Learning (SL), Reinforcement Learning (RL) and Unsupervised Learning (UL).

SL is a learning approach that adopts a supervisor/teacher, through which, the training system can adjust structure and parameters of FNNs according to a given

instructive training data set. A number of works have been done for generating FNNs by SL approaches [4, 7, 12–18]. Typically, these SL methods are based on input-output sets of training data. They aim to learn the input-output relationship upon minimizing errors between the teacher’s actions and the learner’s actions. However, the instructive training data required in SL are not always available especially when a human being has little knowledge about the system or the system is uncertain.

In those situations, UL and RL are preferred over SL as UL and RL are learning processes that do not need any supervisor to tell the learner what actions to take. Several researchers have developed paradigms of training FNNs by RL [3, 19–24] and UL [25–29]. However, systematic design of FNNs by non-SL approaches for both structure identification and parameter estimation is still an open issue.

RL plays an important role in adaptive control [30], especially in case that there are no explicit instructive data available in the environment where an learner must learn to perform optimal control actions through interacting with the environment. Previous works have proposed some paradigms of training consequent parameters (actions) by RL [21–24]; however, applying RL methodology in structure identification and generating premises of FNNs is still an open issue.

UL is usually adopted as a tool for input-space clustering and utilized in hybrid with SL or RL for generation of FNNs. In this thesis, we are looking for combination of UL with RL for generating FNNs without teaching signals.

In this thesis, we have strong interest and motivations in methodologies of machine learning with topics on but not limited to:

Non-instructive learning Machine learning approaches without supervised teaching data are to be developed. FNNs should be generated dynamically without SL by combining RL and UL methodologies.

Criteria of evaluating fuzzy rules How to evaluate significance and sensitivities of fuzzy rules according to reinforcement signals is to be studied in this thesis.

Structure identification and premises determination by RL Novel methodologies of generating structure and premise parts of FNNs in an RL manner are to be developed.

RL for continuous consequent parameters Discrete consequent parameters of FNNs are estimated by conventional RL approach. However, how to apply RL in generating continuous consequent parameters of FNNs is to be studied.

Hybrid approaches with both RL and UL RL and UL are the two non-SL learning methods. How to hybrid the two in both structure identification and parameter estimation are to be studied.

Online learning We are keen on learning algorithms that are capable of learning the dynamics of a system based on online data.

1.1.3 Motivation based on robot learning

The development of robotics has been changing human being's life day after day. Robots and robotics applications have become a popular topic in applying machine learning strategies. In order for employing and implementing robots in increasingly challenging and demanding tasks, it is necessary to embed learning capability to robots.

Usually, robots are expected to response suitable outputs/actions to the sensory inputs/information received from the external environment for accomplishing some given tasks. Control rules can either be determined by human experts or obtained through machine learning. It is more challenging and interesting that if robot can automatically perform the given tasks without much assistance from human operators. Therefore, embedding learning capability and empowering high autonomous ability to robots are studied in this thesis. In this thesis we look forward to applying learning algorithms on robotics applications and doing research with topics on but not limited to:

Self-learning capability Instead of subjective design of robotics control, an au-

onomous controller with self-learning capability is desired.

Evaluative training data Robotic systems can learn with or without instructive training data. However, it is more challenging and interesting that if a robot can learn a task without the instructive data but exploring the state space with an evaluative feedback system only.

Adaptive learning In order for a robot to perform task in a dynamically changing environment, the robot should be able to adjust the controlling rules and actions adaptively via the learning.

Continuous states and actions In many real-world tasks for robots, the sensory and action spaces are continuous. However, conventional RL methods are based on discrete states and actions. How to apply RL in continuous domain is to be studied in this thesis.

Path planning In this thesis, we are going to apply learning algorithms for path planning of a mobile robot. Wall-following and obstacle avoidance tasks are to be studied.

1.2 Major Contributions of the Thesis

In this thesis, a novel RL methodology for automatic generation of FNNs termed Dynamic Self-Generated Fuzzy Q-Learning (DSGFQL) is proposed.

Generation of FNNs can be divided into two levels, namely premise-generation and consequent-generation. At the premise-generation level, structure identification and estimation of premise parameters of FNNs are carried out. Consequent-generation deals with determination of the consequent parameters.

In the DSGFQL, both premise- and consequent-generations are achieved by RL approaches. Especially, the structure can be determined according to the reinforcement evaluation and the premise parameters are adjusted in an RL manner. Therefore, distinguished from those existing methods [22–24] which apply RL

methods only in consequent generations, the proposed DSGFQL applies RL in generating both structure and premise parameters of FNNs as well as the consequent parameters.

Besides evaluating system performance through reward functions (objective functions), local contributions of each fuzzy rules are evaluated via the shared reinforcement signals. By this means, fuzzy rules can be evaluated, adjusted and deleted accordingly.

Contrast to the "squared Temporal Difference (TD) error" criterion in the Dynamic Fuzzy Q-Learning (DFQL) of [23], the TD error criterion in the DSGFQL is directly related to the value function and is applied in an RL manner (reinforce the good and penalize the bad).

The DSGFQL algorithm offers a novel view of FNNs generation. RL methodologies are not only applied for selecting optimal actions (consequent parameters) but also applied in determining the number of rules, pruning and adjusting premise parameters. Similarly as reinforcing good actions and penalizing poor actions in convectional RL approaches, good *rules* are promoted while bad *rules* are demoted or eliminated in the DSGFQL method. Therefore, instead of only focusing on applying RL in training consequent parameters (consequent-generation), the RL is adopted at a higher level (premise-generation level).

The DSGFQL algorithm is applied for robotics navigation in the wall-following and obstacle avoidance tasks. Comparison studies with other Q-learning methods demonstrate the superiority of the proposed algorithm.

In order to extend the DSGFQL algorithm, a reward function scheme is proposed as the reward value function can be applied to all RL problems including those non-TD-based ones. Similar results are obtained via the reward scheme and the reward scheme is a more general approach for generating FNNs by RL.

Moreover, the DSGFQL approach is also enhanced by combining with an extended SOM approach thus an enhanced version of the DSGFQL named EDSGFQL is proposed. The SOM algorithm has been proved as a useful tool for providing real-

time and on-line generalization in RL problems in [31]. An extended SOM in which the M-distance or firing strength of the input data instead of the distance between the input data and discrete center points is proposed for selecting the winning neuron. The extended SOM algorithm is combined with the DSGFQL for input-space partitioning in the EDSGFQL approach. Comparison studies demonstrate that the EDSGFQL approach achieves more efficient FNN structures as better rewards are obtained while fewer fuzzy rules are generated.

At the same time, instead of discrete local actions selected from a fixed action set, continuous local actions are obtained via the integration of the DSGFQL and the continuous action Q-learning (DSGFQL-CA). By this means, continuous actions can be obtained locally as well as from a fuzzy inference. Comparison studies show that the learning becomes faster as continuous actions can also be adjusted within local actions. By this means, continuous consequent parameters of FNNs can be estimated while most of existing RL approaches, e.g. the Fuzzy Q-Learning (FQL) of [22] and DFQL of [23], can only deal with discrete consequent parameters of FNNs.

The main contributions of this thesis are on generations of FNNs, e.g. both structure identification and parameter estimation. In the DSGFQL approach, novel methodology of generating structure and determining premise parameters of FNNs in an RL manner is proposed. The DSGFQL-reward scheme provides a general scheme for all RL methods in generating FNNs as reward function is a basic element of RL. The enhanced version, EDSGFQL, improves the input-space partitioning and hence strengthens the premise parameter estimation. At the same time, the hybrid DSGFQL-CA offers a novel approach of fuzzy Q-learning as continuous local actions are obtained instead of discrete ones. The DSGFQL-CA learns faster than the original DSGFQL approaches and achieves better result than the original continuous action Q-learning of [32].

Therefore, the DSGFQL has been extended at both premise-generation level (DSGFQL-reward and EDSGFQL) and consequent-generation (DSGFQL-CA) level. The salient characteristics of the DSGFQL algorithm and its extensional approaches

are summarized as follows:

Autonomous learning The DSGFQL has a fast learning speed since structure and parameters formulation are done automatically and systematically without partitioning the input space *a priori*.

Dynamic generation Algorithms for dynamic self-generated FNNs are proposed. The DSGFQL algorithm provides an efficient learning methodology whereby not only the conclusion part of an FNN can be adjusted online, but also the structure and premise parameters of an FNN can be constructed simultaneously. Based on the criteria pertinent to some desired system performance, fuzzy rules can be created, adjusted and deleted automatically so as to improve system performances and maximize reward value functions when necessary.

Non-instructive learning Particularly, the DSGFQL algorithm is proposed as a novel RL approach for generating FNNs without instructive training data. The task is described with a reinforcement function, which can be a simple description of success and failure. Due to the low informative degree of the information source, the method represents a very flexible tool. Fuzzy rules are evaluated and adjusted according to RL signals in the DSGFQL system.

Evaluation of fuzzy rules via RL signals Besides applying reinforcement signal for evaluating system performance, contributions of each fuzzy rules are evaluated through the shared reinforcement signals. Therefore, the local performance or individual contributions of each fuzzy rule can be evaluated through the reinforcement sharing mechanism.

Structure identification by RL RL skills and philosophy are implemented to optimize the structure of FNNs with the aim of obtaining the best rewards or objective functions efficiently. The DSGFQL algorithm provides an efficient RL methodology whereby not only the conclusion part of an FNN can be adjusted by RL, but also the structure and premise parameters of an FNN can be constructed according to the reinforcement evaluation.

Pruning techniques Pruning techniques with RL are proposed. Redundant fuzzy rules can be removed and more efficient fuzzy structure can be achieved. Novel pruning techniques with RL have been proposed in this thesis.

General scheme for all RL problems The DSGFQL-reward scheme provides a general approach of determining the premise-generation for all RL methodologies.

Continuous domain For the input and output space, continuous states are handled and continuous actions are generated by fuzzy reasoning and the hybrid DSGFQL-CA. Fuzzy logic introduces generalization in the state space by means of using a vector of fuzzy variables to describe a fuzzy state. At the mean time, the hybrid DSGFQL with continuous action Q-learning can also generate and train continuous local actions for optimal solutions. Therefore, it learns faster than the original FQL and DFQL where the optimal continuous action can be only obtained by fuzzy interference.

Better premise allocation Furthermore, the extended SOM is proposed and incorporated for improving the input-space partitioning. Premise parameters are better adjusted by the extended SOM. With better allocation of fuzzy neurons, the original DSGFQL is enhanced and the number of fuzzy rules can be further reduced.

Incorporate domain knowledge Initial knowledge can be embedded with FNNs easily in the proposed learning structure. Initial domain knowledge is included as a bias in the FNN according to the confidence level of the knowledge. An NN structure for incorporating the bias components according to the confidence level of the initial knowledge is proposed. Therefore, rapid and safe learning can be achieved. These biases can eventually be overridden by more detailed and accurate learned knowledge.

Sharing mechanism for fast learning Bias increments the safety of the learning process and accelerates the learning process since it focuses on the search process of promising parts of the action space immediately. Therefore, a

sharing mechanism of initializing Q-values is proposed to share the previous training result and reduce the training time.

Because fuzzy rules can be generated automatically according to system performance, the DSGFQL is more flexible than the FQL with a fixed structure in [22]. At the same time, the DSGFQL approach is superior to the continuous-action Q-learning approach [32], as better exploration strategy implemented. Compared with the recent DFQL [23] and Clustering and Q-value based Genetic Algorithm learning schemes for Fuzzy system design (CQGAF) [24], the proposed DSGFQL approach can dynamically adjust and delete fuzzy rules at the premise level according to the reinforcement signals. It is more suitable for dynamic environments and it generates more efficient FNNs than the DFQL and CQGAF approaches. The proposed reward value scheme of the DSGFQL algorithm works as well as the TD error scheme and it offers a general RL scheme in generating preconditioning parts of FNNs. By updating the centers of MFs through the extended SOM approach, the original DSGFQL is enhanced. Therefore, more efficient FNNs are generated and fuzzy neurons are better allocated by adopting the extended SOM approach. Comparative studies show that the EDSGFQL approach is superior to the DSGFQL as the number of rules is significantly fewer with similar or even better performances achieved. On the other hand, the DSGFQL is enhanced by combining with the continuous action Q-learning. As better exploration strategy is adopted in the DSGFQL-CA, the system results are better than those of the original continuous action Q-learning. At the same time, as continuous actions are generated in local actions, the system improves faster than the original DSGFQL.

1.3 Outline of Thesis

This thesis is organized in eight chapters and each of which is devoted to a particular sub-issue. A summary of the content of each chapter is given here:

Chapter 1 presents motivations and contributions of the thesis and gives a brief outline of each chapter in the thesis.

Chapter 2 introduces the foundation of fuzzy system and NNs. Intrinsically, the FIS provides a structure, which can convert the linguistic rules based on expert knowledge into some automatic control actions. In order to circumvent the problem of lacking of learning capability, fuzzy logic is combined with NNs. Several design issues on generating FNNs are discussed in this chapter.

Chapter 3 gives an overview of the field of machine learning. Literature review on SL, UL and RL for generating FNNs is provided in this chapter and introductions to RL, TD learning and Q-learning are included in this chapter.

Chapter 4 presents the DSGFQL algorithm for automatically generating FNNs in an RL manner. The novel DSGFQL methodology can automatically create, delete and adjust fuzzy rules according to reinforcement of the entire system as well as the individual fuzzy rules. The salient characteristics of the DSGFQL approach are: 1) Capable of structure self-identification and automatic parameter estimation and modification; 2) FNNs can be quickly generated without SL; 3) Fuzzy rules can be evaluated, adjusted and deleted dynamically to reinforcement signals; 4) MFs of an FNN can be dynamically adjusted according to the reinforcement evaluation.

Chapter 5 carries out experiments and simulation studies with applications to a Khepera robot for the wall-following and obstacle avoidance tasks. Comparison studies of the DSGFQL approach with basic fuzzy controller, original fixed FQL, DFQL as well as the CQGAF approaches are carried out. Both static and dynamic training environments are adopted for comparative studies of the proposed approach. Simulation results demonstrate the superiority of the proposed DSGFQL method.

Chapter 6 studies two extensional approaches to extend the DSGFQL approach in determining premise parameters. Besides the TD error criterion in the original DSGFQL method, a reward function scheme of the DSGFQL algorithm (DSGFQL-reward) is proposed as a general approach for all RL problems. Moreover, an extended SOM is proposed to adjust the center po-

sitions of fuzzy neurons for better feature representation. As a result, the enhanced approach, EDSGFQL, is proposed based on the proposed extended SOM algorithm. Both methods are applied to the wall-following and obstacle avoidance tasks in this chapter.

Chapter 7 extends the DSGFQL in generating consequent parameters. The DSGFQL approach is combined with the continuous action Q-learning to generate continuous local actions. The hybrid system learns faster than the original DSGFQL and achieves better results than the original continuous action Q-learning.

Chapter 8 concludes this thesis and suggests several promising directions for future research based on the results presented in this thesis.

Chapter 2

Fuzzy Systems and Neural Networks

Fuzzy logic has been proposed as a great tool for representing imprecise knowledge which is common in many real-life systems and human linguistic language. On the other hand, Artificial Neural Networks or Neural Networks (NNs) are networks of highly interconnected neural computing elements that have the ability of responding to input stimuli and learning to adapt to the environment. Both fuzzy systems and NNs are dynamic and parallel processing systems that estimate input-output functions [33].

Fuzzy systems, which are treated in a symbolic qualitative manner, are appropriate if sufficient expert knowledge about the process is available. On the other hand, neural systems, which are treated in a numeric quantitative manner, are useful if sufficient process data are available or measurable. However, fuzzy systems do not possess capabilities for self-learning, while NNs do not provide a strong scheme for knowledge presentation. To circumvent these problems, the merits of both fuzzy and neural systems can be integrated in Fuzzy Neural Networks (FNNs) [34–37]. Therefore, the integration of fuzzy and neural systems leads to a symbiotic relationship in which fuzzy systems provide a powerful framework for expert knowledge representation, while NNs provide learning capabilities.

In this chapter, we begin by introducing the basic concept of Fuzzy Inference Systems (FISs) and NNs. After that, we introduce and discuss some issues concerning integrating and designing FNNs.

2.1 Fuzzy Inference Systems

2.1.1 Fuzzy logic and fuzzy systems

Fuzzy logic was first proposed by Zadeh [38] in 1965, with a view to reconcile conventional precise mathematical paradigms and human-like decision making [1, 2, 39–41]. A considerable body of literature has blossomed around the concept of fuzzy sets and systems in an incredibly wide range of areas, from mathematics and logics to traditional and advanced engineering methodologies. A good survey on fuzzy control systems can be found in [42].

Fuzzy sets and systems are knowledge-based or rule-based sets and systems. The key factor of a fuzzy system is a knowledge base consisting of the fuzzy If-Then rules, in which fuzzy logic uses logical operators to collate and integrate human knowledge. It facilitates representation of human being's expert knowledge through the utilization of fuzzy sets in a mathematical form. Fuzzy sets theory participates to the trend of storing and exploiting human knowledge in various domains where little objective and precise data are available [43]. The definition of a fuzzy set was given in Zadeh's famous paper [38]:

A fuzzy set is a class with a continuum of membership grades.

The degree of the Membership Function (MF) is usually between 0 and 1. Such a technique clearly provides a mathematical model for representing linguistic words, vagueness, imprecision and uncertainties. Fuzzy sets and systems adopt fuzzy MFs to represent the imprecise linguistic language in mathematical forms. Distinguished from conventional set theory in which an element either belongs to or does not belong to a given set, fuzzy set theory takes a loose view and describes

elements with degrees of membership of a particular set such that elements are not restricted to either "in" or "out" of a set but are allowed to be "some degree" in or out of the set. It is a more natural approach especially when a human describes information in a linguistic language. The degree is defined by the value of an MF.

While seemingly imprecise to a human being, fuzzy sets are mathematically precise in which they can be fully represented by exact numbers. They can therefore be seen as a method of combining human and machine knowledge representation together. Given such a natural method of representing information in a computer, information processing methods can be applied to it by the use of FISs. An FIS, which is also known as a fuzzy rule-based system, fuzzy model or simply fuzzy system, is a computation framework based on the concepts of fuzzy sets, fuzzy If-Then rules and fuzzy reasoning. The essential part of an FIS is a set of linguistic rules related by the dual concept of fuzzy implication and the compositional rule of inference.

Intrinsically, linguistic rules based on expert knowledge can be converted into automatic control actions via FIS. In literature, there have been many experiments shown that FISs yield results far more superior to those obtained by conventional approaches [1]. In particular, the methodology of FISs appears very useful when qualitatively information and expert knowledge are available even they may be inexact or uncertain. Thus, FISs are viewed as a step towards the approach between conventional precise mathematical paradigms and human-like decisions making [1–3, 39].

2.1.2 Fuzzy membership functions

A fuzzy set F in a referential U is characterized by an MF which associates with each element u in U as a real number in the interval $[0, 1]$, where 0 implies total absence of membership, 1 implies complete membership, and any value in between implies partial membership of the fuzzy set. A fuzzy set F is thus defined as a

mapping

$$F : U \rightarrow [0, 1], \quad (2.1)$$

and it is a kind of generation of the traditional characteristic function of a subset A .

$$A : U \rightarrow \{0, 1\}, \quad (2.2)$$

There are various possible types of fuzzy MFs which provide different meanings for the fuzzy values that they quantify and the most popular membership descriptions are the trapezoid, triangular and Gaussian MFs, which represent very easy ways to compute the degree of input variable membership.

A trapezoid MF is specified by four parameters a, b, c, d as follows:

$$\mu(x; a, b, c, d) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a < x < b \\ 1, & b \leq x \leq c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & d < x. \end{cases} \quad (2.3)$$

If $b = c$ in the trapezoid MF, it becomes a triangular MF. A triangular MF is specified by three parameters a, b, c as follows:

$$\mu(x; a, b, c) = \begin{cases} 0, & x \leq a \\ \frac{x-a}{b-a}, & a < x \leq b \\ \frac{c-x}{c-b}, & b < x \leq c \\ 0, & c < x. \end{cases} \quad (2.4)$$

The parameters a, b, c with $(a < b < c)$ determine the x coordinates of the three corners of the underlying triangular MFs. If isosceles triangular MF is adopted,

the MF can be characterized by two parameters c, σ as

$$\mu(x; c, \sigma) = \begin{cases} 1 - \frac{|x-c|}{\sigma}, & \text{if } |x-c| \leq \sigma \\ 0, & \text{otherwise} \end{cases} \quad (2.5)$$

where c represents the center of an MF and σ determines the width of an MF.

A Gaussian MF is also specified by two parameters, c, σ , but in a different shape:

$$\mu(x; c, \sigma) = \exp\left[-\frac{(x-c)^2}{\sigma^2}\right] \quad (2.6)$$

The shapes of the MFs are shown in Figure 2.1.

Figure 2.1: Fuzzy membership functions

Remarks: Due to the simple formulas and computational efficiency, both trapezoidal MFs and triangular MFs have been used extensively. However, since the trapezoidal and triangular MFs are composed of straight line segments, they are not smooth at the corner points specified by the parameters. At this point, Gaus-

sian MFs offer better smooth and nonlinear functions and they are quite popular for specifying fuzzy sets. Since smooth path planning is preferred in robotics navigation, Gaussian MFs are adopted in this thesis. Moreover, Gaussian functions are well known in statistics and probability and they also possess useful properties such as invariance under multiplication and Fourier transfer [3]. Therefore, we focus on Gaussian functions in this thesis. Similar generation and application of other types of MFs can be extended accordingly.

2.1.3 Fuzzy intersection and union

The intersection of two fuzzy sets A and B is specified in general by a T -norm operator, $T : [0, 1] \times [0, 1] \rightarrow [0, 1]$, which aggregates two MFs as follows:

$$\mu_{A \cap B}(x) = T(\mu_A(x), \mu_B(x)) = \mu_A(x) \tilde{*} \mu_B(x), \quad (2.7)$$

The T -norm operator [44] is a two-place function $T(., .)$ satisfying

$$\begin{aligned} T(0, 0) = 0, T(a, 1) = T(1, a) = a & \quad (\text{boundary}) \\ T(a, b) \leq T(c, d) \text{ if } a \leq c \text{ and } b \leq d & \quad (\text{monotonicity}) \\ T(a, b) = T(b, a) & \quad (\text{commutativity}) \\ T(a, T(b, c)) = T(T(a, b), c) & \quad (\text{associativity}) \end{aligned} \quad (2.8)$$

The most frequently used T -norm operators are:

$$\begin{aligned} \text{Minimum :} \quad T_{\min}(a, b) &= \min(a, b) = a \wedge b. \\ \text{Algebraic product :} \quad T_{ap}(a, b) &= ab. \\ \text{Bounded product :} \quad T_{bp}(a, b) &= 0 \vee (a + b - 1). \\ \text{Drastic product :} \quad T_{dp}(a, b) &= \begin{cases} a, & \text{if } b = 1 \\ b & \text{if } a = 1 \\ 0 & \text{if } a, b < 1 \end{cases} \end{aligned} \quad (2.9)$$

It is shown in [3] that

$$T_{dp}(a, b) \leq T_{bp}(a, b) \leq T_{ap}(a, b) \leq T_{min}(a, b) \quad (2.10)$$

Like fuzzy intersection, the fuzzy union operator is specified in general by an S -norm operator, $S : [0, 1] \times [0, 1] \rightarrow [0, 1]$. In symbols,

$$\mu_{A \cup B}(x) = S(\mu_A(x), \mu_B(x)) = \mu_A(x) \tilde{+} \mu_B(x), \quad (2.11)$$

The S -norm operator [44] is a two-place function $S(., .)$ satisfying

$$\begin{aligned} S(1, 1) &= 1, S(a, 0) = S(0, a) = a && \text{(boundary)} \\ S(a, b) &\leq S(c, d) \text{ if } a \leq c \text{ and } b \leq d && \text{(monotonicity)} \\ S(a, b) &= S(b, a) && \text{(commutativity)} \\ S(a, S(b, c)) &= S(S(a, b), c) && \text{(associativity)} \end{aligned} \quad (2.12)$$

The most frequently used S -norm operators are:

$$\begin{aligned} \text{Maximum :} \quad & S_{max}(a, b) = \max(a, b) = a \vee b. \\ \text{Algebraic sum :} \quad & S_{as}(a, b) = a + b - ab. \\ \text{Bounded sum :} \quad & S_{bs}(a, b) = 1 \wedge (a + b). \\ \text{Drastic sum :} \quad & S_{ds}(a, b) = \begin{cases} a, & \text{if } b = 0 \\ b & \text{if } a = 0 \\ 1 & \text{if } a, b > 0 \end{cases} \end{aligned} \quad (2.13)$$

It is shown in [3] that

$$S_{max}(a, b) \leq S_{as}(a, b) \leq S_{bs}(a, b) \leq S_{ds}(a, b) \quad (2.14)$$

For details of the subjects on the T-norm and S-norm, readers can refer to [3] and [45].

2.1.4 Fuzzy If-Then rules

FISs are rule-based systems which can be incorporated with human being's expert knowledge. In an FIS, each fuzzy rule defines a desired consequent action when a particular fuzzy state occurs. The fuzzy state usually is determined by a combination of fuzzy values. The rules are defined in an If-Then form as follows:

$$\text{If } \textit{premise states}, \quad \text{Then } \textit{consequent actions} \quad (2.15)$$

Usually, the inputs of the fuzzy system are associated with the premise states, and the outputs are associated with the consequent actions. The basic form of a linguistic rule is

$$R_j : \quad \text{If } X \text{ is } S_j, \quad \text{Then } Y \text{ is } W_j \quad (2.16)$$

where X and Y are input and output linguistic variables respectively, S_j and W_j are linguistic values or labels of fuzzy sets in the j th fuzzy rule characterized by appropriate MFs $\mu_{ij}(x_i)$, $i = 1, \dots, N$ and $\mu_{kj}(y_k)$, $k = 1, \dots, M$ respectively, and $R_j, j = 1, \dots, L$ represents the j th fuzzy rule. The rule in Eq.(2.16) can be written in a canonical form as follows:

$$R_j : \quad \text{If } x_1 \text{ is } S_{1j} \text{ and } \dots \text{ and } x_N \text{ is } S_{Nj}, \quad \text{Then } y \text{ is } W_j \quad (2.17)$$

It has been proved in [1] that the canonical fuzzy If-Then rules in the form of Eq. (2.17) include the following as special cases:

(1) "Partial rules":

$$\text{If } x_1 \text{ is } S_{1j} \text{ and } \dots \text{ and } x_P \text{ is } S_{Pj}, \quad \text{Then } y \text{ is } W_j$$

(2) "Or rules":

$$\begin{aligned} &\text{If } x_1 \text{ is } S_{1j} \text{ and } \dots \text{ and } x_P \text{ is } S_{Pj}, \quad \text{Or } x_{P+1} \text{ is } S_{(P+1)j} \text{ and } \dots \text{ and } x_N \text{ is } S_{Nj} \\ &\text{Then } y \text{ is } W_j \end{aligned}$$

- (3) "Single fuzzy statement":

$$y \text{ is } W_j$$

- (4) "Gradual rules", for example:

$$\textit{The smaller the } x, \textit{ the bigger the } y$$

- (5) Conventional production rules

If we assume that the output variables of a Multi-Input Multi-Output (MIMO) rule are independent, an MIMO FIS can be represented as a collection of Multi-Input Single-Output (MISO) FISs by decomposing the above rules into M sub-rules with $y_k, k = 1, \dots, M$ as the single consequent of the k th sub-rule [1, 2]. For notational simplicity, MISO FISs would be considered in the rest of the chapter.

Another form of fuzzy If-Then rules has fuzzy sets involved only in the premise part. This form of fuzzy If-Then rules can be categorized into two models, namely Simplified Model and Takagi-Sugeno-Kan Model.

- (1) Simplified Model (S-model): In S-model, a fuzzy singleton is used for the output [1], i.e.

$$R_j : \text{If } X \text{ is } S_j, \text{ Then } y \text{ is } C_j \quad (2.18)$$

where C_j is a fuzzy singleton in the j th rule.

- (2) Takagi-Sugeno-Kan Model (TSK-model): Takagi and Sugeno proposed the following fuzzy model in 1985 [46]:

$$R_j : \text{If } X \text{ is } S_j, \text{ Then } y \text{ is } f_j(x_1, \dots, x_N) \quad (2.19)$$

The premise of this rule is defined in the same way as that for the rule of the standard fuzzy system. However, instead of a linguistic term with an associated MF, a function of the inputs that does not have an associated MF is adopted in the consequents of the rules. Usually, $f_j(x_1, \dots, x_N)$ is a polynomial in the input variables, but it can be any function as long as it

can appropriately describe the output of the model within the fuzzy region specified by the antecedent of the rule. If no input variables are considered, the TSK-model is exactly the same as the S-model. Therefore, the S-model can be considered as a special case of the TSK-model. Experiments show that the TSK-model has advantages like computational efficiency, compatibility with linear, adaptive and optimization techniques and continuity of the output surface.

Both types of fuzzy If-Then rules have been extensively used in both modelling and control. Human expert knowledge can be easily embedded to the control and modelling systems through the fuzzy If-Then rules in linguistic forms. On the other hand, knowledge can be extracted from FISs generated and trained via computers and machines.

2.1.5 Mechanism of Fuzzy Inference Systems (FISs)

An FIS in Figure 2.2 can be defined as a nonlinear mapping of the input data vector into a scalar output using fuzzy rules [47]. The mapping process involves membership functions for both inputs and outputs, fuzzy operators, If-Then rules, aggregation of output sets and defuzzification.

Figure 2.2: Fuzzy inference systems

There are four main components in an FIS: the fuzzification interface (fuzzifier), inference engine, fuzzy rules, and defuzzification (defuzzifier). The fuzzification interface (fuzzifier) defines a mapping from a real-valued space to a fuzzy space and the defuzzification (defuzzifier) defines a mapping from a fuzzy space to a real-valued space [42]. Fuzzy rules can be provided by experts as well as extracted from numeric data.

Once the fuzzy rules have been established, the FIS can be viewed as a system that maps an input vector to an output vector. This mapping process and fuzzy reasoning is performed in four steps:

Fuzzification Converts crisp input variables into fuzzy sets through the MFs to determine the degree of truth for each rule premise.

Inference The inference engine transfers mapping from input fuzzy sets into output fuzzy sets. The true value for the premise of each rule is computed and applied to the conclusion part of each rule. This results in one fuzzy subset to be assigned to each output variable for each rule. Usually, only MIN or PRODUCT is used in inference rules. Under MIN inference rule, the output MF is clipped off at a height corresponding to the rule premise's computed degree of truth (fuzzy logic AND). Under PRODUCT inference rule, the output MF is scaled by the rule premise's computed degree of truth.

Composition If more than one rules are fired at the same time, all fuzzy subsets assigned to each output variable are combined together to form a single fuzzy subset for each output variable. Again, usually MAX or SUM is used. In MAX composition, the combined output fuzzy subset is constructed by taking the point wise maximum over all fuzzy subsets assigned to each variable by the inference rule (fuzzy logic OR). In SUM composition, the combined output fuzzy subset is constructed by taking point wise sum over all fuzzy subsets assigned to the output variable by the inference rule.

Defuzzification Whereby the fuzzy output set is converted to a crisp number. There are lots of defuzzification methods. Two of the most common tech-

niques are the MAXIMUM and CENTROID methods. In the MAXIMUM method, one of the variable values at which the fuzzy subset has its maximum true value is chosen as the crisp value for the output variable. The most popular defuzzification method is the CENTROID, in which the crisp value of the output variable is computed by finding the variable value of the center of gravity of the MF for the fuzzy value [47].

Remarks: As pointed out in [47], unlike rules in conventional expert systems, a fuzzy rule localizes a region of space along the function surface instead of isolating a point on the surface. For a given input, more than one rule may fire. Also in an FIS, multiple regions are combined in the output space to produce a composite region.

2.1.6 Three types of Fuzzy Inference Systems (FISs)

Several types of fuzzy reasoning have been proposed in the literature [40, 48–53]. Most FISs can be classified into three types of FISs, depending on the type of fuzzy reasoning [3, 7].

Tsukamoto-type FIS The overall output y is the weighted average of the j th rule's crisp output, ω_j , by the firing strength of the j th rule, f_j . Here, f_j is calculated by the T-norm operations, e.g.

- Intersection:

$$f_j = \min[\mu_{1j}(x_1), \mu_{2j}(x_2), \dots, \mu_{Nj}(x_N)] \quad (2.20)$$

- Algebraic product:

$$f_j = \prod_{i=1}^N \mu_{ij}(x_i) \quad (2.21)$$

It should be stressed that the output MFs must be monotonically non-decreasing (see [48] for details). This type of fuzzy inference systems with

two-rule two-input is illustrated in Figure. 2.3.

Figure 2.3: Tsukamoto-type fuzzy inference system

Remarks: The Tsukamoto FIS scheme aggregates the output of each fuzzy rule by the method of weighted average and thus avoid time-consuming process of defuzzification. However, the Tsukamoto is not often used as it is not transparent as either the Mamdani or TSK type of FIS [3].

Mamdani-type FIS The overall fuzzy output MF $o(\omega)$ is derived by applying "maximum" operation to the qualified fuzzy output MF $o_j(\omega), j = 1, 2, \dots, L$ as follows:

$$o(y) = o(\omega_1) \vee o(\omega_2), \vee, \dots, \vee o(\omega_L) \quad (2.22)$$

where,

$$o(\omega_j) = f_j \wedge \mu(\omega_j) \quad (2.23)$$

Here, f_j is defined the same as that in the Tsukamoto type and $\mu(\omega_j)$ is the MF of consequent output of the j th rule.

Various schemes have been proposed to choose the final crisp output y based on the overall fuzzy output MF $o(\omega)$; some of them are Centroid of Area (COA), Mean of Maximum (MOM), maximum criterion, etc [3, 40]. It is shown in [54, 55] that the COA strategy yields superior results. For the COA method, the crisp output y is shown as follows [40]:

$$y = \frac{\int o(\omega)\omega d\omega}{\int o(\omega)d\omega} = \frac{\sum_{j=1}^L o(\omega_j)\omega_j}{\sum_{j=1}^L o(\omega_j)} \quad (2.24)$$

The type of Mamdani FIS is illustrated in Figure. 2.4 which depicts a two-rule two-input FIS.

Figure 2.4: Mamdani-type fuzzy inference system

Remarks: Mamdani-type of FIS is intuitive and has widespread acceptance. It is well suited to human input. However, the computation of Mamdani type FIS is quite time-consuming as the calculation involved in the defuzzification operation is heavy. Furthermore, most of the studies on Mamdani type of FISs are based on experiment results and are not easily subject to rigorous mathematical analysis [3].

TSK-type FIS TSK-model fuzzy IF-THEN rules, which are described in Section 2.1.4, can be used to implement FISs [46]. The output of each rule is a polynomial in the input variables, and the final crisp output, y is the weighted

average of each rule's output, $\omega_j = g(x_1, \dots, x_N)$, i.e.

$$y = \frac{\sum_{j=1}^L f_j \omega_j}{\sum_{j=1}^L f_j} \quad (2.25)$$

where the firing strength f_j is calculated according to Eq. (2.21) and ω_j is usually computed as a linear combination of the inputs as follows [46]:

$$\omega_j = a_{0j} + a_{1j}x_1 + \dots + a_{Nj}x_N \quad (2.26)$$

One-way to view the TSK-model fuzzy system is that it is a nonlinear interpolator between the mappings that are defined by the linear functions in the consequents of the rules. When ω_j is a constant, we have a zero-order TSK fuzzy model. The output of a zero-order TSK model is a smooth function of its input variables as long as the neighboring MFs in the antecedent have enough overlaps. Since each rule has a crisp output, the overall output is obtained via weighted average, thus avoiding the time-consuming process of defuzzification required in other fuzzy models. A two-input two-rule FIS is depicted in Figure. 2.5.

Figure 2.5: TSK-type fuzzy inference system

Remarks: A TSK-type FIS is extremely well suited to the task of smoothly interpolating the linear gains that would be applied across the input space and it

is a natural and efficient gain scheduler. Similarly, a TSK-type FIS is suited for modelling nonlinear systems by interpolating between multiple linear models [56]. Compared with the Mamadani type of FIS scheme, the TSK FIS is computationally efficient and it is well suited to mathematical analysis. In the mean time, the TSK-type FIS works well with optimization and adaptive techniques as well as linear techniques. The TSK-type FIS also has guaranteed continuity of the output surface [3]. Therefore, the TSK inference scheme is studied in this thesis.

2.2 Neural Networks and Fuzzy Neural Networks

As described in the previous section, fuzzy logic is closer in spirit to human thinking and natural language than conventional digital logic. However, one of the shortcomings of fuzzy logic is the lack of learning capability. To circumvent this problem, fuzzy logic is usually hybrid with NNs which were first introduced by McCulloch and Pitts in the work of [5].

NNs are simplified models of the central nervous system. NNs are networks of highly interconnected neural computing elements that have the ability of responding to input stimuli and the capability of learning to adapt to the environment. NNs provide strong learning capability, parallel implementation and generalization ability. NN is an efficient method to approximate any nonlinear mapping [57]. NNs have been shown to be effective as computational processors for various tasks including pattern recognition, associative recall, classification, data compression, modelling and forecasting, combinatorial problem solving, adaptive control, multi-sensor data fusion and noise filtering [58]. However, the main drawback of NNs is the lack of interpretability. Therefore, the FNN, a hybrid fuzzy logic and NN system, has attracted great attention since Lee and Lee's papers in the 1970s [34, 35]. The technology of FNNs combines the profound learning capability of NNs with the mechanisms of explicit and easily interpretable knowledge presentation provided by fuzzy logic. In essence, FNNs are able to represent meaningful real-world concepts in comprehensive knowledge bases. An FNN is able to use its knowledge by fuzzy

logic to infer the relevance or degree of "truth" or membership of these concepts in relation to the current state of its environment. With the adaptive nature of NNs, this knowledge can be gained autonomously through learning.

The transparent logic-based architectures realized by these computationally intelligent entities facilitate communication between themselves and humans, where their learned structures may be modified or augmented by the user if desired. The value of knowledge interpretability becomes further evident in many applications, such as real-time process control, where desired behavior can be achieved via potentially discovering ways within the models structure to optimize its operation; another example is in robotics, e.g. learning and generalizing for navigation in unpredictable environments, where its learned knowledge could be checked by the designer to verify appropriate behavior. Some surveys on FNNs can be found in [33,42,59–61]. With the development of technology and industry, various hardware technologies have been implemented to represent FNNs [62–73].

2.3 Combination of Fuzzy and Neural Systems

A typical type of NNs is the Radial Basis Function (RBF) NNs [74,75], which is depicted in Figure. 2.6. The NNs can be considered as a mapping $f : R^N \rightarrow R^M$

Figure 2.6: RBF neural networks

according to:

$$y(X) = w_0 + \sum_{i=1}^L w_i R_i(||X - C_i||) \quad (2.27)$$

where $X \in R^N$ is the input, $R_i(\cdot)$ is a basis function, $||\cdot||$ indicates the Euclidean norm on the input space, w_i , $0 \leq i \leq L$, are the weights or strength, $C_i \in R^N$ ($1 \leq i \leq L$), are the prototypes of the input vectors, which are also known as the RBF centers, and L is number of RBF units, which also indicates the number of rules in an FNN structure. Although single output is considered here, extension to multiple outputs is straightforward.

The RBF NNs have achieved great success in function approximation and other applications [14, 17, 71, 74, 76, 77]. Theoretical investigation and practical results suggest that choice of the nonlinearity $R(\cdot)$ is not crucial to the performance of the RBF NNs [78–80]. Typical selections for $R(\cdot)$ are given in [74, 78] such as Thin-plate-spline function, multi-quadric function, inverse multi-quadric function and Gaussian function. In this thesis, Gaussian function is adopted for $R(\cdot)$ as it is found to be suitable not only in generalizing global learned mapping, but also in refining local features without much alteration to the already learned mapping so that fast learning speed can be achieved [75]. A typical Gaussian function is given as follows:

$$R(x) = \exp\left(-\frac{x^2}{\sigma^2}\right) \quad (2.28)$$

where σ is a real constant.

As several researchers have studied the functional equivalence between FISs and NNs, both fuzzy systems and NNs can be integrated easily.

It is pointed out in [81] that the functional equivalence between an RBF NN and an FIS can be established if the following conditions are true:

- The number of receptive field units is equal to the number of If-Then rules.
- The output of each fuzzy If-Then rule is composed of a constant.
- MFs within each rule are chosen as Gaussian functions with the same variance.
- The T-norm operator used to compute each rule's firing strength is multiplication.
- Both the RBF NNs and the FIS under consideration use the same method, i.e. either weighted average or weighted sum to derive their overall outputs.

Therefore, a simple model of TSK scheme FIS can be embedded in an RBF NN if Gaussian functions are chosen as the MFs. The RBF NNs can be viewed as a mechanism of representing the rule-based fuzzy knowledge by using its localized network structure and performing the associated fuzzy reasoning using feed forward computational algorithms. From the point of view of knowledge representation, the RBF network is essentially a network representation of If-Then rules, in which each hidden unit represents a rule and the basis function is equivalent to the MF in a fuzzy system [82]. Therefore, integration of FIS with NNs is realistic and available.

It is possible to tune FIS by using FNNs [83] and extract fuzzy rules from FNNs by training from the given numerical examples [8–10]. FNNs have been intended to capture capabilities of both types of systems and overcome drawbacks of each system [3, 7, 8, 10, 59, 60, 80, 83–95].

2.4 Design Issues in Fuzzy Neural Networks

The conventional approaches of designing an FNN are subjective approaches which are adopted to express domain knowledge of experts. However, it is either impossible or quite hard to obtain expert knowledge for systems which are in uncertain or dynamic or changing environments. Therefore, In order to acquire fuzzy rules,

much work has been done in [1, 8, 10, 39, 48–53, 85, 92, 96–104]. However, how to design FNNs in a systematic way and solve the following problems are still open research issues:

- There are a great number of SL methodologies which require sufficient teaching data for the learning process proposed for generating FISs. However, automatic generation of FISs without instructive training data is an open issue and a desired research area. Applying non-SL methods, especially RL methods, for generating both the structure and parameters of FNNs is a key concern in this thesis.
- Although several RL approaches for automatic adjustment of consequent parameters are derived in [22–24, 105], premises determination, e.g. structure identification and estimation of premise parameters via RL approaches are still open issues.
- Problems with high dimensionality are often encountered in implementation of fuzzy systems because the number of rules increases rapidly for multiple input systems [85, 96–98, 104], which is so called the *curse of dimensionality* in [84]. In order to circumvent this problem, several strategies such as tree-structured algorithms [85, 99, 100, 102], hierarchical fuzzy system structure [96, 104] and so on, are adopted. Efficient algorithms which relieve this problem and do not increase the complexity of FNNs are highly desired.
- In order to save computational cost and avoid the curse of dimensionality, pruning technology is required for removing redundant rules. Although there are several pruning methods in literature [4, 14, 79, 106, 107], most of them are based on SL. Pruning technology based on RL is a research issue in this thesis.

Chapter 3

Machine Learning

3.1 Overview of Machine Learning

Even since computers were invented, people have wondered whether they might be able to be made to learn. The ability of learning is one of the central feature of intelligence, which makes it an important concern for both cognitive psychology and Artificial Intelligence (AI).

Learning has been concerned with developing general mechanisms for cognition, perception and action [108]. Machine learning draws on ideas from a diverse set of disciplines, including AI, probability and statistics, computational complexity, information theory, psychology and neurobiology, control theory and philosophy [109].

According to the formal definition in [108]:

Learning is the improvement of performance in some environments through the acquisition of knowledge resulting from experiences in those environments.

A framework for machine learning is shown in Figure 3.1 as proposed in [108], where the dashed line indicates an optional link.

Designing a machine learning approach involves a number of design choices, including the type of training experience, the target performance function to be learned,

Figure 3.1: The interactions among learning, performance, knowledge and the environment.

a representation of this target function and an algorithm for learning the target function from training.

There are quite broad topics covered in machine learning, including fuzzy logic, NNs, Evolutionary Algorithms (EAs), probability and statistics, decision trees, computational learning, UL, RL, etc. Machine learning can also be applied in various applications such as pattern recognition, data mining, gaming, bio-science, communication, control and robotics applications. It is impossible to cover all the topics and all the applications in a thesis. The scope of this thesis is to study and develop machine learning methodology for automatic generation of FNNs with applications to robotic systems.

3.2 Supervised Learning

SL is a machine learning technique for creating a function from training data with pairs of input objects (typically vectors) and the desired outputs. The task of the supervised learner is to predict the value of the function for any valid input object after having seen a number of training examples (i.e. pairs of inputs and the desired outputs). Towards this end, the learner has to generalize from the presented data to unseen situations in a "reasonable" way. The key characteristic of SL is the existence of a "teacher", the training input-output data. The learning purpose of SL is to minimize the system error between the predicated output from the system and the actual output.

Typical SL paradigms of FNNs are to build standard NNs which approximate a fuzzy algorithm or a process of fuzzy inference [7, 8, 10, 77, 80, 83, 84, 93–95, 99, 100, 103]. The FNN, which is also known as neural-networks-driven fuzzy reasoning in [83], can readily solve the lack of systematic design for MFs and the lack of adaptability for possible changes in the reasoning environment. A self-organizing learning scheme for structure learning was adopted in [80] and [84] and a hierarchical self-organizing paradigm was developed in [77]. The premise structure of an FNN, is determined by clustering the input space in [93] and identified by input-output pairs in [95]. A tree structure [85, 100, 103] and an iterative construction algorithm called Adaptive Spline Modelling of Observation Data (AS-MOD) [99] are developed for high-dimension partitioning and online-learning. In order to solve the slowness and trapping in local minima of Back Propagation (BP) algorithm [79, 97], several paradigms with fast learning speed have been proposed, e.g., Resource Allocating Networks (RAN) [12] and its improved versions, such as the RANEKF [13] and the Minimal RAN (M-RAN) [14]. Based on those aforementioned methods, a hierarchical on-line self-organizing learning algorithm termed Dynamic Fuzzy Neural Networks (DFNN) was developed in a series of papers [15–18] and was summarized in [4].

If there are sufficient instructive training data, e.g. input-output data patterns, SL is a good approach for determining the input-output relationship and it is successful in function approximation as well as control applications. However, for some cases, these input-output training data are not available or too costly to obtain. Therefore, non-SL methodology is desired especially for those unknown and dynamic changing systems and environments.

3.3 Unsupervised Learning

UL is a method of machine learning where a model is fit to observations by typically treating input objects as a set of random variables and building a joint density model then. It is distinguished from SL by the fact that there is no *a priori*

output required. UL is usually used for data compression and clustering [25] as well as similarity detection [3].

UL offers the possibility of exploring the structure of data without guidance in the form of class information and can often reveal features not previously expected or known about [110]. A good survey for UL in neural computing has been included in [29] and several available good text-books covering major approaches are in [25–28].

The goal of UL is finding a new compressed representation for the observations. UL can be interpreted as coding of the data and categorized into two types [111]:

- extensions of linear transform coding methods of statistics;
- learning vector coding methods that are based on competitive learning.

The first class of UL methods is motivated by standard statistical methods like Principle Component Analysis (PCA) [27, 112–115] or Factor Analysis (FA) [116–118], which give a reduced subset of linear combinations of the original input variables. Independent Component Analysis (ICA) and Blind Source Separation (BSS) [28, 76] are more recent techniques in this category.

The second class of methods is close to clustering. UL methods such as Boltzmann machines [119] are computationally expensive, and iterative clustering methods such as Kohonen maps [120] and K-means clustering models [121] offer the same modelling power with greatly reduced training time. In the SOM introduced by Kohonen [120], there is an extra feature compared to mere clustering: the neurons are arranged to a one-, two- or multidimensional lattice, such that each neuron has a set of *neighbors*. The goal of SOM learning is not only to find the most representative code vectors for the input training set in the sense of minimum distance, but at the same time to form a topological mapping from the input space to the grid of neurons. This idea originally stems from the modelling of the topographic maps on the sensory cortical areas of the brain. A related early work in neural modelling is [122]. Several choices for the initial values and for the neighborhood

are available. These choices as well as the convergence and the mathematical properties of this algorithm have been considered in [26, 123–125]. For SOM learning, topology preservation and its relation to a cost function can be found in [126–128]. A more efficient learning rule for the SOM is the batch algorithm of [26]. The two-dimensional map is also a powerful tool for data visualization e.g., a color code can be used in which each unit has its own characteristic color. There is a software implementation of the SOM in the public domain, with various graphical tools for map presentations as well as with preprocessing methods in [129]. A database of well over five thousand applications of SOM is given by [130].

UL approaches are good methods for clustering and partitioning the input-space. However, it is lack of feedback information from the system output. On the other hand, RL helps improving the system performance by taking feedback from evaluation. Therefore, in this thesis, we will combine UL with RL approaches together in generating FNNs.

3.4 Reinforcement Learning

Distinguished from SL, RL is a learning process without explicit teacher for any correct instructions. RL method is also different from other UL approaches as it learns from an evaluative feedback of the system. RL has been accepted as a fundamental paradigm for machine learning with particular emphasis on computational aspects of learning [3].

3.4.1 Basic elements of reinforcement learning system

The RL problem is meant to be a straight forward framing of the problem of learning from interaction to achieve a goal. The learner is named as agent which interacts with the environment in [30]. These interactions are continual: the agent selects actions to the environment and the environment responses to these actions and presents new states to the agent.

Generally speaking, there are three basic elements of RL: a *policy*, a *reward function* and a *value function*.

A *policy* is how an agent takes actions by mapping from perceived states of the environment. The policy guides or controls the agent's learning behavior. The policy is the core of a RL problem as it alone is sufficient to determine behavior. A good policy should result in good achievement in obtaining the goals of the task and the learning behavior is usually evaluated via reward function and value function. Learning objective of RL is to obtain a good policy which can achieve maximum reward values (maximize objective function).

A *reward function* defines the goal and evaluation criteria in an RL problem. It maps each perceived state (or state-action pairs) of the environment to the intrinsic desirability of that state by a numeric value, a *reward*. In RL, the reward evaluation is defined by the training tasks. The reward evaluation can be defined by scaled values as "-1" for failure, "+1" for success and "0" for neutral. Moreover, continuous functions and fuzzy evaluation can also be adopted for the reward function as well [131, 132].

Whereas a reward function indicates the desirability of states and quality of policy (or actions) in an immediate sense, a *value function* specifies the quality in the long run. Roughly speaking, the *value* of a state is the total (or discounted) amount of reward an agent can expect to accumulate over the future. Therefore, reward function evaluates the immediate, intrinsic desirability of states while value function indicates the long-term desirability of states after taking into account the states that are likely to follow and the rewards available in future.

Without rewards, there could not be value functions and the purpose of estimating values is to achieve more reward (better performance). At the same time, value function can guide and help to learner to judge and adjust the control/action policy.

A diagram summarizing the agent-environment interaction is shown in Figure 3.2.

Figure 3.2: Agent-environment interaction in reinforcement learning

RL is a trial-and-error learning scheme whereby a computational agent learns to perform an appropriate action by receiving evaluative feedback through interactions with the environment. This learning method emulates animal's learning behaviors and it also matches the common-sense ideas of [3, 30, 133] which state that:

If an action is followed by a satisfactory state of affairs or an improvement in the state of affairs, the tendency to produce that action is strengthened or reinforced (rewarded). Otherwise, that tendency is weakened or inhibited (penalized).

We adopt this philosophy for designing self-learning systems throughout this thesis.

3.4.2 Reinforcement learning versus supervised learning

In RL, correctness is a relative property of actions determined only by trying them and comparing their rewards [30]. The key feature of RL different from SL is that RL is a learning by selection and evaluation from feedback while SL is a learning by instruction. In RL, the feedback from training environment only offers the evaluation of the current selection/states and there is no instruction to tell which action should be taken. On contrast, the "feedback" in SL directly indicates what actions should have been and typically independent of actions selected [30]. In this sense, RL is different from SL as there is no "instructive" information given but only evaluative feedback.

On the other hand, both SL and RL systems are learning according to both the

inputs and output responses. If the goal of the learning is just to minimize the differences between the system outputs and desired outputs, the system error in SL can be regarded as a penalty function in RL. By this means, the instructive training data in SL offer evaluation criteria as the reward functions in RL. In this sense, SL can be regarded equivalent to RL by

$$reward = K - \zeta \|e_i\| \quad (3.1)$$

where ζ is the scaling factor and K is a positive value which is set according to the desired accuracy.

Therefore, all SL methods can be regarded as special cases of RL, in which the reward evaluations are related to the system error between the predicate values and the actual outputs. RL can be considered as a more general form of machine learning systems.

Once SL is not applicable, due to the lack of instructive training data or the knowledge of the system, RL can still be adopted. There are four basic representative architectures for RL, delineated by Sutton, in [134]:

- Policy-only
- Reinforcement comparison
- Adaptive heuristic critic
- Q-learning

The last two, e.g. Adaptive heuristic critic learning and Q-learning, are prevalent approaches to RL and both of them are Temporal Difference (TD) learning.

3.4.3 Temporal-difference learning and Q-learning

TD learning is a central and novel idea of RL [30]. TD method is a combination of Monte Carlo approach [135–137] and Dynamic Programming (DP) [138, 139]. TD

methods are superior to Monte Carlo methods as they do not need to wait for a final outcome but wait only until the next time step. Obviously, TD methods can be naturally implemented online and in a fully incremental fashion. At the same time, TD methods are also superior to DP approaches as they can learn directly from raw experience without any model of environment dynamics which are required in DP approaches. Roughly speaking, TD methods combine the sampling of Monte Carlo methods with the bootstrapping of DP approaches.

As all RL methods, the objective of TD learning is to maximize the sum of rewards in case there is a terminal state:

$$R_t = r_{t+1} + r_{t+2} + \dots + r_T \quad (3.2)$$

where T is the final time step.

On the other hand, if the training task is a continuing task and there is no terminating state, RL is to maximize the expected discounted return:

$$R_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \dots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (3.3)$$

where γ is the discount rate which is in the range of $[0, 1]$.

Therefore, a value function $V(s)$ can be defined as:

$$V(s) = E_{\pi}\{R_t | s_t = s\} = E_{\pi}\left\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s\right\} = E_{\pi}\{r_{t+1} + \gamma V(s_{t+1}) | s_t = s\} \quad (3.4)$$

where $E_{\pi}\{\}$ is the expected value given that the agent follows policy π .

A procedural form of TD method is shown below as that in [30].

- (1) Initialize $V(s)$ arbitrarily, π to the policy to be evaluated
- (2) Repeat (for each episode):
 - $a \leftarrow$ action given by π for s
 - Take action a ; observe reward, r , and next state, s'
 - $V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)]$
 - $s \leftarrow s'$
- (3) until s is terminal

where s is the state of the environment, π is the policy for determining actions, r is the reward value, $V(s)$ is the value function of the state s and α is the learning rate.

The two most popular TD learning approaches are the adaptive heuristic critic approach and Q-learning. The adaptive heuristic critic approach, which is also named *actor-critic* model [140], typically includes two principal components: the *critic* (evaluation/prediction) module and the *action* (control) module. The *critic* generates an estimate of the value function from state vectors and external reinforcement generated by the environment. That is, the *critic* plays an important role in predicting the evaluation function. The *action* module attempts to learn optimal control or decision-making skills. On the other hand, Q-learning, proposed by Watkins [141], is one of the most important breakthroughs in RL. Q-learning is a simple way of dealing with incomplete-information Markovian-action problems based on the action-value function Q that maps state-action pairs to expected returns. The learner tries an action at a particular state and evaluates its consequence in terms of the immediate reward or penalty it receives and its estimate of the state value resulting from the taken action.

Actor-critic learning architecture requires two fundamental memory buffers: one for the evaluation function and one for the policy. On the other hand, Q-learning maintains only one: a pair of state and action. Compared with actor-critic learning, Q learning is an off-policy TD control algorithm, where optimal action-value

function Q^* is directly approximated by the learned action-value function, Q , independent of the policy being followed. According to TD learning, the $Q(s, a)$ function can be updated by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]; \quad s \leftarrow s' \quad (3.5)$$

By this means, it dramatically simplifies the analysis of the algorithm and enables early convergence proofs [141].

A procedural form of the Q-learning is shown below as that in [30].

- (1) Initialize $Q(s, a)$ arbitrarily,
- (2) Repeat (for each episode):
 - Initialize s
 - Repeat (for each step of episode)
 - Choose a from s using policy derived from Q
 - Take action a ; observe reward, r , and next state, s'
 - $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
 - $s \leftarrow s'$
 - until s is terminal

3.4.4 Reinforcement learning with fuzzy systems

The basic idea behind fuzzy RL is to apply fuzzy partitioning to the continuous state-space and to introduce linguistic interpretation. Such averaging over neighboring partitioned subspaces can create generalization abilities [3]. In the literature, several learning methods are based on the idea of actor-critic model. Berenji and Khedkar proposed the Generalized Approximate Reasoning for Intelligent Control (GARIC) model in [19], which has three components: the action selection network, the action evaluation network and the stochastic action

modifier. The action selection network is expressed in a neuro-fuzzy framework. Lin and Lee's Reinforcement Neural-Network-based Fuzzy Logic Control System (RNN-FLCS) [142] consists of a fuzzy controller and a fuzzy predictor. The whole RNN-FLCS is expressed in a neuro-fuzzy framework, where both critic and action modules share the antecedent parts of the fuzzy rules. In addition to parameter learning, it can perform structure learning and find an appropriate number of fuzzy rules. Lin and Jou [143] and Chiang et al. [20] proposed genetic RL algorithms for designing fuzzy systems by exploiting global optimization capability of Genetic Algorithms (GAs) in order to overcome the local minima problem in network learning due to the use of the gradient descent learning method. Bruske et al. [144] and Rak et al. [145] employed the actor-critic model to learn fuzzy controllers for autonomous robots. Jouffe's Fuzzy Actor-Critic Learning deals with the conclusion part of fuzzy rules [22]. Kandadai and Tien proposed a fuzzy-neural architecture that is capable of automatically generating a fuzzy system for use in hierarchical knowledge-based controllers [146].

On the other hand, other approaches allow efficient learning of fuzzy systems based on the Q-learning since Q-learning is the most popular RL method that directly calculates the optimal action policy without an intermediate cost evaluation step and without the use of a model. FNNs have strong generalization abilities of dealing with continuous inputs and outputs. Moreover, the fuzzy Q-learning method is considered to be a more compact version of the fuzzy actor-critic method. Glorennec and Jouffe considered a collection of fuzzy rules as an agent that produces continuous-valued actions in [147, 148]. This approach termed Fuzzy Q-Learning (FQL) produces an action by some rules triggering on the same state-space and cooperating collectively. Kim et al. proposed a self-organizing FNN by Q-learning [149] which is the same as that of [22]. The idea of FQL has also been implemented for a vision guided behavior acquisition in [150].

Similar rule structure and adaptive rewards are used in the simulation of object chase agents [151]. Horiuchi et al. considered a similar algorithm, termed Fuzzy Interpolation-Based Q-Learning, and further proposed an extended roulette selec-

tion method so that continuous-valued actions can be selected stochastically based on the distribution of Q-values [152]. Berenji [21] proposed another version of Q-learning dealing with fuzzy constraints. In this case, we do not have fuzzy rules, but "fuzzy constraints" among the actions that can be done in a given state. These works, however, only adjust the parameters of fuzzy systems online. Structure identification, such as partitioning the input and output space and determination of the number of fuzzy rules are still carried out off-line and it is time consuming. In [23], the number of fuzzy rules is decided through the ε -completeness and the squared TD error criterion. An "aligned clustering" which is similar to the ε -completeness criterion has been adopted in an online Clustering and Q-value based Genetic Algorithm learning schemes for Fuzzy system design (CQGAF) to generate a fuzzy structure in [24]. However, the TD error criterion in [23] is not applicable for RL methods which are not based on TD, and the threshold values of this TD error criterion are hard to define. The two methods only consider the input space clustering for deciding the number of rules and there are no pruning techniques for removing redundant rules.

Besides adopting UL in determining premises of FNNs as [24], RL paradigms are desired in generating premise parts of FNNs. Instead of only considering the input space clustering, system evaluative feedback should be concerned in structure identification and estimation of premise parameters. Therefore, RL algorithms for automatic and dynamic determination of both structure and parameters of FNNs are developed in this thesis and presented in following chapters.

Chapter 4

Dynamic Self-Generated Fuzzy Neural Networks

In this chapter, generation of a typical five-layer FNN is studied. In the literature, several research works have been done in generating this FNN or similar FNNs, e.g. [7, 18, 22–24].

In stead of SL in [7, 18], a novel RL approach termed Dynamic Self-Generated Fuzzy Q-Learning (DSGFQL) for automatically generating FNNs is proposed in this chapter. Distinguished from the Fuzzy Q-Learning (FQL) of [22], Dynamic Fuzzy Q-learning (DFQL) of [23] and the online Clustering and Q-value based Genetic Algorithm learning schemes for Fuzzy system design (CQGAF) of [24], the novel DSGFQL methodology can automatically create, delete and adjust fuzzy rules according to reinforcement evaluations of the entire system as well as individual fuzzy rules.

Besides applying RL for training consequent parameters of FNNs, the DSGFQL method also adopts the philosophy of RL in generating the structure and determining the premise parameters of FNNs. In the DSGFQL system, a reinforcement sharing mechanism is proposed for evaluating contributions of fuzzy rules. By this means, fuzzy rules can be evaluated, adjusted and eliminated accordingly.

The salient characteristics of the DSGFQL approach are: 1) Capable of structure

self-identification and automatic parameter estimation and modification; 2) RL is applied at both premise-generation and consequent-generation levels and FNNs can be quickly generated without SL; 3) Fuzzy rules can be evaluated, adjusted and deleted dynamically; 4) MFs of an FNN can be dynamically adjusted according to the reinforcement evaluation; 5) RL pruning techniques are proposed to remove redundant fuzzy rules.

4.1 A Typical Structure of Fuzzy Neural Networks

In this thesis, generation of a typical five-layer FNN is studied. This FNN is based on extended Ellipsoidal Basis Function (EBF) NNs [16] and similar to the Adaptive Network-based Fuzzy Inference Systems (ANFIS) in [7].

If we assume that the output variables of a Multi-Input Multi-Output (MIMO) rule are independent, an MIMO FIS can be represented as a collection of Multi-Input Single-Output (MISO) FNNs [1, 2].

The structure of EBF NNs for an MISO system is depicted in Figure. 4.1 as that in [15, 16, 18, 23].

Layer one takes the N -dimension input to the FNN and the input space is partitioned at layer two. Layer two is a fuzzification layer which evaluates the MFs of the input variables. The MF is chosen as a Gaussian function and each input variable x_i ($i = 1, 2, \dots, N$) has L MFs given by

$$\mu_{ij}(x_i) = \exp\left[-\frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}\right] \quad i = 1, 2, \dots, N, j = 1, 2, \dots, L \quad (4.1)$$

where μ_{ij} is the j th MF of x_i , while c_{ij} and σ_{ij} are the center and width of the j th Gaussian MF of x_i respectively.

Remarks: The input space refers to layers 1 and 2 and fuzzy MFs are generated, adjusted and deleted at layer 2.

4.1 A Typical Structure of Fuzzy Neural Networks

50

Figure 4.1: Structure of the Fuzzy Neural Networks System

Layer three is a rule layer. The output of the j th rule $R_j (j = 1, 2, \dots, L)$ in layer three is given by

$$f_j(x_1, x_2, \dots, x_N) = \exp\left[-\sum_{i=1}^N \frac{(x_i - c_{ij})^2}{\sigma_{ij}^2}\right] \quad j = 1, 2, \dots, L \quad (4.2)$$

if multiplication is adopted for the T-norm operator.

Normalization takes place in layer four and we have

$$\phi_j = \frac{f_j}{\sum_{i=1}^L f_i} \quad j = 1, 2, \dots, L \quad (4.3)$$

Lastly, nodes of layer five define output variables. If the Center-Of-Gravity (COG) method is performed for defuzzification, the output variable, as a weighted summation of the incoming signals, is given by

$$y = \sum_{j=1}^L \phi_j \omega_j \quad (4.4)$$

where ω_j is the consequent parameters of the FNNs, L is the number of fuzzy rules and ϕ_j is the normalized firing strength of the j th rule.

For the TSK model:

$$\omega_j = a_{j0} + a_{j1}x_1 + \dots + a_{jn}x_n, \quad j = 1, 2, \dots, L \quad (4.5)$$

The consequent parameters are real constants if the simplified fuzzy model (S-model) is applied as follows:

$$\omega_j = a_j, \quad j = 1, 2, \dots, L \quad (4.6)$$

Other MFs, such as triangular MFs, and other T-norm operators can also be applied similarly.

A fuzzy rule in this structure is expressed as follows:

$$\begin{aligned} & \text{If } x_1 \text{ is } (c_{1i}, \sigma_{1i}) \text{ and } x_2 \text{ is } (c_{2i}, \sigma_{2i}) \dots \text{ and } x_N \text{ is } (c_{Ni}, \sigma_{Ni}) \\ & \text{Then, } y \text{ is } a_i \end{aligned}$$

where (c, σ) are the center position and width of fuzzy neurons and a is the weight of fuzzy neurons in the system. Here, (c, σ) are regarded as premise parameters while a is considered as consequent parameters. In this thesis, a is also the local action selected through RL.

For generating the FNN, there are three issues to concern:

- How to determine the number of rules?
- How to allocate and adjust the premise parameters, e.g. center position and width of fuzzy neurons?
- How to determine the consequent parameters?

4.2 Previous Approaches of Generating the FNN

In literature, there are several research works done in generating this FNN. Wu and Er proposed Dynamic Fuzzy Neural Network (DFNN) as a hierarchical approach

of generating both the structure and parameters in [15], and they developed the DFNN for many applications and enhanced it as a General Dynamic Fuzzy Neural Network (GDFNN) with Gao et al later, in [4, 16, 18]. However, both DFNN and GDFNN are SL based approaches and they can only work when there are enough instructive training data. As mentioned before, in case of lacking instructive training data, RL is preferred.

There are several RL approaches for determining consequent parameters such as the Generalized Approximate Reasoning for Intelligent Control (GARIC) of [19], Lin and Lee's Reinforcement Neural-Network-based Fuzzy Logic Control System (RNN-FLCS) [142], Jouffe's Fuzzy Actor-Critic Learning and FQL [22], Horiuchi et al's Fuzzy Interpolation-Based Q-Learning [152], Berenji's Q-learning with fuzzy constraints [21], Er and Deng's DFQL [23] and the Juang's CQGAF [24]. However, how to determine the premise parameters and adjust the structure in an RL manner is still an open issue.

In [22], the author has extended the original Actor-Critic method of [140] and the Q-learning of [141] by combining RL with fuzzy systems so as to convert discrete states and actions to continuous ones by fuzzification and defuzzification. The consequent parts of the FNN are adjusted by the RL approach in [22]. However, structure and premise parameters are still determined by *a priori* knowledge.

To circumvent this problem, the DFQL was proposed in [23]. The main idea of the DFQL algorithm is to automatically generate fuzzy rules according to the ε -completeness and the "squared TD error" criteria. The TD error criterion proposed in [23] generates new rules when the squared TD error is too big. However, this "squared TD error" criterion in [23] is not used in an RL manner, e.g. reward the good and weaken the bad. Moreover, this criterion is not applicable for RL methods which are not based on TD, and the threshold values of this TD error criterion are hard to define as the meaning of that "squared TD error" is unclear.

In the DFQL, a new rule is created if the "squared TD error" is bigger than a threshold value. Creating a new rule may be necessary if the TD error is too negative, e.g. the performance is much worse than the estimation. However, it

is arguable whether a new rule is necessary when the TD error is much positive, e.g. the performance is much better than expectation. In such a case, the fuzzy system should increase and update the estimation function by TD learning rather than creating new rules.

In addition, the rules generated in the DFQL system may be more than necessary and there is no pruning method to delete those redundant rules.

Without using the "squared TD error" criterion, Juang proposed the CQGAF [24] based on an "aligned clustering algorithm" which is similar to the ε -completeness criterion in [23] to generate fuzzy structures.

Both the DFQL and CQGAF methods create fuzzy rules when the input space is not well partitioned. However, both methods cannot adjust the premise parameters except during rule creation. The center positions and widths of fuzzy rules are allocated by only considering input clustering. In fact, premise parameters also play important roles in the system performance and they should be evaluated, allocated and adjusted to improve the system performance as well.

Moreover, both the DFQL and CQGAF methods cannot delete fuzzy rules once they are generated even when the rules become redundant. Although the author of [24] declared that the number of rules could be reduced by a similarity test of the MFs, the generated rules cannot be deleted as there is no criterion to delete them. To reduce the computational cost and the training time, dormant or unnecessary rules should be deleted.

Therefore, we look for novel RL methodologies of generating FNNs with improvement and concern in these three issues:

- A proper RL method of generating the structure of FNNs with concern of the system performance.
- Dynamic allocation and adjustment of premise parameters of FNNs to improve and enhance the system performance.
- Pruning techniques by RL to remove redundant rules and optimize the struc-

ture of FNNs.

A weight decay method is discussed in the survey of [106], a pruning approach based on the output of each neuron unit is presented in [14] and the Error Reduction Ratio method presented in [79] is adopted as a pruning strategy in [15]. However, they are suitable for SL approaches only and cannot work well with insufficient training data. Therefore, novel pruning methods without SL are desired.

To circumvent the problems of applying RL in determining premise structure and parameters, a novel approach termed Dynamic Self-Generated Fuzzy Q-learning (DSGFQL) is proposed in this chapter.

Contrary to the existing methods, both the structure and preconditioning parts of an FNN can be generated and adjusted dynamically via RL in the proposed method. The DSGFQL method focuses on improvement and enhancement in structure identification and premises estimation by RL. Consequent parameters of FNNs are generated similarly as the FQL and DFQL approaches as in [22, 23].

Remarks: Although the proposed DSGFQL algorithm is presented based on the Q-learning approach, other RL approaches such as actor-critic algorithm [153], modified Q-Learning [154] and REINFORCE method [148] can also be applied similarly to the proposed DSGFQL method.

4.3 Fuzzy Q-Learning

The main focus of this thesis is utilizing RL methods for generating the structure and adjust the premise parameters of FNNs.

In this thesis, a novel RL approach is proposed for generating structure and premise parameters of FNNs. On the other hand, the FQL in [22, 23] is adopted for estimating the consequent parameters. The FQL is reviewed here.

4.3.1 Fuzzy structure for Q-learning

The FQL approach is an extension of the original Q-Learning [141] method into a fuzzy environment. Instead of discrete state in the original Q-learning, fuzzy state is adopted which is described by a vector of fuzzy variables. The state-space coding is realized at the first layer of the FNN. A learner may partially visit a fuzzy state, in the sense that real-valued descriptions of the state of the system may be matched by a fuzzy MF. Since more than one fuzzy state may be visited at the same time, possibly with different degrees, e.g. firing strengths, a smooth transition between a state and its neighbors, and consequently, smooth changes of actions done in different states can be achieved.

Both the actions and Q-functions are represented by an FNN whose fuzzy logic rules can be self-constructed based on the system performance. The FNN structure of the FQL system follows the structure in Figure 4.1 and the consequent parameters are the actions selected through the FQL approach.

In the FQL approach, the consequent parameters of FNNs are selected from an action set with M possible discrete actions for each rule R_i . An $L \times M$ Q-table memorizes the parameter vector q associated with the mapping of each actions and the fuzzy states. Local actions (consequent parameters) are determined by competition of the q-values so as to maximize the discounted sum of TD error obtained while achieving the task.

The FNN with competing actions for each rule R_i is built as follows:

$$\begin{aligned}
 R_i : \quad & \text{if } X \text{ is } S_i \quad \text{then} \quad Y \text{ is } a_1 \text{ with } q(S_i, a_1) \\
 & \quad \quad \quad \text{or} \quad \quad \quad a_2 \text{ with } q(S_i, a_2) \\
 & \quad \quad \quad \text{.....} \\
 & \quad \quad \quad \text{or} \quad \quad \quad a_M \text{ with } q(S_i, a_M)
 \end{aligned}$$

where X is the vector of input variables and S_i is the label of fuzzy sets that describe the fuzzy state of the i th rule.

The consequent parts of the FQL system is shown in Figure 4.2 as that in [23,

Figure 4.2: Consequent parts of the FQL system

105]. By weighting sum of the actions selected in each fired rules according to the normalized firing strengths ϕ , continuous global action is obtained.

Subsequently, the Q-values of the action-state pairs are updated by the TD method considering the participation of each rules.

4.3.2 Local action exploration and selection

In the FQL approach, there is a local action decided by each fuzzy rule. In order to explore the set of possible actions and acquire experiences through reinforcement signals, the actions are selected using an exploration-exploitation strategy of [22, 23].

In the exploration-exploitation strategy, a local action a is selected from possible discrete actions vector A , as follows:

$$a_\pi = \Pi_{a \in A}(q(S, a)) = \operatorname{argmax}_{a \in A}(q(S, a) + \eta(S, a)) \quad (4.7)$$

The term of exploration η stems from a vector of random value ψ (exponential distribution) scaled up or down to take into account the range of q values as

follows:

$$s_f = \begin{cases} 1 & \text{if } \max(q) = \min(q) \\ \frac{s_p(\max(q) - \min(q))}{\max(\psi)} & \text{otherwise} \end{cases} \quad (4.8)$$

$$\eta = s_f \psi \quad (4.9)$$

where s_p is the noise size and s_f is the corresponding scaling factor. Decreasing the s_p factor implies reducing the undirected exploration. Details of the exploration-exploitation strategy can be found in [22].

4.3.3 Generation of global actions

As a learner may visit more than one fuzzy states, the system output/global action is determined by weighting all the local actions. Assume that L fuzzy rules have been generated. At time step t , the input state is X_t and the normalized firing strength vector of each rule is ϕ . Each rule R_i has M possible discrete actions and the local action of each rule is selected from the action set A by competing with each other based on their q -values. The winning local action a_i of every rule cooperates to produce the global action based on the rule's normalized firing strength ϕ . Therefore, the global action is given by

$$U_t(X_t) = \sum_{i=1}^L \pi_A(q_t) \phi_t^i = \sum_{i=1}^L \phi_i^t a_i^t \quad (4.10)$$

where a_i^t is the selected action of rule R_i at time step t and U_t is the global action which is the same as y in Eq (4.4).

4.3.4 Update of Q-value

As defined before, Q function presents the action quality with respect to the state. In the FQL, Q-values are also obtained similarly as the global actions and they are weighted value of the q-values of those local winning actions. The Q function

is given by

$$Q_t(X_t, U_t) = \sum_{i=1}^L q_t(S_i, a_i^t) \phi_i^t \quad (4.11)$$

where U_t is the global action, a_i^t is the selected action of rule R_i and q_t is the q -value associated with the fuzzy state S_i and action a_i^t at time step t . Based on the TD learning, the value function corresponding to the rule optimal actions is defined as follows:

$$V_t(X_t) = \sum_{i=1}^L (\max_{a \in A} q_t(S_i, a)) \phi_i^t \quad (4.12)$$

and the global TD error is given by

$$TD_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - Q_t(X_t, U_t) \quad (4.13)$$

where r_{t+1} is the reinforcement signal received at time $t+1$ and γ is the discounted factor utilized to determine the present value of future TD errors. Note that this TD error term only needs to be estimated with quantities available at time step $t+1$.

This TD error can be used to evaluate the action just selected. If the TD error is positive, it suggests that the quality of this action should be strengthened for future use; whereas if the TD error is negative, it suggests that the quality should be weakened. The learning rule is given by

$$q_{t+1}(S_i, a_i^t) = q_t(S_i, a_i^t) + \alpha TD_{t+1} \phi_i^t, \quad i = 1, 2, \dots, L \quad (4.14)$$

where α is the learning rate.

4.3.5 Eligibility traces

In order to speed up learning, eligibility traces are used to memorize previously visited state-action pairs, weighted by their proximity at time step t [22, 23]. The

trace value indicates how state-action pairs are eligible for learning. Thus, it permits not only tuning of parameters used at time step t , but also those involved in past steps.

Let $Tr_t(S_i, a_j)$ be the trace associated with the discrete action a_i of rule R_i at time step t

$$Tr_t(S_i, a_j) = \begin{cases} \lambda Tr_{t-1}(S_i, a_j) + \phi_i^t & \text{if } a_j = a_i^t \\ \lambda Tr_{t-1}(S_i, a_j) & \text{otherwise} \end{cases} \quad (4.15)$$

where the eligibility rate λ is used to weight time steps. For all rules and actions, the parameter updating law given by Eq (4.14) becomes

$$q_{t+1}(S_i, a_j) = q_t(S_i, a_j) + \alpha T D_{t+1} Tr_t(S_i, a_j), \quad i = 1, 2, \dots, L, \quad j = 1, 2, \dots, M \quad (4.16)$$

and the traces are updated between action computation and its applications.

If we divide generation of FNNs into two levels, namely the premise-generation and the consequent-generation. At the premise-generation level, structure identification and estimation of premise parameters of FNNs are carried out while consequent-generation deals with determination of the consequent parameters. All those existing methods have only applied RL at the consequent-generation level. Therefore, we are going to propose a novel RL method for generating FNNs at the premise-generation level in this chapter.

4.4 Dynamic Self-Generated Fuzzy Q-Learning

4.4.1 ε -Completeness criterion for input space partitioning

In the FQL approach, the input variable fuzzy sets are used to represent appropriate high-dimensional continuous sensory spaces. In the conventional approaches [22], the number and position of the input fuzzy labels are set as *a priori* knowledge of the user. Generally speaking, if we do not have any knowledge about

the system, identical MFs whose domains can cover the region of the input space evenly are chosen. By this means, a fuzzy rule has to be considered for every possible combination of input fuzzy variables. However, the number of rules increases exponentially with increase in the number of input variables. As a consequence, the FNN often includes many redundant or improper MFs and fuzzy rules. This leads us to develop a learning algorithm that is capable of automatically determining the fuzzy rules online.

First of all, the ε -completeness criterion for judging clustering of the input space is adopted as in [15, 23, 24, 40]. As pointed out by the author of [24], a rule in an FNN corresponds to a cluster in the input space geometrically. An input data with higher firing strength of a fuzzy rule means that its spatial location is closer to the cluster center compared to those with smaller strengths. The definition of the ε -completeness of fuzzy rules is given in [40] as:

For any input in the operating range, there exists at least one fuzzy rule so that the match degree (or firing strength) is no less than ε .

The ε -completeness criterion is to check whether the whole input space has been completely covered with a certain degree (ε). If the criterion is not satisfied, it means more fuzzy rules should be created to accomplish the input space.

4.4.2 Allocation of newly generated rules

Assignment of membership functions

If the existing system does not satisfy the ε -completeness criterion, a new rule should be considered. Once a new rule is considered, the next step is to assign centers and widths of the corresponding MFs. The incoming multidimensional input vector X is projected to the corresponding one-dimensional MF for each input variable i ($i = 1, 2, \dots, N$). Assume that L MFs have been generated in the i th input variable and the Euclidean distance (ed_i) between the data x_i and the

boundary set B_i is computed as follows:

$$ed_i(j) = |x_i - B_i(j)|, \quad j = 1, 2, \dots, L + 2 \quad (4.17)$$

where $B_i \in \{x_{imin}, c_{i1}, c_{i2}, \dots, c_{iL}, x_{imax}\}$, we find

$$j_k = \arg \min_{j=1,2,\dots,L+2} (ed_i(j)) \quad (4.18)$$

If

$$ed_i(j_k) \leq k_{mf} \quad (4.19)$$

where k_{mf} is the predefined constant that controls the similarity of neighboring MFs, x_i is deemed completely represented by the existing fuzzy set $E_{ij_k}(c_{ij_k}, \sigma_{ij_k})$ without generating any new MF. Usually, k_{mf} is set between 0.1 and 0.3 for normalized input variables. Decreasing k_{mf} indicates that higher similarity between two MFs is allowed. Otherwise, if

$$ed_i(j_k) > k_{mf} \quad (4.20)$$

a new Gaussian MF is allocated whose center is with

$$c_{i(L+1)} = x_i \quad (4.21)$$

and the widths of MFs in the i th input variable are adjusted as follows:

$$\sigma_{ik} = \frac{\max(|c_{ik} - c_{i(k-1)}|, |c_{i(k+1)} - c_{ik}|)}{\sqrt{\ln(1/\epsilon)}}, \quad k = 1, \dots, L + 1 \quad (4.22)$$

where $c_{i(k-1)}$ and $c_{i(k+1)}$ are the two centers of adjacent MFs of the middle MF whose center is c_{ik} . Note that only the new MF and its neighboring MFs need to be adjusted. The main result concerning adjusting MFs to satisfy the ϵ -completeness of fuzzy rules has been proved in [4, 16]

Novel sharing mechanism for initialization of new q-values

Instead of randomly assigning q-values for the newly created rules, a novel initialization approach is proposed here. In order to reduce the training time, the initial q-values for the newly generated fuzzy rule are set by the nearest neighbors. By this means, the newly generated rules learn/share the training experience from the neighboring rules. For instance, if rules R_m and R_n are the two nearest neighbors to the newly generated rule R_i , then

$$q(S_i, a_j) = \frac{q(S_m, a_j)f_m(C_i) + q(S_n, a_j)f_n(C_i)}{f_m(C_i) + f_n(C_i)}, \quad j = 1, 2, \dots, M. \quad (4.23)$$

where $f_m(C_i)$ and $f_n(C_i)$ stand for the firing strengths of the newly generated center C_i to rules R_m and R_n , which can be obtained from Eq (4.2).

4.4.3 Global TD error evaluation

The objective of RL is to maximize the expected value function. Q-learning is a TD-based approach which utilizes the TD error for updating the estimated reward value function. In TD-based methods, the estimated value function is updated, as in [30], as follows:

$$V(s) \leftarrow V(s) + \alpha[r + \gamma V(s') - V(s)] \quad (4.24)$$

where $TD = r + \gamma V(s') - V(s)$ and $V(s)$ is the value function of state s .

Thus, we have

$$V(s) \leftarrow V(s) + \alpha TD \quad (4.25)$$

If the initial value of $V(s, 0)$ is 0, the value function can be regarded as

$$\begin{aligned} V(s, k) &= \alpha \sum_{t=0}^k [r(t) + \gamma V(s', t+1) - V(s, t)] \\ &= \alpha \sum_{t=0}^k TD(t) \end{aligned} \quad (4.26)$$

It can be seen that the TD error estimates the value function and it becomes a criterion for measuring the learning system as well the performance of FNNs. The bigger the TD error is, the greater the value function will be. Therefore, TD error can be considered as a criterion for measuring the system performance in RL.

Remarks: If negative TD error exists, it means that the performance of the system is worse than the value estimation/expectation. Therefore, the action should be discouraged and the system should try other options to overcome the under-estimation and decrease the estimation at the mean time. Thus, negative TD error reflects poor performance as well as poor estimation (over-estimation). On the other hand, if positive TD error exists, it means that the system performs well (even better than the estimation) but the estimation of the value function should be increased. Eventually, the TD error should converge to zero by increasing the estimation. If we focus on the system performance, the bigger TD error is, the better the system performance is. At the mean time, the TD learning methods will increase the estimation of value functions if a better value function is achieved than previous estimation.

As a criterion of system performance, the TD error is to be checked periodically, e.g. a number of training steps or an episode. At the end of each training episode or after several steps, the performance of the fuzzy system is examined by the TD error obtained during that period. Average TD error is adopted if the training environment is static. However, discounted TD error is to be considered for dynamic environment, i.e.

$$TD_{avg} = \begin{cases} \frac{\sum_{t=0}^{t=T} TD(t)}{T} & \text{Static environment} \\ \frac{\sum_{t=0}^{t=T} \gamma^t TD(t)}{T} & \text{Dynamic environment} \end{cases} \quad (4.27)$$

where T is the training time of each episode, $TD(t)$ is the TD error at time t and γ is the discounted factor which is less than 1.

By this means, the averaged system TD error is proposed as an index of the system performance.

4.4.4 Local TD error evaluation

Besides evaluation of system performance via reinforcement signals, contributions of each fuzzy rules are also measured through reinforcement signals in the proposed DSGFQL system. In order to evaluate the individual contributions of each fuzzy rules, a local TD error criterion is proposed. By this means, contributions or significance of each fuzzy rules are evaluated through the local TD errors. Inspired from the TD error sharing mechanism in [155], a local TD error scheme is proposed here to share the reinforcement with each local rule according to its contributions. The local TD error for each fuzzy rule is given, similarly as in [155], as follows:

$$TD_j^{local}(t) = \phi_j^t r(t+1) + \gamma \phi_j^{t+1} \max_{a_j \in A_j} q_j^t(s^t, a_j) - \phi_j^t q_j^t(s^t, a_j^t) \quad (4.28)$$

where $j = 1, 2, \dots, L$, and a_j^t is the action selected at time t .

Remarks: The local TD error adopted here is different from that in [155] in that the original form is multiplied by the firing strength of the corresponding fuzzy rule. In [155], the local TD error is used for selecting local actions while the local TD error is proposed as an evaluation criterion of individual rules in this thesis.

4.4.5 Participation of fuzzy rules

In the DSGFQL approach, the system/global TD error is selected as a measurement of system performance, the local TD error is adopted as an evaluation for individual contributions, while an averaged firing strength is proposed as a measurement for participation. If the averaged firing strength is lower than a threshold value, it means the significance of that rule is too low. Therefore, the rule can be eliminated to reduce the computational cost.

4.4.6 Reinforcement learning at premise-generation

If the global TD error is less than a threshold value, it means that the overall performance of the entire system is unsatisfactory. Therefore, the FNNs need to

be adjusted.

As mentioned before, RL emulates animal's learning behaviors and it also matches the common-sense ideas of [3, 30, 133] which state that:

If an action is followed by a satisfactory state of affairs or an improvement in the state of affairs, the tendency to produce that action is strengthened or reinforced (rewarded). Otherwise, that tendency is weakened or inhibited (penalized).

Previous studies only apply this learning behavior for selecting good or optimal actions in the consequent parts of FNNs [22–24] but never use this philosophy for selecting and adjust rules at a higher level. Hence, we propose a novel RL approach for determining the premises of FNNs here.

The philosophy of RL is adopted in the DSGFQL for generating structure and adjusting premise parameters of FNNs:

- If a *fuzzy rule* is followed by a satisfactory state of contributions or an improvement in the state of contributions, the tendency to select that rule is strengthened or reinforced when necessary.
- If a *fuzzy rule* is followed by an unsatisfactory state of contribution or a deterioration in the state of contributions, the tendency to select that rule is weakened or inhibited.
- If a *fuzzy rule* is acting as a "black sheep" in the system, the rule should be eliminated.
- Redundant fuzzy rules should be removed to save the computing cost.

4.4.7 Restructure of FNNs via reinforcement learning

In the proposed DSGFQL system, if the global average TD error is too negative e.g. it is less than a threshold value k_g , the overall performance is considered unsatisfactory. If the FNN passes the ε -completeness criterion but fails the average

TD error test, it means that the input space is well partitioned but the overall performance needs to be improved.

To resolve this problem, the weights of some good rules should be increased which means that the system will be modified by promoting the rule with the best performance, e.g. the best local TD error, to a more important position. As a result, the overall performance is improved as the rules with good contributions participate more in the system while those rules with poor contributions participate less, correspondingly. In this case, the width of the j th fuzzy rule's MF (the one with the best local TD error) will be increased as follows

$$\begin{aligned} \sigma_{ij} &\longleftarrow \kappa \sigma_{ij}, & i = 1, 2 \dots N & \text{ if } \kappa \sigma_{ij} \leq \sigma_{max} \\ &\longleftarrow \sigma_{max}, & & \text{ if } \kappa \sigma_{ij} > \sigma_{max} \end{aligned} \quad (4.29)$$

where κ is slightly larger than 1 and σ_{max} is the maximum width allowed for Gaussian MFs, which can be set to 1.2 if ε is chosen as 0.5 ($\sigma_{max} = \frac{1}{\sqrt{\ln(1/\varepsilon)}}$).

Remarks: The restructuring is only adopted when the system under-performs. Enlarging the MF of the rule will increase the firing strength of the rule. As a result, the global action is improved by becoming closer to the best local action.

On the other hand, if the local TD error is larger than a heavy threshold value k_{lh} , but smaller than a light threshold value k_{ll} , a punishment will be given to the rule by decreasing its width of the MF as follows:

$$\sigma_{ik} \longleftarrow \tau \sigma_{ik}, \quad i = 1, 2 \dots N \quad (4.30)$$

where τ is a positive value less than 1.

Remarks: This is a performance-oriented approach, in which fuzzy rules with the best local TD error values are to be promoted and those with unsatisfactory local TD error are to be demoted or even removed from the system.

4.4.8 Pruning of redundant fuzzy rules

The local TD error criterion offers a direct evaluation of contributions of fuzzy rules. The more negative the local TD error is, the worse result is offered by the fuzzy rule and vice versa. Therefore, a fuzzy rule should be punished if the local TD error is extremely poor. If the local TD error of a fuzzy rule is less than a heavy threshold value k_{lh} ($k_{lh} < k_{ll}$), the individual contributions are unsatisfactory and the fuzzy rule will be deleted as a serious punishment.

Removing problematic fuzzy rules can help to improve the overall performance of the FNN and new rules may be generated if the ε -completeness criterion fails due to the elimination of rules at the next step. By this means, performance can be improved as "black sheep" has been eliminated and it is better to restart the learning rather than staying in the problematic region.

Besides the TD error, firing strength should also be considered for system evaluation as it is a measurement for participation. If a fuzzy rule has very low firing strength during the entire episode or a long period of time recently, it means that this rule is unnecessary for the system. As more fuzzy rules mean more computation and longer training time, the rule whose mean firing strength over a long period of time is less than a threshold value, k_f , should be deleted.

Remarks: If a fuzzy rule keeps failing the light local TD error check, its firing strength will be reduced by decreasing width of its MF. When the width is reduced to a certain level, it will fail the firing strength criterion and will be deleted. The light local test gives a certain tolerance, which means the fuzzy rule is not deleted due to one slight fault. However, the fuzzy rule which does not provide satisfactory result is still punished by being demoted and it will be deleted if it keeps failing the light local test.

4.4.9 Gradualism of learning

The values of the thresholds for the TD error criteria are set according to the expectation of the task. At the early stage of training, each fuzzy rule needs time

to adjust its own performance. For a training system, it is natural to set the demanding requirement small at the beginning and increase it later when the training becomes stable. Thus, the idea of gradualism learning in [77], which uses coarse learning in the early training stage and fine learning in the later stage, is adopted here. Moreover, the elimination process is frozen in the early stage and rules are only pruned after a certain period of time.

In this chapter, a linear gradualism learning is proposed and the values of the thresholds for global and local TD error are set as follows:

$$k_g = k_g^{min} + (k_g^{max} - k_g^{min}) \frac{episodes}{episodes + K_r} \quad (4.31)$$

$$k_{lh} = k_{lh}^{min} + (k_{lh}^{max} - k_{lh}^{min}) \frac{episodes}{episodes + K_r} \quad (4.32)$$

$$k_{ll} = k_{ll}^{min} + (k_{ll}^{max} - k_{ll}^{min}) \frac{episodes}{episodes + K_r} \quad (4.33)$$

where k_g^{min} and k_g^{max} are the minimal and maximal values for the global TD error threshold values respectively, k_{lh}^{min} and k_{lh}^{max} are the minimal and maximal values for the heavy local TD error threshold values respectively, and k_{ll}^{min} and k_{ll}^{max} are the minimal and maximal values for the light TD error threshold values respectively. The term *episodes* is the number of training episodes or periods and K_r is a controlling constant which can be set according to the number of training episodes. Those threshold values are set according to the target of the training system. If the total TD error (value function) is expected to be V in T training steps, the values of thresholds of the TD error criterion should be set around the value V/T . The maximal threshold values should be slightly bigger than the minimal ones as the system trained by the DSGFQL is supposed to obtain better results via the training. Another suggested guidance is to check the performances of some classical controllers (such as basic fuzzy Controller and FQL controller) and set the results as the minimal threshold values.

Remarks: The gradualism learning is optional for applying the DSGFQL method.

The DSGFQL can also be applied with unique threshold values throughout the training, if users do not like to adjust the thresholds. Gradualism learning offers a framework or guideline in case that users are keen on adjusting the thresholds during the learning process.

4.5 Incorporation of Expert Knowledge in the DSGFQL System

Fuzzy rules provide a natural framework of incorporating human expert knowledge as the bias components for RL [156]. On one hand, fuzzy systems are MISO mappings from a real-valued vector to a real-valued scalar and precise mathematical formulas of these mappings can be obtained. On the other hand, fuzzy systems are knowledge-based systems constructed from human knowledge in the form of fuzzy If-Then rules. An important contribution of fuzzy system theory is that it provides a systematic procedure for transforming from a knowledge base to some nonlinear mappings [1]. In FNNs, we use knowledge-based linguistic information to construct an initial fuzzy system, and then adjust the structure and parameters of the fuzzy logic system based on numerical training of the neural networks.

The If-Then fuzzy rules corresponding to the domain knowledge pertaining to the tasks can be incorporated into the DSGFQL design. The premise of rules can be used to generate EBF units over the fuzzy input space and the consequents of rules can be used to generate the initial q-values, which are called bias in [23, 32]. Bias suggests actions for situations that otherwise would be time consuming to learn [32]. Thus, domain knowledge accelerates the learning process via bias since it focuses on the search process on promising parts of the action space immediately.

Depending on the confidence level of expert knowledge, the domain knowledge is incorporated as follows:

Extremely Confident Knowledge If expert knowledge is perfectly-defined, fuzzy rules with fixed parameters are incorporated in the system. No updating is

Figure 4.3: Neural network structure with incorporated knowledge

to be done for those fixed fuzzy rules.

Normal Expert Knowledge The q -value of the selected action a from the knowledge is initialized to a value $k_{conf}k_q$, while all the q -values of other actions are given random values according to a uniform distribution in $[0, k_q/2]$ and k_{conf} is the confident level of the knowledge.

The NN structure with incorporated knowledge is depicted in Figure.4.3, where the solid neurons are fixed fuzzy rules with extremely confident knowledge and the consequent weights are fixed. The neurons with diagonal lines are the preferred rules and with q -vectors which are initialized according to the confident level. Therefore, *priori* knowledge can be incorporated to the system.

The DSGFQL algorithm is an automatic method capable of self-tuning FNN, i.e., generating fuzzy rules according to system performance. The generated basic rules initialized from *priori* knowledge keep the learner safe and direct it in the right direction during the early stages of learning. The basic fuzzy rules, deduced from a human being's intuitive experience, can yield an action that is feasible but far from optimal.

It is almost impossible or difficult to find optimal fuzzy rules through a trial-and-error approach where a great number of variables are involved in the control task. In view of this, RL is added to tune the fuzzy rules online, which are eventually

overwritten and improved by more accurate learned actions. Because the basic fuzzy rules are used as starting points, it is possible to determine optimal parameters without too many iterations and the robot can be operated safely even during learning.

4.6 Summary

In this chapter, the DSGFQL approach, which is capable of both structure identification and parameter modification, has been proposed. The DSGFQL algorithm is a novel RL approach for generating FNNs. The existing methods, e.g. the FQL, DFQL and CQGAF etc, focus on applying RL (Q-learning) in selecting local actions at the consequent-generation level of FNNs. The DSGFQL approach adopts the philosophy of RL in generating the structure and determining the premises of FNNs at the premise-generation level.

At the end of each episode or a fixed training period, the overall performance of the FNN will be examined by the TD error scheme and the FNN will be restructured when necessary. At the same time, the local TD error criterion offers an approach of evaluating contributions of each fuzzy rules in the FNN systems. By this means, fuzzy rules can be evaluated, adjusted and eliminated accordingly.

If the system performance is unsatisfactory, the fuzzy rule with the best local TD error value will be promoted by increasing the width of its MF. On the other hand, if a fuzzy rule fails the local TD error criterion, it will be demoted or deleted. Inactive fuzzy rules with extremely small firing strength over a period of time are also removed to reduce the computational cost. At the mean time, the partitioning of the input space of an FNN is checked by the ε -completeness criterion. A new fuzzy rule is to be created if the FNN fails the ε -completeness criterion. Similarity tests will check whether a new MF is needed or not and a novel sharing mechanism is proposed for initializing Q-values.

The DSGFQL offers a novel view of generation FNNs. RL methodologies are not only applied for selecting optimal actions (consequent parameters) but also applied

Figure 4.4: Feedback controller with fuzzy reinforcement learning

in determining the number of rules, pruning and adjusting premise parameters. Similarly as rewarding good actions and penalizing poor actions in convectional RL approaches, good *rules* are promoted while bad *rules* are demoted or eliminated in the DSGFQL method. Therefore, instead of only focusing on applying RL in training consequent parameters, the RL is adopted at a higher level (premise-generation level).

Besides self-generating the FNN without any *priori* rules, the DSGFQL approach can also be incorporated by the domain knowledge from human experts or from previous training. The feedback controller is depicted in 4.4 and the flowchart of the DSGFQL algorithm is presented in Figure. 4.5.

Figure 4.5: Flowchart of the learning algorithm for DSGFQL

Chapter 5

Simulation Studies and Experiments on Mobile Robot Control

In this chapter, simulation studies and experiments have been carried out with applications to wall-following and obstacle avoidance control of a Khepera robot. Comparison studies of the DSGFQL approach with basic fuzzy controller, original fixed FQL, DFQL and CQGAF approaches have been accomplished. Both static and dynamic training environments have been adopted for comparative studies. Simulation results on the wall-following and obstacle avoidance tasks demonstrate the superiority of the proposed DSGFQL method. More efficient FNNs are generated through the proposed DSGFQL approach.

5.1 The Khepera Robot

The robot employed in the experiments described in this thesis is a miniature mobile robot called Khepera [157] shown in Figure 5.1. The Khepera mobile robot is cylindrical in shape, with 55 mm in diameter and 30 mm in height weighting only 70g. Its small size allows experiments to be performed in a small work area.

Figure 5.1: The miniature mobile robot: Khepera

The robot is supported by two lateral wheels that can rotate in both directions and two rigid pivots in the front and back.

The basic configuration of Khepera is composed of the CPU and the sensory/motor boards. The micro-controller includes all the features needed for easy interfacing with memories, with I/O ports and with external interrupts.

The sensory/motor board includes two DC motors coupled with incremental sensors, eight analogue infra-red (IR) proximity sensors denoted by $(S_0, \dots, S_7)$ in Figure 5.2, and an on-board power supply. Each IR sensor is composed of an emitter and an independent receiver. The dedicated electronic interface uses multipliers, sample/holds and operational amplifiers. This allows absolute ambient light and estimation, by reflection, of the relative position of an object to the robot to be measured. By this estimation, the distance between the robot and the obstacle can be derived. This estimation gives, in fact, information about the distance between the robot and the obstacle. The sensor readings are integer values in the range of $[0, 1023]$. A sensor value of 1023 indicates that the robot is very close to the object, and a sensor value of 0 indicates that the robot does not receive any reflection of the IR signal.

We have applied the proposed methodologies to a real robot to demonstrate the

Figure 5.2: Position and orientation of sensors on the Khepera

feasibility on real robots. In addition to the real robot, we also use the simulation version of the Khepera [158] for carrying out a comprehensive numerical comparison of the different approaches we are interested in. The program simulates Kheperas in the MATLAB environment. Simulated Kheperas are controlled in the same way as real physical Kheperas. Simulation studies on the wall following and obstacle avoidance tasks are presented in the following sections.

5.2 Wall Following Task for Khepera Robot

In this section, the Khepera robot is to be applied for a wall-following task. The aim of the experiment is to design a controller for wall following. In order to simplify the problem, we only consider robots moving in clockwise direction at a constant speed as that in [23]. Thus, we only need to deal with four input variables, which are the values of sensor S_i ($i = 0, 1, 2, 3$).

All these sensor values can be normalized within the interval $[0, 1]$. The output of the controller is the steering angle of the robot. In order for the robot to follow a wall, it must keep the distance from the wall while staying between a maximum distance, d_+ , and a minimum distance, d_- . The distance to the wall being followed, d , can be calculated via the sensor values. The robot receives a reward evaluation after performing every action U . The reward function depends on this action and the next situation as given in [23]:

$$r = \begin{cases} 0.1, & \text{if } (d_- < d < d_+) \text{ and } (U \in [-8^\circ, +8^\circ]) \\ -3.0, & \text{if } (d \leq d_-) \text{ or } (d_+ \leq d) \\ 0.0, & \text{otherwise.} \end{cases} \quad (5.1)$$

Here, $d_- = 0.15$ and $d_+ = 0.85$, which are normalized values, and U is the steering angle of the robot. These values are the same as the settings used in [23].

The training environments with lighted shaped walls used for a real robot and simulation studies are shown in Figure 5.3 and Figure 5.4 respectively.

Figure 5.3: Actual environment for wall-following experiments

Figure 5.4: MATLAB simulation environment for wall-following experiments

The performances of different approaches are evaluated at every episode of 1000 control steps according to two criteria, namely the number of failures which correspond to the total number of steps the robot has left the "lane" and rewards which

are accumulated. In order to compare the different approaches systematically and find appropriate parameters, we have done a number of comparison studies of these methods on simulations. At the same time, we have also implemented the proposed approach on a real robot to demonstrate the feasibility of the method in real robot.

5.3 Basic Fuzzy Controller

First, a fuzzy controller based on intuitive experiences is designed. For each input linguistic variable, we define two linguistic values: *Small* and *Big*, whose MFs cover the region of the input space evenly with the value of ε -Completeness set to 0.5. This means that there are 16 (2^4) fuzzy rules. Through trial and error, 16 fuzzy rules are obtained as a special case of the TSK fuzzy controller, whose consequents are constant, as in [23]. The 16 fuzzy rules are listed in Table 5.1.

Table 5.1: Basic fuzzy control rules for wall following

Rule	S_0	S_1	S_2	S_3	Steering angle
1	Small	Small	Small	Small	30
2	Small	Small	Small	Big	30
3	Small	Small	Big	Small	30
4	Small	Small	Big	Big	15
5	Small	Big	Small	Small	30
6	Small	Big	Small	Big	15
7	Small	Big	Big	Small	15
8	Small	Big	Big	Big	15
9	Big	Small	Small	Small	30
10	Big	Small	Small	Big	15
11	Big	Small	Big	Small	0
12	Big	Small	Big	Big	0
13	Big	Big	Small	Small	30
14	Big	Big	Small	Big	-15
15	Big	Big	Big	Small	-15
16	Big	Big	Big	Big	-15

If the robot only uses the basic fuzzy controller, it can actually follow the wall, but along inefficient trajectories. When only the basic fuzzy controller is used, the robot encounters 79 failures and -164.0 of reward value per episode on average. Certainly,

we can provide finer partitioning of the input space or tune the parameters of the MFs and consequents so as to obtain better performances. However, the number of rules will increase exponentially with increase in the number of input variables. Furthermore, tuning consequents of rules is time consuming because of the risk of creating conflicts among the rules. It is almost impossible or impractical to design an optimal fuzzy controller by hand due to a great number of variables involved. Similar to the idea of [23, 32], we incorporate RL into fuzzy controller design and the basic fuzzy rules designed from intuitive experiences are used as a starting point for learning. The confident coefficient k_{conf} is set to be 1. As a consequence, it overcomes some limitations of basic RL where an extremely long learning time is needed and unaccepted behavior may be generated during learning.

5.4 Fixed Fuzzy Q-Learning

We first consider the FQL approach of [22] that has a fixed structure of fuzzy rule sets. A total of 16 fuzzy rules, same as the basic fuzzy controller, are used. However, the consequents of the rules can be adjusted based on the FQL. Here, we simply use the undirected exploration method employed in [22] to select a local action a from possible discrete action vector A , as follows:

$$a_\pi = \Pi_{a \in A}(q(S, a)) = \operatorname{argmax}_{a \in A}(q(S, a) + \eta(S, a)) \quad (5.2)$$

The term of exploration η stems from a vector of random value ψ (exponential distribution) scaled up or down to take into account the range of q values as follows:

$$s_f = \begin{cases} 1 & \text{if } \max(q) = \min(q) \\ \frac{s_p(\max(q) - \min(q))}{\max(\psi)} & \text{otherwise} \end{cases} \quad (5.3)$$

$$\eta = s_f \psi \quad (5.4)$$

where s_p is the noise size and s_f is the corresponding scaling factor. Decreasing the

s_p factor implies reducing the undirected exploration. Details of the exploration-exploitation strategy can be found in [22]. An exploration rate of 0.001 in the simulation studies is chosen which is the same as the value adopted in [23].

The set of discrete actions is given by $A = [-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30]$. The initial q-values for incorporated fuzzy rules are set as, $k_q = 3.0$, with $k_{conf} = 1$, which are chosen according to the method described in Section 4.5. Other parameters in the learning algorithm are: Discounted factor, $\gamma = 0.95$; Trace-decay factor, $\lambda = 0.7$; TD learning rate, $\alpha = 0.05$. The controller with 81 (3^4) fuzzy rules whose MFs satisfy the 0.5 ε -completeness is also considered. Average performances of the two controllers during 40 episodes over 30 runs are shown in Figures 5.5 and 5.6.

Figure 5.5: Number of failures for comparison of performances of fuzzy controllers for (a)Basic fuzzy controller with 16 fuzzy rules, (b)16 fuzzy rules based on FQL, (c)81 fuzzy rules based on FQL

At the very beginning, performances of the two controllers based on the FQL are worse than that of the basic fuzzy controller due to the exploration feature of RL. The robot has to explore different actions in order to ensure that better actions are selected in the future. However, the performance of the robot is improved gradually and is better than that of the basic fuzzy controller. Therefore, it can be concluded that performance can be improved by explorations in the action set

Figure 5.6: Reward values for comparison of performances of fuzzy controllers for (a) Basic fuzzy controller with 16 fuzzy rules, (b) 16 fuzzy rules based on FQL, (c) 81 fuzzy rules based on FQL

and tuning consequent parameters.

To assess the effectiveness of finer partitioning of the input space, we compare the performances of the FQL using 16 rules and 81 rules. The speed of learning 81 rules is slower than that of 16 rules because a lot more parameters need to be explored. However, asymptotic performances of these two methods are almost the same. It is impractical to partition the input space further due to the curse of dimensionality. Therefore, automatic generation of premises of FNNs is desired.

5.5 DSGFQL Approach for Wall-following

The DSGFQL is proposed as a novel approach of determining premise parts of FNNs. In this section, the performance of the DSGFQL approach is to be assessed.

The parameter values for consequents training are the same as those used in the FQL approach. Other learning parameters for rule generation are ε -completeness, $\varepsilon = 0.5$ and similarity of MF, $k_{mf} = 0.3$. The training aim is to limit the number of failures to 60; therefore, the threshold values of the TD error criterion should be set around $(-3 \times 60 / 1000) = -0.18$. If it requires each rule to be active at least with an

average firing strength for 10 times in the 1000 training steps among about 50 rules, the firing strength threshold value is set as $10/(1000 \times 50) = 0.0002$. Therefore, the global TD error threshold values are $k_g^{max} = -0.05$ and $k_g^{min} = -0.45$; heavy local threshold values are $k_{lh}^{max} = -0.10$ and $k_{lh}^{min} = -0.30$; light local TD error threshold values are $k_{ll}^{max} = 0$ and $k_{ll}^{min} = -0.20$; the firing strength threshold values are $k_f = 0.0002$ and $K_r = 20$, $\kappa = 1.05$ and $\tau = 0.95$. These values give good performances of the algorithms in an initial phase. However, it should be pointed out that we have not searched the parameter space exhaustively.

The performances of the DSGFQL approach shown in Figures 5.7 and 5.8 which are also the mean values during 40 episodes over 30 runs.

Figure 5.7: Number of failures for comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) DSGFQL

As expected, the DSGFQL performs better than the FQL with respect to both number of failures and reward values. The number of fuzzy rules generated at every episode is shown in Figure 5.9. The MFs produced by the DSGFQL after learning input variables at one run are shown in Figure 5.10. The number of rules can be generated automatically online and does not increase exponentially with the increase in the number of input variables. At the same time, MFs can be adjusted dynamically according to the performance of controlling rules. Thus, a compact and excellent fuzzy controller can be obtained online.

Figure 5.8: Reward values for comparison of performances of fuzzy controllers for (a) 16 fuzzy rules based on FQL, (b) 81 fuzzy rules based on FQL, (c) DSGFQL

Figure 5.9: Number of rules generated by the DSGFQL approach

Figure 5.10: Membership functions after learning at one run

The reason why the DSGFQL method outperforms the FQL method is that the DSGFQL approach is capable of online self-organizing learning. Besides the input space partitioning, both overall and local performances have been evaluated to determine the structure of a fuzzy system. The common approach of conventional input-space partitioning is the so-called grid-type partitioning. The FQL with 16 rules partitions the state space coarsely, on the other hand, the speed of learning 81 rules is slow because a lot more parameters need to be explored. The proposed DSGFQL does not need to partition the input space *a priori* and is suitable for RL. It partitions the input space online dynamically according to both the accommodation boundary and the reinforcement criteria. The compact fuzzy system considers sufficient rules in the critical state space which requires high resolution and does not include redundant rules in unimportant or unvisited state space so that learning is rapid and efficient.

5.6 Adaptation in a New Environment

In order to show the adaptation capability of the proposed DSGFQL approach, performance of the learned DSGFQL navigation strategies is tested in an environ-

ment different from that used for training. This new environment is depicted in Figure 5.11. We can train the robot either to learn from the beginning by direct

Figure 5.11: New training environment for wall-following

training or to learn based on the knowledge obtained previously by re-training. The performances of the robot in the new environment by training directly and re-training with the FNN generated in the previous environment are shown in Figures 5.12 to 5.14.

Figure 5.12: Performance comparison of DSGFQL with training directly and re-training in a new environment: Number of failures vs number of episodes.

During re-training, the robot is re-trained in the new environment with the navigation strategies obtained from the previous training. Figures 5.12 to 5.14 show good adaptive capability of the DSGFQL approach. Some redundant rules are

Figure 5.13: Performance comparison of DSGFQL with training directly and re-training in a new environment: Reward value vs number of episodes.

Figure 5.14: Performance comparison of DSGFQL with training directly and re-training in a new environment: Number of fuzzy rules vs number of episodes.

removed while some new rules are generated. As a result, the robot is able to achieve satisfactory performance with the pre-obtained FNN and is able to adapt its previous knowledge to the new environment quickly.

5.7 Other Dynamic FQL Approaches

There are two extended FQL approaches which can dynamically generate a structure of FNNs proposed in [23] and [24].

In the DFQL of [23], the number of fuzzy rules is decided through the ε -completeness and the "squared TD error" criteria. An "aligned clustering" which is similar to the ε -completeness criterion has been adopted in the CQGAF to generate a fuzzy structure in [24]. Comparison studies of the proposed DSGFQL with both DFQL and CQGAF approaches are demonstrated in the wall-following task.

For the DFQL approach, the same settings which are exactly the same as those in the original paper [23] are adopted. The TD error factor for the DFQL is, $K = 50$, threshold for the TD error criterion is, $k_e=1$, ε -completeness, $\varepsilon = 0.5$ and similarity of MF, $k_{mf} = 0.3$. Details of setting these parameters are discussed in [23]. Actually, our settings of the DSGFQL approach just follow the settings of [23] and without any heuristic search in the parameter space.

The CQGAF proposed in [24] utilizes GA to obtain an optimal solution for the consequent actions and employs the aligned clustering algorithm for structure identification of the FNN. As this chapter focuses on self-generation of the FNN structure (premise-generation), the same action set as that for the DFQL and DSGFQL is applied for the CQGAF, which does not apply GA approach to train the action set. The aligned clustering algorithm in [24] is adopted and the consequent part of FNN is trained through normal FQL which does not apply the GA in the CQGAF approach.

As the aligned clustering algorithm is similar to the ε -completeness criterion in the DFQL but described in another way, all parameters for the aligned clustering

algorithm are set the same as those listed above for the DFQL.

The performances of these different approaches are evaluated at every episode of 1000 control steps. Three criteria, namely *number of failures*, *reward values* and *number of rules*, are considered for measuring the performance. The first two criteria are to measure how well the task has been performed and the last one is to measure how complex the fuzzy controller (FNN) is. The design purpose is to generate an efficient FNN which can provide satisfactory result with a small number of rules.

The performances of these three approaches have been measured and the mean values for 40 episodes over 30 runs are shown in Figures 5.15(a)-5.15(c).

To make the comparison clearer, the performances of the first 20 episodes are depicted in Figures 5.16(a)-5.16(c). It can be observed that these learning systems have achieved stable performances after 20 training episodes and the mean values of the last 20 episodes are chosen as a measurement for the performances achieved in the stable episodes. The mean values and standard derivations of the number of failures, reward values and the number of fuzzy rules after 20 episodes have been listed in Tables 5.2 and 5.3 for better numerical comparisons.

Table 5.2: Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following

	DFQL	CQGAF (without GA)	DSGFQL(TD)
Failures	63	57	55
Rewards	-102.8	-83.5	-82.2
Rules	53	39	29

Table 5.3: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following

	DFQL	CQGAF (without GA)	DSGFQL(TD)
Failures	6.51	4.38	5.60
Rewards	20.67	14.0	18.43
Rules	0.85	0.87	0.20

Compared with the DFQL [23], the number of failures has been reduced by 12.7% and the reward values have been increased by 20% in the DSGFQL system. At the same time, the number of rules generated in the DSGFQL system is 45% fewer

5.7 Other Dynamic FQL Approaches

89

(a) Number of failures vs number of episodes

(b) Reward values vs number of episodes

(c) Number of rules vs number of episodes

Figure 5.15: Comparison of performances of DFQL, CQGAF and DSGFQL in the wall-following task

(a) Number of failures vs number of episodes

(b) Reward values vs number of episodes

(c) Number of rules vs number of episodes

Figure 5.16: Comparison of performances of DFQL, CQGAF and DSGFQL in first 20 episodes of the wall-following task

than that of the DFQL system.

Compared with the CQGAF (without GA) method, the DSGFQL achieves a slight better performance and the number of rules generated in the DSGFQL is reduced by 25%. Therefore, more efficient FNNs are generated by the DSGFQL than those of DFQL and CQGAF systems.

From Figures 5.15(a)-5.15(c) and Tables 5.2 and 5.3, it can be concluded that the proposed DSGFQL algorithm is superior to the DFQL and CQGAF (without the GA) approach as the DSGFQL method achieves similar or even better performances than those approaches but with a much smaller number of rules. Therefore, the DSGFQL generates more efficient FNN than the DFQL and CQGAF (without GA).

Remarks: Here, the main concern is the efficiency of generated FNNs. Because the DSGFQL controller achieves satisfactory performance with much fewer number of rules, the FNN generated by the DSGFQL is more efficient than those generated by the DFQL and CQGAF approaches.

As mentioned in previous chapters, the "squared TD error" criterion in the DFQL may not be a proper criterion and it is hard to define. At the same time, it is not sufficient to consider only the input space clustering to decide the number of rules in the CQGAF method.

Both the DFQL and CQGAF methods achieve online structure identification by creating fuzzy rules when the input space is not well partitioned. However, both methods cannot adjust the premise parameters except when creating new rules. Moreover, both methods cannot delete fuzzy rules once they are generated even when the rules become redundant.

On the other hand, the proposed DSGFQL approach can dynamically create, adjust and delete fuzzy rules and MFs according to the performance as well as the input space clustering. Therefore, the proposed DSGFQL is superior to the DFQL and CQGAF approaches as more efficient FNNs are generated.

The superiority of the DSGFQL algorithm is due to the pruning capability as well

as self-modification of the fuzzy MFs. The number of rules have been reduced as redundant rules have been eliminated. Moreover, the MFs can be dynamically adjusted according to the performance evaluation while the MFs can only be adjusted according to the input clustering when a new rule is created in the DFQL and CQGAF approaches.

At the mean time, the proposed DSGFQL applies the philosophy of RL, e.g. rewarding good rules and penalizing bad rules. This criterion is easier to understand and apply rather than the "squared TD error" criterion of the DFQL.

5.8 Dynamically Changing Environments

Instead of using a static training environment throughout the entire training process, a dynamically changing environment is adopted. Four different environments are randomly selected for each training episode and the environments are depicted in Figure 5.17. Other settings are kept the same.

Figure 5.17: Dynamically changing environments for training.

The performances of these three approaches have been measured and the mean values for 40 episodes over 30 runs have been presented in Figures 5.18-5.20.

Figure 5.18: Comparison of performances of DFQL, CQGAF and DSGFQL in dynamically changing environments of the wall-following task: Number of failures vs number of episodes.

Figure 5.19: Comparison of performances of DFQL, CQGAF and DSGFQL in dynamically changing environments of the wall-following task: Reward value vs number of episodes.

Figure 5.20: Comparison of performances of DFQL, CQGAF and DSGFQL in dynamically changing environments of the wall-following task: Number of fuzzy rules vs number of episodes.

For better numerical analysis, mean values and standard derivations after 20 episodes are listed in Tables 5.4 and 5.5.

Table 5.4: Mean value of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task

	DFQL	CQGAF (without GA)	DSGFQL
Failures	54	51	48
Rewards	-76.7	-64.5	-60.1
Rules	50	37	28

Table 5.5: Standard derivations of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task

	DFQL	CQGAF (without GA)	DSGFQL
Failures	4.97	4.48	4.01
Rewards	15.93	13.89	13.30
Rules	0.89	0.57	0.54

From Figures 5.18-5.20 and Tables 5.4 and 5.5, it can be concluded that the DSGFQL approach is superior to the DFQL and CQGAF (without the GA) approaches as the DSGFQL method achieves similar or even better performances than those approaches but with much smaller number of rules. This is because fuzzy rules are not able to be pruned in the DFQL and CQGAF approaches. More-

over, the MFs of fuzzy rules can only be adjusted by generating new rules in the DFQL and CQGAF approaches.

In the DSGFQL, fuzzy neurons are dynamically created, adjusted and deleted according to the reinforcement as well as the input allocations. The philosophy of RL has been adopted in generating structure and determining the premises of FNNs.

Therefore, better structure identification and estimation of premise parameters have been achieved by the proposed DSGFQL approach. More efficient FNNs are generated by the DSGFQL approach.

5.9 Obstacle Avoidance Task

The task here is to design a controller for obstacle avoidance. Eight inputs from sensors $S_i (i = 0, 1, \dots, 7)$, are considered. The output of the controller is the steering angle of the robot. All parameters of the learning algorithms are set the same as those for the wall-following task.

The same reward function as what is used in [159] is adopted:

$$r = \begin{cases} 1, & \text{if } (\sum_{i=0}^7 S_i(t-1) - \sum_{i=0}^7 S_i(t)) > 0 \\ -1, & \text{if } (\sum_{i=0}^5 S_i(t) > 1800) \text{ or } (\sum_{i=6}^7 S_i(t) > 600) \\ 0, & \text{otherwise.} \end{cases} \quad (5.5)$$

where S_i is the i th input sensor value which is in the range of $[0, 1023]$ (the bigger value it is, the closer it is to the obstacle).

The training environment shown in Fig. 5.21 is adopted. Similar comparisons as those in the wall-following task are shown in Tables 5.6 and 5.7. The parameters are set the same as those in the wall-following task to show the results without exhaustive search in the parameter space.

From Tables 5.6 and 5.7, it can be concluded that the proposed DSGFQL algo-

Figure 5.21: Training environment for the obstacle avoidance task.

Table 5.6: Mean value of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance

	DFQL	CQGAF	DSGFQL(R)
Failures	75	76	71
Rewards	319.7	304.9	329.3.0
Rules	44	38	20

Table 5.7: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance

	DFQL	CQGAF	DSGFQL(R)
Failures	6.25	4.03	5.26
Rewards	11.06	8.40	7.80
Rules	1.93	1.79	0.629

rithm is superior to the DFQL and CQGAF without the GA approach as the DSGFQL method generate more efficient FNNs than those of the DFQL and CQGAF methods. In the DSGFQL system, similar or even better performance than those approaches are achieved with significantly fewer number of rules. Therefore, more efficient FNNs are generated by the DSGFQL.

In the DSGFQL approach, RL methods have been applied for generating and adjusting structure and parameters of FNNs successfully. The superiority of the DSGFQL algorithm is due to the pruning capability as well as self-modification of the fuzzy MFs via RL. In this task, it solves another benchmark problem with the same parameters as in the wall-following task.

5.10 Summary

In this chapter, detailed comparison studies among basic fuzzy controller, fixed FQL, DFQL, CQGAF and DSGFQL have been carried out through wall-following and obstacle avoidance tasks by a Khepera robot. Both static and dynamic training environments have been adopted for training. Compared with a basic fuzzy controller, the DSGFQL method offers an online tuning strategy which can adjust both structure and parameters of FNNs. Compared with the original FQL approach, which is with a predefined structure and only tunes the consequent parameters, the DSGFQL algorithm can dynamically determine the number of rules and estimate both premise and consequent parameters. The number of fuzzy rules does not increase exponentially. Therefore, the *curse of dimensionality* has been avoided by the DSGFQL approach. Both the DFQL and CQGAF approaches also provide online tuning capability of FNNs. The superiorities of the proposed DSGFQL algorithm to the DFQL and CQGAF methods are due to two reasons. Firstly, the DSGFQL approach adopts RL in generating structure and determining premises of FNNs. By this means, the DSGFQL approach adjusts the structure and premise parameters of FNNs not only by the input-space partitioning but also according to the reinforcement evaluation. Secondly, pruning methodologies have been proposed in the DSGFQL approach. Therefore, redundant fuzzy rules can be eliminated to reduce the computational cost. A summary of the comparison among the proposed DSGFQL algorithm and other Q-learning approaches for generating FNNs is given in Table 5.8.

Table 5.8: Comparison among basic FCL, FQL, DFQL, CQGAF and DSGFQL

	FQL	DFQL	CQGAF	DSGFQL
Recruit new rules	NIL	ε -completeness and TD error criteria	Aligned clustering algorithm	ε -completeness
Pruning rules	NIL	NIL	NIL	Reinforcement & firing strength criteria
Membership function modification	NIL	Adjusted only when generate new rules	Adjusted only when generate new rules	Adjusted when generate new rules & according to reinforcement
Consequents adjustment	Q-learning	Q-learning	Q-learning	Q-learning

Chapter 6

Extended DSGFQL Approaches I: Premise Generation

In the last two chapters, a novel approach termed DSGFQL has been proposed in generating both structure and parameters of FNNs in an RL manner. The DSGFQL methodology has been compared with other Q-learning-based approaches. In this chapter, two approaches which extend the DSGFQL approach in generating premises of FNNs are presented.

In the DSGFQL proposed in Chapter 5, TD error has been adopted as criteria for evaluating contributions of fuzzy rules. However, the criteria are not available and applicable for those non-TD-based RL problems. In order to develop a general scheme for generating and determining premise parts of FNNs for all RL problems, a reward-scheme has been proposed. The DSGFQL-reward scheme makes DSGFQL method more general and applicable for all RL problems as reward function is a basic element of all RL problems, e.g. including both TD-based and non-TD-based problems.

In the original DSGFQL algorithm, the widths of MFs are adjusted according to the reinforcement evaluation as well as the input-space clustering. However, the center positions of fuzzy neurons are fixed once generated. As the EBF networks are local approximation in the sense of feature representation, the centers of local

units (EBF neurons) should be adjusted to move to the real centers for representing the input features [160]. An extended SOM algorithm is proposed to adjust the center positions of fuzzy neurons for better feature representation. Therefore, an enhanced approach termed Enhanced Dynamic Self-Generated Fuzzy Q-Learning (EDSGFQL) is developed based on the proposed extended SOM algorithm. Both methods have been applied to the wall-following and obstacle avoidance tasks in this chapter.

6.1 Reward Scheme for the DSGFQL Algorithm

6.1.1 Global and local reward value functions

The objective of RL is to maximize the expected reward value. Therefore, the reward value function becomes a natural criterion for judging the performance of the RL rules. Average rewards are adopted for evaluating global or system performance if the training environment is static. However, discounted rewards are to be considered for dynamic environment, i.e.

$$r_{avg} = \begin{cases} \frac{\sum_{t=0}^{t=T} r(t)}{T} & \text{Static environment} \\ \frac{\sum_{t=0}^{t=T} \gamma^t r(t)}{T} & \text{Dynamic environment} \end{cases} \quad (6.1)$$

where T is the training time of each episode, $r(t)$ is the reward obtained at time t and γ is the discounted factor which is less than 1.

Remarks: As an extension of the TD error based DSGFQL, the reward scheme offers a wider approach of applying RL in structure identification and parameter estimation as it can be applied to both TD-based and non-TD-based RL methods.

Similar to the TD error scheme, local reward values are adopted as evaluations of individual fuzzy rules. A simple local reward sharing mechanism in [155] is used here to share the reinforcement with each local rule according to its contributions

as follows:

$$r_j^{local}(t) = \phi_j r(t) \quad j = 1, 2, \dots, L \quad (6.2)$$

where ϕ_j is the normalized firing strength of the j th rule and r is the system reward.

It is easy to prove that the summation of all local rewards is equal to the global value.

6.1.2 Reward scheme of the DSGFQL

In the reward value scheme, global and local reward values are adopted as criteria for evaluations of system performance and local contributions correspondingly when generating FNNs. This is similar to the global and local TD error in the original DSGFQL. If the global reward value is less than a threshold value, k_g , the structure of FNNs needs to be modified. Fuzzy rules with the best reward values are to be promoted while those with bad reward values are to be demoted or removed.

If the FNN passes the ε -completeness criterion but the global reward is less than the threshold value k_g , it means that the input space is well partitioned but the system performance needs to be improved. To resolve this problem, the weights of some good rules with higher local rewards should be promoted. By promoting rules with the best local performances, the global performance can be improved by becoming closer to the best local performance. In this case, the width of the j th fuzzy rule's MF (the one with the best local TD error) will be increased as follows

$$\begin{aligned} \sigma_{ij} &\leftarrow \kappa \sigma_{ij}, & i = 1, 2, \dots, N \quad \text{if } \kappa \sigma_{ij} \leq \sigma_{max} \\ &\leftarrow \sigma_{max}, & \text{if } \kappa \sigma_{ij} > \sigma_{max} \end{aligned} \quad (6.3)$$

where κ is slightly larger than 1 and σ_{max} is the maximum width allowed for Gaussian MFs, which can be set to 1.2 if ε is chosen as 0.5 ($\sigma_{max} = \frac{1}{\sqrt{\ln(1/\varepsilon)}}$).

It should be noted that the restructure is only to be done when the system is under-performance and the maximum width is set to avoid any local rule dominating the decision and the system being trapped in local optimal solutions.

On the other hand, if the local reward value of a fuzzy rule is less than a heavy threshold value k_{lh} , the rule is eliminated as the individual contributions of the rule are unsatisfactory. Similarly as the TD error scheme in the original DSGFQL, it is better to restart the learning rather than staying in the problematic region and the rule with extremely poor performance should be removed.

If the local reward value of the rule is larger than a heavy threshold value k_{lh} , but smaller than a light threshold value k_{ll} , a punishment will be given to the rule by decreasing its width of the MF as follows:

$$\sigma_{ik} \leftarrow \tau \sigma_{ik}, \quad i = 1, 2 \dots N \quad (6.4)$$

where τ is a positive value less than 1.

At the same time, if a fuzzy rule has very low firing strength during the entire episode or a long period of time recently, it means that this rule is unnecessary for the system. The rule whose mean firing strength over a long period of time is less than a threshold value, k_f , should be deleted.

6.1.3 Wall-following task by the reward value scheme

Similar to the TD error scheme, the reward value scheme of the DSGFQL algorithm is adopted for the wall-following task. All parameters and threshold values are set exactly the same as those for the TD error scheme without exhaustive search of the parameter space.

Similar to the previous chapter, performances of the two schemes of the DSGFQL approaches are evaluated at every episode of 1000 control steps. Three criteria, namely *number of failures*, *reward values* and *number of rules*, are considered for measuring the performance. The first two criteria are to measure how well the

task has been performed and the last one is to measure how much computational cost has been spent.

The performances of the two schemes have been measured and the mean values for 40 episodes over 30 runs have been presented in Figures 6.1-6.3. The mean values and standard derivations of the last 20 episodes are listed in Tables 6.1 and 6.2.

Figure 6.1: Comparison of performances of reward and TD error schemes of the DSGFQL in the wall-following task: Number of failures vs number of episodes.

Table 6.1: Mean value of failures, rewards and fuzzy rules after 20 episodes for the wall following task with TD error scheme

	DFQL	CQGAF	DSGFQL (TD)	DSGFQL(reward)
Failures	63	57	55	53
Rewards	-102.8	-83.5	-83.0	-75.5
Rules	53	39	29	29

Table 6.2: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for the wall following task with TD error scheme

	DFQL	CQGAF	DSGFQL(TD)	DSGFQL (R)
Failures	6.51	4.38	5.60	3.52
Rewards	20.67	14.03	18.43	11.31
Rules	0.84	0.87	0.20	0.28

From Figures 6.1-6.3 and Tables 6.1 and 6.2, it can be concluded that the proposed reward value scheme of the DSGFQL algorithm works as well as the TD error scheme and similar performances have been achieved. As reward scheme provides

Figure 6.2: Comparison of performances of reward and TD error schemes of the DSGFQL in the wall-following task: Reward value vs number of episodes.

Figure 6.3: Comparison of performances of reward and TD error schemes of the DSGFQL in the wall-following task: Number of fuzzy rules vs number of episodes.

wider applications than the original TD error scheme, the DSGFQL-reward scheme is a general scheme for all RL problems in generating premises of FNNs.

6.1.4 Performances with different parameter settings

In order to show the performance of the DSGFQL controller with different threshold values, we have randomly selected four other settings. Simulation results with different threshold values are shown in Tables 6.3 and 6.4 and the parameter settings are listed in Table 6.5.

Table 6.3: Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following with different threshold settings for the DSGFQL (reward) approach

	Set A	Set B	Set C	Set D	Set E
Failures	55	56	54	52	53
Rewards	-82.2	-86.4	-79.4	-74.6	-75.5
Rules	29	31	30	30	29

Table 6.4: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following with different threshold settings for the DSGFQL (reward) approach

	Set A	Set B	Set C	Set D	Set E
Failures	3.87	4.12	2.97	2.99	3.52
Rewards	12.67	13.73	9.64	9.65	11.31
Rules	0.96	0.46	0.65	0.28	0.28

Table 6.5: Different sets of parameters

	Set A	Set B	Set C	Set D	Set E
k_g^{max}	0	-0.1	-0.1	-0.1	-0.05
k_g^{min}	-0.3	-0.5	-0.2	-0.2	-0.45
k_{lh}^{max}	-0.2	-0.1	-0.35	-0.2	-0.1
k_{lh}^{min}	-0.45	-0.35	-0.6	-0.3	-0.3
k_{ll}^{max}	0.05	0	-0.1	-0.1	0
k_{ll}^{min}	-0.2	-0.25	-0.35	-0.2	-0.2

From Tables 6.3 and 6.4, there is slight variation with different threshold values. Gradualism learning offers flexibility for fine tuning the system. However, as long as the threshold values are set close to V/T as suggested in this thesis, the DSGFQL controllers can achieve satisfactory results.

Table 6.6: Mean value of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task

	DFQL	CQGAF	DSGFQL (TD)	DSGFQL(reward)
Failures	54	51	48	48
Rewards	-76.7	-64.5	-62.2	-60.1
Rules	50	37	29	28

Table 6.7: Standard derivations of failures, rewards and fuzzy rules after 20 episodes with dynamically changing environments of wall following task

	DFQL	CQGAF	DSGFQL (TD)	DSGFQL(R)
Failures	4.97	4.48	4.80	4.01
Rewards	15.93	13.89	15.83	13.30
Rules	0.89	0.57	0.48	0.54

6.1.5 Dynamically changing environments

Similar to the previous chapter, a dynamically changing environment is adopted. Performances of two extended approaches have been measured and the mean values for 40 episodes over 30 runs have been presented in Figures 6.4 to 6.6. For a better numerical analysis, mean values and standard derivations after 20 episodes are listed in Tables 6.6 and 6.7.

Figure 6.4: Comparison of performances of reward and TD error schemes of the DSGFQL in dynamically changing environments of the wall-following task: Number of failures vs number of episodes.

Figure 6.5: Comparison of performances of reward and TD error schemes of the DSGFQL in dynamically changing environments of the wall-following task: Reward value vs number of episodes.

Figure 6.6: Comparison of performances of reward and TD error schemes of the DSGFQL in dynamically changing environments of the wall-following task: Number of fuzzy rules vs number of episodes.

From Figures 6.4-6.6 and Tables 6.6 and 6.7, it can be concluded that DSGFQL with reward value scheme obtains similar performance as the TD error scheme. It should also be highlighted that the reward value scheme can be applied to all RL methods as reward value function is a basic element of all RL approaches. Therefore, for those non-TD-based RL problems, the reward scheme is still applicable for generating the structure and premises of FNNs.

6.2 Enhanced DSGFQL with Extended SOM

6.2.1 Hybrid DSGFQL with the SOM

In this chapter, an enhanced version of the DSGFQL approach termed EDSGFQL, applying an extended SOM algorithm, is proposed. The development of the SOM is inspired by research in neurobiology [120]. The author of [31] has proved that the SOM is a useful tool for providing real-time and on-line generalization in RL problems. Therefore, online SOM is performed for each incoming training pattern during the training of the EDSGFQL algorithm. Extended from the original SOM in [120], the concept of fuzzy region is applied to take the place of lattice point.

In the original SOM, the distances between the input data and the center points are considered. The neuron whose center is closest to the input is regarded as the best matching neuron. The shortcoming of this method is that it does not consider the width or the size of the fuzzy region. To circumvent this problem, the M-distance or firing strength of the input data is considered for selecting the winning neuron. The fuzzy neuron who is with the smallest M-distance or with the biggest firing strength is appointed as the best matching agent. Obviously, the extended method is more suitable as it considers each fuzzy neuron as a region instead of a point. In other words, the extended SOM algorithm considers not only the center position but also the width of the fuzzy region.

Similar to the original SOM, the winning fuzzy rule and its topological neighbors are updated so that they move closer to the current input $X(t)$ in the input space.

The update rules for the centers are as follows:

$$C_j(t+1) = C_j(t) + \beta(t)\phi_j^\nu h_j^\nu(t)[X(t) - C_j(t)], \quad j = 1, 2, \dots, L \quad (6.5)$$

where ν denotes the winning neuron, $\beta(t)$ is the adaptation coefficient and ϕ_j^ν is the firing strength of the center of the j th rule R_j to the winning neuron ν . The term h_j^ν is the neighborhood kernel centered on the winner unit and is given by

$$h_j^\nu(t) = \exp\left(-\frac{\|C_\nu - C_j\|^2}{2\sigma^2(t)}\right) \quad (6.6)$$

Here, $\beta(t)$ and $h_j^\nu(t)$ decrease monotonically with t .

Therefore, as an enhancement from the DSGFQL algorithm, the centers of MFs are updated according to the extended SOM algorithm for the EDSGFQL approach. By adopting the SOM, the center positions of fuzzy neurons are better adjusted than those in the DSGFQL approaches. The flowchart of the EDSGFQL algorithm is presented in Figure. 6.7.

6.2.2 Simulation studies of wall following

The EDSGFQL approaches (with both TD error scheme and reward scheme) have been applied to the wall-following task as well. All parameters and thresholds are set the same as those in previous sections for the DSGFQL approach. The adaption coefficient is $\beta=0.05$ and $\sigma=1$ for the EDSGFQL approach, which are normal settings for the SOM.

As in the earlier comparisons, the number of failures, reward values and the number of fuzzy rules are studied. Firstly, comparative studies among the EDSGFQL (reward scheme), DFQL and CQGAF (without GA) are presented. The performances of these three approaches have been measured and the mean values for 40 episodes over 30 runs have been presented in Figures. 6.8(a)-6.8(c). The EDSGFQL approach is superior to the DFQL of [23] and the CQGAF (without GA) of [24].

Figure 6.7: Flowchart of the learning algorithm for EDSGFQL

(a) Number of failures vs number of episodes

(b) Reward values vs number of episodes

(c) Number of rules vs number of episodes

Figure 6.8: Comparison of performances of wall-following by DFQL, CQGAF (without GA) and EDSGFQL (reward) with reward scheme (1-40 episodes)

It can be observed that these learning systems have achieved stable performances after 20 training episodes. To make the comparison clearer, the mean values of the last 20 episodes are chosen as a measurement for the performances achieved in the stable episodes and listed in Tables 6.8 and 6.9.

Table 6.8: Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following

	DFQL	CQGAF (without GA)	EDSGFQL
Failures	63	57	48
Rewards	-102.8	-83.5	-62.1
Rules	53	39	25

Table 6.9: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following

	DFQL	CQGAF (without GA)	EDSGFQL
Failures	6.51	4.38	4.33
Rewards	20.67	14.03	14.23
Rules	0.84	0.87	0.30

From Figures 6.8(a) to 6.8(c) and Tables 6.8 and 6.9, it can be concluded that the proposed EDSGFQL algorithm is superior to the DFQL and CQGAF without the GA approach as the EDSGFQL method achieves better performance than those approaches but with much smaller number of rules. The mean number of failures of the EDSGFQL system is 24% less than that of the DFQL and 15% less than that of the CQGAF (without GA), the mean number of rewards of the EDSGFQL system is 40% higher than that of the DFQL and 26% higher than that of the CQGAF while the mean number of rules of the EDSGFQL system is 53% less than that of the DFQL and 35% less than that of the CQGAF (without GA). Therefore, more efficient FISs are generated via the EDSGFQL approach.

At the same time, the EDSGFQL is also compared with the DSGFQL approach. The training results are shown in Figures 6.9(a) to 6.9(c) and the mean values and standard derivations of the number of failures, reward values and the number of fuzzy rules after 20 episodes are also listed in Tables 6.10 and 6.11.

From Table 6.10, the number of failures are reduced by about 10%, the reward values are increased by about 20% while the number of fuzzy rules are reduced by

Table 6.10: Mean value of failures, rewards and fuzzy rules after 20 episodes for wall-following

	DSGFQL(R)	DSGFQL(TD)	EDSGFQL(R)	EDSGFQL(TD)
Failures	53	55	48	50
Rewards	-75.5	-83.0	-62.1	-67.0
Rules	29	29	25	24

Table 6.11: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for wall-following

	DSGFQL(R)	DSGFQL(TD)	EDSGFQL(R)	EDSGFQL(TD)
Failures	3.52	5.60	4.33	3.68
Rewards	11.31	18.43	14.23	11.86
Rules	0.28	0.20	0.30	0.36

about 10%-15% than those of the original DSGFQL approaches.

Comparative studies show that the EDSGFQL approach is superior to the DSGFQL as the number of rules is significantly fewer with similar or even better performances achieved. By updating the centers of MFs through the extended SOM approach, the original DSGFQL is enhanced. Therefore, more efficient FNNs are generated and fuzzy neurons are better allocated by adopting the extended SOM approach.

6.2.3 Obstacle avoidance task

The obstacle avoidance task has also been carried out by the two extended approaches. Similar comparisons as those in the wall-following task are shown in Tables 6.12 and 6.13. The parameters are set the same as those in the wall-following task to show the results without exhaustive search in the parameter space.

From the results, it shows that the DSGFQL with the reward value scheme achieves similar performance as the TD error scheme and the EDSGFQL approaches are superior to the DSGFQL approaches which have been shown to be superior to the DFQL and CQGAF algorithms.

A summary of the comparison among the EDSGFQL, DSGFQL, CQGAF and DFQL approaches is list in Table 6.14.

(a) Number of failures vs number of episodes

(b) Reward values vs number of episodes

(c) Number of rules vs number of episodes

Figure 6.9: Comparison of performances of DSGFQL and EDSGFQL (reward scheme) in the wall-following task

Table 6.12: Mean value of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance

	DSGFQL(R)	DSGFQL(TD)	EDSGFQL(R)	EDSGFQL(TD)
Failures	71	71	66	68
Rewards	333.0	329.3	348.0	339.2
Rules	20	20	17	18

Table 6.13: Standard derivations of failures, rewards and fuzzy rules after 20 episodes for obstacle avoidance

	DSGFQL(R)	DSGFQL(TD)	EDSGFQL(R)	EDSGFQL(TD)
Failures	5.45	5.26	5.28	6.18
Rewards	8.74	7.80	14.35	14.51
Rules	0.459	0.629	0.489	0.55

Table 6.14: Comparison among DFQL, CQGAF, DSGFQL and EDSGFQL

	DFQL	CQGAF	DSGFQL	EDSGFQL
Recruit new rules	ε -completeness and TD error criteria	Aligned clustering algorithm	ε -completeness	ε -completeness
Pruning rules	NIL	NIL	Reinforcement & firing strength criteria	Reinforcement & firing strength criteria
Width adjustment	Adjusted only when generate new rules	Adjusted only when generate new rules	Adjusted when generate new rules & according to reinforcement	Adjusted when generate new rules & according to reinforcement
Center adjustment	NIL	NIL	NIL	Extended SOM

6.3 Summary

In this chapter, two extended approaches of the DSGFQL have been proposed in generating premises of FNNs. A novel reward value scheme has been proposed for the DSGFQL methodology. The reward value scheme works as well as the TD error scheme and it provides an option for solving those non-TD-based problems while the original TD error scheme can only be applied to TD-based problems. Therefore, the reward value scheme is a general approach for generating FNNs in all RL problems as reward value function is an essential element of RL.

In addition, an extended SOM algorithm has been proposed to enhance the original DSGFQL as the SOM is a useful tool for online generalization. Therefore, the hybrid DSGFQL and SOM result in an enhanced DSGFQL, e.g. the EDSGFQL. By utilizing the SOM for updating the center positions of fuzzy neurons, more efficient FNNs are generated. Simulation studies on path planning of mobile robot have demonstrated the enhancement by the EDSGFQL approaches.

Chapter 7

Extended DSGFQL Approaches

II: Consequents Generation

In this chapter, we are going to extend the DSGFQL approach in generating consequent parameters of FNNs. A hybrid DSGFQL with Millan et al's continuous action (CA) Q-learning [32] termed DSGFQL-CA is proposed in this chapter.

In the original DSGFQL, Jouffe's FQL [22] is adopted for determining consequent parts of FNNs. In Jouffe's FQL, local actions are selected from a fixed action set by competing their q-functions and the global/system action is a weighted summation of those local actions. As fixed action set is adopted in estimating consequent parameters, the consequent parameters are discrete ones. Even though continuous global action can be obtained through fuzzy inference and fuzzy reasoning, it takes time to adjust fuzzy structure to obtain optimal actions. It would be better if continuous consequent parameters of FNNs can be estimated.

On the other hand, in the continuous action Q-learning of [32], the resulting local action is an average of the discrete actions of the nearest units weighted by their Q-values. Therefore, continuous actions are generated. However, one of shortcomings of the continuous action Q-learning of [32] is that only the nearest fuzzy unit is used to select the action. Therefore, the system output (global action) is not explored sufficiently and the learner may get trapped to locally optimal actions of the rule.

By this means, instead of only determining the action from the nearest neuron in [32], all fuzzy rules are considered and weighted due to their firing strengths. Therefore, better exploration is achieved in the hybrid approach.

In this chapter, we are going to utilize continuous action Q-learning in determining continuous local actions, e.g. estimating consequent parameters. The final action is weighted according to the firing strengths of each rules. By this means, a continuous local action is obtained instead of discrete actions selected from the fixed action set in the original DSGFQL. At the same time, all rules are considered and weighted by their firing strengths instead of only considering the nearest fuzzy rule in Millan et al's continuous action Q-learning [32].

7.1 Continuous Action Q-Learning

In the literature, various research works have been done for applying RL in continuous state and action space [22, 23, 32, 161, 162]. In particular, the continuous action Q-learning approach, which is a recent approach restricted to the generation of continuous actions by means of Q-learning has been proposed in [32].

In the continuous action Q-learning, only the nearest unit is used to select the action. The resulting continuous action is an average of the discrete actions of the nearest units weighted by their Q-values. Instead of the exploration-exploitation strategy, the continuous-action Q-learning considers one action to each side of the optimal action, a_m , and generate the final action and Q-function according to the following rules [32]:

$$\begin{aligned}
 left &= \frac{1}{2+(q(S_i, a_m)-q(S_i, a_{m-1}))^2} \\
 right &= \frac{1}{2+(q(S_i, a_m)-q(S_i, a_{m+1}))^2} \\
 U &= a_m + right \times (a_{m+1} - a_m) + left \times (a_{m-1} - a_m) \\
 Q(U) &= \frac{q(S_i, a_m) + right \times q(S_i, a_{m+1}) + left \times q(S_i, a_{m-1})}{1 + right + left}
 \end{aligned} \tag{7.1}$$

where S_i is the state of i th rule, a_m is the selected action by the competition of Q-values and a_{m-1} and a_{m+1} are the neighboring actions of the selected action, U

is the final action and $Q(U)$ is the generated Q-function.

However, if the number of discrete actions is large, more neighboring discrete actions need to be considered for the continuous-action Q-learning. Otherwise, the continuous action is not explored sufficiently. The discrete actions whose Q-values are not good will degrade the continuous action.

On the other hand, the discrete action in the DSGFQL approach is selected according to the exploration-exploitation strategy for every fuzzy rule in the DSGFQL approach. The resulting continuous action is an average of the actions weighted by the firing strengths of fuzzy rules. It is more efficient to use the firing strengths of fuzzy rules as weights rather than only consider the neighboring actions. In order to compare these two algorithms, we only consider the difference of updating Q-values and assume that the others are the same [23]. The results (mean values over 30 runs) are shown in Figures 7.1 and 7.2.

Figure 7.1: Comparison of performances of DSGFQL and Continuous Action Q-learning in the wall-following task: Number of failures vs number of episodes.

At the very beginning, the performance of continuous-action Q-learning is better than that of DSGFQL because it always explores near optimal initial values. However, the performance is worse than that of the DSGFQL later since the learner may get trapped to locally optimal actions. Discrete actions whose Q-values are

Figure 7.2: Comparison of performances of DSGFQL and Continuous Action Q-learning in the wall-following task: Reward value vs number of episodes.

not good will degrade the continuous action.

7.2 Hybrid DSGFQL with Continuous Action Q-Learning

7.2.1 Local action selection

In the previous section, it has been shown that the proposed DSGFQL is superior to the continuous action Q-learning in [32].

Although the global action of the DSGFQL system is continuous, each local actions are selected from a discrete action set. Each discrete action selected is the local optimal action from the discrete action set which may be trapped in local minima.

In the DSGFQL, this problem has been solved by applying fuzzy system to generate continuous system actions and the continuous global actions are obtained by adjusting the fuzzy relationships among all the rules. Generating and adjusting structure of FNNs for an optimal continuous global action is time consuming.

Therefore, we look for extensions of the DSGFQL approach which can generate continuous local actions instead of discrete local actions.

In this section, the DSGFQL is combined with the continuous action Q-learning approach and the hybrid method termed as DSGFQL-CA is proposed. In the hybrid approach, each local action and q -function are generated as follows:

$$\begin{aligned}
 left_i &= \frac{1}{2+(q(S_i, a_{i,m})-q(S_i, a_{i,m-1}))^2} \\
 right_i &= \frac{1}{2+(q(S_i, a_{i,m})-q(S_i, a_{i,m+1}))^2} \\
 \bar{a}_i &= a_{i,m} + right \times (a_{i,m+1} - a_{i,m}) + left \times (a_{i,m-1} - a_{i,m}) \\
 \bar{q}(S_i, a_i) &= \frac{q(S_i, a_{i,m}) + right \times q(S_i, a_{i,m+1}) + left \times q(S_i, a_{i,m-1})}{1 + right + left}
 \end{aligned} \tag{7.2}$$

where S_i is the state of i th rule, $a_{i,m}$ is the selected action by the exploration-exploitation strategy and $a_{i,m-1}$ and $a_{i,m+1}$ are the neighboring actions of the selected local action, $\bar{a}_i$ is the final selected local action and $\bar{q}(S_i, \bar{a}_i)$ is the generated local q -function. The global action and q -function are averaged values of the local actions and q -functions weighted by the firing strengths of fuzzy rules respectively.

Remarks: Continuous action Q-learning is adopted in selecting local actions rather than system actions in [32]. Global actions and Q-values are the weighted sum by the firing strengths of each fuzzy rules. Therefore, better exploration is achieved and continuous actions are generated at both system and local levels.

7.2.2 Global action determination

The global action is obtained by weighting local actions with the firing strengths of the fuzzy rules. We have

$$U_t(X_t) = \sum_{i=1}^L \pi_A(\bar{q}_t) \phi_i^t = \sum_{i=1}^L \phi_i^t \bar{a}_i^t \tag{7.3}$$

where $\overline{a}_i^t$ is the selected action of rule R_i at time step t and U_t is the global action and it is the same as y in Eq. (4.4).

Remarks: The local action $\overline{a}_i^t$ is a continuous action value instead of a discrete action from the fixed action set.

Furthermore, we have

$$Q_t(X_t, U_t) = \sum_{i=1}^L \overline{q}_t(S_i, \overline{a}_i^t) \phi_i^t \quad (7.4)$$

where U_t is the global action, $\overline{a}_i^t$ is the selected action of rule R_i and $\overline{q}_t$ is the Q-value associated with the fuzzy state S_i and action $\overline{a}_i^t$ at time step t .

Remarks: Both $\overline{a}_i^t$ and $\overline{q}_t$ are obtained from Eq. (7.2) instead of selecting from the fixed action set and q-table directly.

7.2.3 Update Q-values

Based on the TD learning, the Q-values corresponding to the rule optimal actions are defined as follows:

$$V_t(X_t) = \sum_{i=1}^L (\max_{a \in A} q_t(S_i, a)) \phi_i^t \quad (7.5)$$

and the global TD error is given by

$$TD_{t+1} = r_{t+1} + \gamma V_t(X_{t+1}) - Q_t(X_t, U_t) \quad (7.6)$$

where r_{t+1} is the reinforcement signal received at time $t+1$ and γ is the discounted factor utilized to determine the present value of future rewards. Note that we have to estimate this error term only with quantities available at time step $t+1$.

The learning rule is given by

$$q_{t+1}(S_i, a_j) = q_t(S_i, a_j) + \alpha TD_{t+1} Tr_t(S_i, a_j), \quad i = 1, 2, \dots, L, \quad j = 1, 2, \dots, M \quad (7.7)$$

and the traces are updated between action computation and its applications.

7.2.4 Simulations and comparison studies

Simulation studies on the wall-following task have been carried out for comparing the hybrid DSGFQL-CA and the original DSGFQL. The original DSGFQL with action set $A = [-30, -25, -20, -15, -10, -5, 0, 5, 10, 15, 20, 25, 30]$ has been compared with the hybrid DSGFQL-CA approach. The results (mean values over 30 runs) are shown in Figures 7.3 and 7.4.

Figure 7.3: Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Number of failures vs number of episodes.

From Figures 7.3 and 7.4, it can be seen that the hybrid approach learns faster than the original DSGFQL and achieves better results than the original continuous action Q-learning. In order to see the difference clearly, the mean values of the first 10 runs are shown in Figures 7.5 and 7.6.

From Figures 7.3 and 7.6, it can be seen clearly that the learning speed of the hybrid approach is much faster than the original DSGFQL and the results are better than the original continuous action Q-learning.

Figure 7.4: Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Reward value vs number of episodes.

Figure 7.5: Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Number of failures vs number of episodes (1-10runs).

Figure 7.6: Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning in the wall-following task: Reward value vs number of episodes (1-10runs).

In order to do further comparison, DSGFQL with action set $A' = [-30, -20, -10, 0, 10, 20, 30]$ is adopted. With fewer number of actions in the action set, the superiority of the hybrid DSGFQL-CA to the original DSGFQL is greater.

The mean values of the performance index for the wall-following task are shown in Figures 7.7 and 7.8.

From Figures 7.7 and 7.8, it can be seen that the hybrid DSGFQL-CA learns much faster than the DSGFQL and achieves much better results than the original continuous action Q-learning. As better exploration strategy has been adopted in the DSGFQL-CA, the system results are better than those of continuous action Q-learning. At the same time, as continuous actions are generated in local actions, the system performance improves faster than the original DSGFQL. In the original DSGFQL, continuous actions are obtained via fuzzification and defuzzification and local actions are discrete values. Therefore, we need to adjust the fuzzy structure well in order to obtain suitable continuous actions. On the other hand, continuous actions are also obtained in the local actions in the hybrid DSGFQL-CA. Therefore, the DSGFQL-CA is better in generating continuous actions.

Figure 7.7: Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning (7 actions) in the wall-following task: Number of failures vs number of episodes.

Figure 7.8: Comparison of performances of DSGFQL, Continuous Action Q-learning and Hybrid DSGFQL with Continuous Action Q-Learning (7 actions) in the wall-following task: Reward value vs number of episodes.

7.3 Summary

In this chapter, the DSGFQL-CA is proposed to extend the DSGFQL in generating consequent parameters of FNNs. By combining with the continuous action Q-learning for generating continuous local actions, the DSGFQL is enhanced and faster learning is achieved. In the original DSGFQL, all the local actions are selected from a fixed action set and the consequent parameters of FNNs are selected from discrete values. Therefore, the DSGFQL is combined with continuous action Q-learning in the hybrid DSGFQL-CA. By this means, a continuous local action is obtained and the final action is weighted according to the firing strengths of each rules. Continuous actions are obtained both at system level as well as local level and the consequent parameters of FNNs in the DSGFQL-CA are continuous. Simulation studies demonstrate that the hybrid DSGFQL is superior to both the DSGFQL-CA and continuous action Q-learning.

Chapter 8

Conclusions and Recommendations

Conclusions obtained from this thesis are summarized in this chapter. At the same time, possible ways of extending this work are discussed and recommended.

8.1 Conclusions

In this thesis, a novel RL algorithm termed DSGFQL has been proposed for automatic generating and training FNNs. From comparison studies with other related approaches, the DSGFQL has been proved as an efficient method of generating FNNs.

Based on the evaluations of system performances, individual contributions as well as the input clustering, fuzzy rules can be created, adjusted and eliminated automatically, which allows us to circumvent the problem of generating fuzzy rules by hand or through SL.

In literature, there are many works in applying RL for training consequent parameters [22–24]. Instead of only applying RL at consequent-generation, the DSGFQL method offers an approach of utilizing RL for generating FNNs at structural and premise level. Fuzzy rules can be evaluated and then adjusted or removed accord-

ing to the reinforcement signals. Two sharing reinforcement mechanisms (TD error and reward function) have been proposed as criteria of evaluating contributions from each fuzzy rules.

In the DSGFQL algorithm, the philosophy of RL has been adopted in structure identification and estimation of premise parameters in generating FNNs. By reinforcing good fuzzy rules and penalizing bad rules, efficient structure of FNNs can be generated. In this RL method, individual contributions are evaluated by a sharing mechanism of reinforcement signal weighted by the firing strengths of those rules.

The conventional RL methodologies are more action-oriented and RL is applied for selecting optimal actions. On the other hand, the DSGFQL offers a novel RL method which is structure-oriented (or rule-oriented) and optimal structure is the training target of the DSGFQL approach. Fuzzy rules are evaluated, adjusted and eliminated according to the reinforcement evaluation and this explores a new topic of applying RL.

In order to test the performance of DSGFQL, two typical applications of mobile robots have been investigated. In Chapter 5, simulations on the Khepera robot for the wall-following and obstacle avoidance behaviors have demonstrated the efficiency of the DSGFQL method. Comparative studies with basic fuzzy controller, FQL, DFQL, CQGAF and the continuous action Q-learning have shown the superiority of the proposed methods. Furthermore, the adaptive capability of the DSGFQL approach has been tested in a new environment as well as dynamically changing environments. Therefore, it can be concluded that the proposed DSGFQL is a successful RL methods in evaluating contributions of fuzzy rules as well as determining premise structure of FNNs.

In order to extend the DSGFQL algorithm, we have explored this method in two directions: premises generation and consequents determination.

For identifying premise parts of FNNs, a reward scheme is proposed as a general RL scheme for applying RL in determining the structure and premise parameters

of FNNs. The reward scheme achieves similar results as the TD error scheme and it offers a wider range of applications as reward value function is one of the basic elements of RL problems. Therefore, it can be concluded that the reward scheme offers a proper RL scheme for generating premise parts of FNNs.

In addition, the SOM has been approved as a useful tool for providing real-time and on-line generalization in RL problems. By combining the SOM with the DSGFQL, an enhanced method has been achieved for estimating premise parameters. The EDSGFQL improves the allocation of center positions of those fuzzy neurons. By this means, it can be concluded that the DSGFQL is enhanced in the EDSGFQL for determining premise parameters of FNNs.

On the other hand, a continuous action Q-learning has been adopted in hybrid with the DSGFQL algorithm and local continuous actions can be obtained. Here, Q-learning is not used in a discrete manner but in a continuous domain. From comparative studies among the DSGFQL-CA, continuous action Q-learning and the original DSGFQL, the hybrid approach achieves better result than the original continuous action Q-learning and learns faster than the original DSGFQL. Therefore, it can be concluded that the DSGFQL is enhanced in the DSGFQL-CA and continuous consequent parameters can be estimated through the DSGFQL-CA.

From the viewpoint of FNNs, the DSGFQL method can be concluded as a learning technique which is capable of generating, tuning and deleting fuzzy rules automatically based on simple reinforcement signals. From the viewpoint of machine learning, the DSGFQL method can be concluded as a mechanism of implementing RL in determining the structure and adjust fuzzy rules in FNNs rather than just selecting optimal local actions. From the viewpoint of RL, the DSGFQL can be concluded as an approach which generalizes the continuous input space with fuzzy rules and generates global continuous-valued actions using fuzzy reasoning. The DSGFQL-CA can also generate local continuous actions by weighting the Q-values.

In addition, expert knowledge can be incorporated into the FNN structure according to the confidence level, through which the system can focus on the search process on promising parts of the system immediately and reduce the training time

significantly. On the other hand, the sharing mechanism of initializing Q-values offers a better strategy which learns from the neighbors and utilizes previous trained knowledge.

8.2 Recommendations for Further Research

There are several promising directions for further works based on the results presented in this thesis. We look briefly at some of these directions and discuss their potential usefulness.

8.2.1 Convergence property

As DSGFQL deals with continuous states as well as continuous actions, it is not easy to obtain a formal guarantee of convergence. Although we have demonstrated the DSGFQL methods in a number of simulations, the analysis of the performance of general function approximation based on nonlinear architecture in RL is still an open question. On the other hand, a fuzzy system can be represented as a linear architecture with fuzzy basis functions. By virtue of the value iteration algorithm with linear architectures [163], the convergence property can be established.

8.2.2 Evolutionary algorithms

Evolutionary algorithms, e.g., Genetic Algorithms (GAs) [164, 165], evolutionary strategies [166], and etc, have been adopted for generation of FNNs. A good survey of genetic fuzzy systems can be found in [167]. GAs for generation of FNNs may be adopted for improving the DSGFQL approaches.

A GA is a parallel global-search technique that emulates the processes of natural evolution including crossover, mutation and survival of the fittest. GA can, in effect, often seek many local minima and increase the likelihood of finding global minima representing the problem goals [168]. Inspired by the results of [24], Q-

value can be selected as the fitness for GA evolution. At the same time, the sharing reinforcement mechanisms proposed in this thesis can also be adopted as a criterion for the fitness or objective functions in the GA. However, we should bear in mind that evolutionary algorithms are slow and they could take a tremendous amount of time to obtain a less-than-optimal solution.

8.2.3 Incomplete sensor information and noisy data

It is more challenging and interesting to look for robotics applications based on incomplete sensor information or partially observed environment. Sometimes, environments may be too huge or too complicated for robots to gather complete information based on limited sensor information. Some researchers suggest using memory of previous actions and observations to disambiguate the current state to make a proper behavior and decision in hidden state tasks [169–171]. On the other hand, problems can be even more difficult and challenging that if noisy signal is received. Fuzzy logic may be a good tool to solve that problem and some learning algorithms in NN learning can also be helpful.

8.2.4 Swam robots and multi-agent systems

In this thesis, we have demonstrated robotics application in a single robot. However, it is more challenging in research on multi-agent systems, especially, swam robotics systems.

Swarm robotics is the study of a large number of relatively simple agents designed to perform a desired collective behavior. It emerges from the local interactions among agents and between the agents and the environment. It is a new direction of coordinating large number of robots and it is simply inspired from the observation of social insects such as ants, wasps and bees. Social insects are known to coordinate to accomplish a task which is beyond the capabilities of a single individual. It also offers a proving ground where artificial intelligence, computing, sensors, mechanics and communication technologies merge. The DSGFQL

8.2 Recommendations for Further Research

133

learning algorithm can be further deployed for multi-agent systems by considering system/team behavior as well as individual functions and contributions.

Author's Publications

Book Chapters

Meng Joo Er and Yi Zhou: Dynamically self-generated fuzzy neural networks with industry applications. *book chapter in Softcomputing Applications in Industry*, Springer-Verlag, pp. 47-68, 2008.

Yi Zhou and Meng Joo Er: Mobile robot navigation via reinforcement learning, *accepted as a book chapter in Mobile Robots: New Research*, Nova publishers.

Journal Papers

- (1) Meng Joo Er and Yi Zhou: Automatic generation of fuzzy inference systems via unsupervised learning, *accepted by Neural Networks*.
- (2) Yi Zhou and Meng Joo Er: A reinforcement learning approach towards automatic generation of fuzzy inference systems, *accepted by IEEE Transactions on Systems, Man, and Cybernetics, Part B*.
- (3) Meng Joo Er and Yi Zhou: A novel framework for automatic generation of fuzzy neural networks, *Neurocomputing*, Vol. 71, pp. 584-591, 2008.
- (4) Yi Zhou and Meng Joo Er: A novel approach for generation of fuzzy neural networks, *International Journal of Fuzzy Systems*, Vol. 9, No. 1, pp. 8-13, 2007.

- (5) Yi Zhou and Meng Joo Er: "Dynamic fuzzy Q-learning control of humanoid robots for automatic gait synthesis", *International Journal of Fuzzy Systems*, Vol. 8, No.4, pp. 190-199, 2006.
- (6) Meng Joo Er, Yi Zhou and Lin Chen: "Design of proportional parallel distributed compensators for nonlinear systems", *International Journal of Computer Applications in Technology*, Vol. 27, Nos. 2/3, pp.204-211, 2006.

Published Conference Papers

- (1) Meng Joo Er and Yi Zhou: Intelligent fuzzy Q-learning control of humanoid robots. *Lecture notes in computer science, vol. 3498, (Proceeding of International Symposium on Neural Networks 2005 (ISNN2005))*, J. Wang, X. Liao and Y. Zhang, Eds, in Chong Qin, China, pp.216-221, 2005.
- (2) Yi Zhou and Meng Joo Er: Dynamic fuzzy Q-learning control of uncertain systems with applications to humanoids. *Proceeding of 2005 IEEE Conference of Control Applications (CCA2005)* in Toronto, Canada, pp.459 - 464, 2005.
- (3) Meng Joo Er and Yi Zhou: Dynamic self-generated fuzzy systems for reinforcement learning. *Proceeding of International Conference on Computational Intelligence for Modeling Control and Automation 2005 (CIMCA05)* in Vienna, Austria, pp. 193-198, 2005.
- (4) Meng Joo Er and Yi Zhou: A novel reinforcement learning approach for automatic generation of fuzzy inference systems. *Proceeding of 2006 IEEE International Conference on Fuzzy Systems* in Vancouver, BC, Canada, pp. 100- 105, 2006.
- (5) Yi Zhou and Meng Joo Er: A hybrid self-learning approach for generating fuzzy inference systems, *Lecture Notes in Computer Science, Vol. 4233, (Proceeding of the 13th International Conference on Neural Information Pro-*

- cessing (ICONIP06)*), I. King, J. Wang, L. Chan, L. and D. Wang, (Eds.), in Hong Kong, pp. 1002-1011, 2006.
- (6) Meng Joo Er, Yi Zhou and Chiang-Ju Chien: Gait synthesis self-generation by dynamic fuzzy Q-learning control of humanoid robots, *Proceeding of 2006 IEEE International Conference on Systems, Man, and Cybernetics (IEEESMC06)* in Taipei, Taiwan, vol. 5, pp. 4231-4236, 2006.
- (7) Yi Zhou and Meng Joo Er: Self-generation of fuzzy inference systems by enhanced dynamic self-generated fuzzy Q-learning, *Proceeding of 9th International Conference on Control, Automation, Robotics and Vision (ICARCV2006)* in Singapore, 2006.
- (8) Jing Zhou, Meng Joo Er and Yi Zhou: Adaptive neural network control of uncertain nonlinear systems in the presence of input saturation, *Proceeding of 9th International Conference on Control, Automation, Robotics and Vision (ICARCV2006)* in Singapore, 2006.
- (9) Yi Zhou and Meng Joo Er: Self-learning in obstacle avoidance of a mobile robot via dynamic self-generated fuzzy Q-learning, *Proceeding of International Conference on Computational Intelligence for Modeling Control and Automation 2006 (CIMCA06)* in Sydney, Australia, 2006.
- (10) Yi Zhou, Meng Joo Er and Yu Wen: A hybrid approach for automatic generation of fuzzy inference systems without supervised learning, *Proceeding of American Control Conference 2007 (ACC2007)*, New York, USA, pp. 3371 - 3376, July, 2007.
- (11) Yi Zhou and Meng Joo Er: A novel Q-learning approach with continuous states and action, *Proceeding of IEEE Multi-conference on Systems and Control*, Singapore, 2007.

Bibliography

- [1] L. X. Wang, *A course in fuzzy systems and control*, Prentice Hall, NJ, USA, 1997.
- [2] K. M. Passino and S. Yurkovich, *Fuzzy control*, Addison Wesley Longman Inc., 1998.
- [3] J. S. R. Jang, C. T. Sun, and E. Mizutani, *Neuro-fuzzy and soft computing*, Prentice Hall, NJ, USA, 1997.
- [4] M. J. Er, S. Wu, and Y. Gao, *Dynamic fuzzy neural networks: architectures, algorithms and applications*, McGraw-Hill, NY, USA, 2003.
- [5] W. S. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
- [6] J. M. Zurada and A. Malinowski, Multilayer perceptron networks: selected aspects of training optimization, *Applied Mathematics and Computer Science*, vol. 4, no. 3, pp. 281–307, 1994.
- [7] J. S. R. Jang, Anfis: adaptive-network-based fuzzy inference system, *IEEE Trans. System, Man and Cybernetics*, vol. 23, no. 3, pp. 665–684, 1993.
- [8] J. J. Shann and H. C. Fu, A fuzzy neural network for rule acquiring on fuzzy control systems, *Int. J. of Control*, vol. 71, no. 3, pp. 345–357, 1995.

- [9] R. Setiono, W. K. Leow, and J. M. Zurada, Extraction of rules from artificial neural networks for nonlinear regression, *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 564–577, 2002.
- [10] C. T. Chao, Y. J. Chen, and C. C. Teng, Simplification of fuzzy-neural systems using similarity analysis, *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 26, no. 2, pp. 344–354, 1996.
- [11] N. J. Nilsson, *Introduction to machine learning*, online book, <http://ai.stanford.edu/~nilsson/mlbook.html>, 1996.
- [12] J. Platt, A resource-allocating network for function interpolation, *Neural Comput.*, vol. 3, no. 2, pp. 213–225, 1991.
- [13] V. Kadiramanathan and M. Niranjan, A function estimation approach to sequential learning with neural networks, *Neural Comput.*, vol. 5, no. 6, pp. 954–975, 1993.
- [14] Y. Lu, N. Sundararajan, and P. Saratchandran, A sequential learning scheme for function approximation using minimal radial basis function neural networks, *Neural Comput.*, vol. 9, no. 2, pp. 461–478, 1997.
- [15] S. Wu and M. J. Er, Dynamic fuzzy neural networks: a novel approach to function approximation, *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 30, no. 2, pp. 358–364, 2000.
- [16] S. Wu, M. J. Er, and Y. Gao, A fast approach for automatic generation of fuzzy rules by generalized dynamic fuzzy neural networks, *IEEE Trans. Fuzzy Systems*, vol. 5, no. 4, pp. 578–594, 2001.
- [17] M. J. Er, S. Wu, J. W. Lu, and H. L. Toh, Face recognition using radial basis function (RBF) neural networks, *IEEE Trans. Neural Networks*, vol. 13, no. 3, pp. 697–710, 2002.
- [18] M. J. Er and Y. Gao, Robust adaptive control of robot manipulators using generalized dynamic fuzzy neural networks, *IEEE Trans. Industrial Electronics*, vol. 50, no. 3, pp. 620–628, 2003.

- [19] G. A. Rummery, *Problem solving with reinforcement learning*, Ph.d thesis, Cambridge University, 1995.
- [20] C. K. Chiang, H. Y. Chung, and J. J. Lin, A self-learning fuzzy logic controller using genetic algorithms with reinforcements, *IEEE Trans. Fuzzy Systems*, vol. 15, no. 3, pp. 460–467, 1997.
- [21] H. R. Berenji, Fuzzy Q-learning: a new approach for fuzzy dynamic programming, in *Proceedings of the Third IEEE Conference on Fuzzy Systems*, 1994, vol. 1, pp. 486–491.
- [22] L. Jouffe, Fuzzy inference system learning by reinforcement methods, *IEEE Trans. Systems, Man, and Cybernetics, Part C*, vol. 28, no. 3, pp. 338–335, 1998.
- [23] M. J. Er and C. Deng, Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning, *IEEE Trans on Systems, Man and Cybernetics, Part B*, vol. 34, no. 3, pp. 1478–1489, 2004.
- [24] C. F. Juang, Combination of online clustering and Q-value based ga for reinforcement fuzzy systems, *IEEE Trans on Fuzzy Systems*, vol. 13, no. 3, pp. 289–302, 2005.
- [25] G. Hinton and T. J. Sejnowski, Eds., *Unsupervised learning*, Bradford Company, Scituate, MA, USA, 1999.
- [26] T. Kohonen, Ed., *Self-organizing maps*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.
- [27] K. I. Diamantaras and S. Y. Kung, *Principal component neural networks: theory and applications*, John Wiley & Sons, Inc., NY, USA, 1996.
- [28] A. Hyvarinen, J. Karhunen, and E. Oja, *Independent component analysis*, Wiley, NY, USA, 2001.
- [29] E. Oja, Unsupervised learning in neural computation, *Theor. Comput. Sci.*, vol. 287, no. 1, pp. 187–207, 2002.

- [30] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, The MIT Press, Cambridge, Massachusetts., 1998.
- [31] A. J. Smith, Applications of the self-organising map to reinforcement learning, *Neural Networks*, vol. 15, no. 8-9, pp. 1107–1124, 2002.
- [32] J. D. R. Millan, D. Posenato, and E. Dedieu, Continuous-action Q-learning, *Machine Learning*, vol. 49, no. 2-3, pp. 247–265, 2002.
- [33] S. Mitra and Y. Hayashi, Neuro-fuzzy rule generation: survey in soft computing framework, *IEEE Trans. Neural Networks*, vol. 11, no. 3, pp. 748–768, 2000.
- [34] S. C. Lee and E. T. Lee, Fuzzy sets and neural networks, *Journal of Cybernetics*, vol. 4, no. 2, pp. 83–103, 1974.
- [35] S. C. Lee and E. T. Lee, Fuzzy neural networks, *Mathematical Biosciences*, vol. 23, pp. 151–177, 1975.
- [36] S. K. Pal and S. Mitra, *Neuro-fuzzy pattern recognition: methods in soft computing*, John Wiley & Sons, Inc., NY, USA, 1999.
- [37] C. Zanchettin and T. B. Ludermir, A neuro-fuzzy model applied to odor recognition in an artificial nose, *Design and application of hybrid intelligent systems*, pp. 917–926, 2003.
- [38] L. A. Zadeh, Fuzzy sets, *Information and control*, vol. 8, pp. 338–358, 1965.
- [39] L. X. Wang, *Adaptive fuzzy systems and control: design and stability analysis*, Prentice Hall, Englewood Cliffs, NJ, USA, 1994.
- [40] C. C. Lee, Fuzzy logic in control systems: fuzzy logic controller, *IEEE Trans. System, Man and Cybernetics*, vol. 20, no. 2, pp. 404–435, 1990.
- [41] X. Li, S. K. Tso, and J. Wang, Real-time tool condition monitoring using wavelet transforms and fuzzy techniques, *IEEE Trans. Systems, Man and Cybernetics, Part C*, vol. 30, no. 3, pp. 352–357, 2000.

- [42] G. Feng, A survey on analysis and design of model-based fuzzy control systems, *IEEE Trans. Fuzzy Systems*, vol. 14, no. 5, pp. 676–697, 2006.
- [43] D. Dubois, H. Prade, and R. R. Yager, Eds., *Fuzzy information engineering: a guided tour of applications*, Wiley, NY, USA, 1997.
- [44] D. Dubois and H. Prade, Eds., *Fuzzy sets and systems: theory and applications*, Academic press, NY, USA, 1980.
- [45] R.E. Bellman and M. Giertz, On the analytic formalism of the theory of fuzzy sets, *Information Science*, vol. 5, no. 1, pp. 149–156, 1973.
- [46] T. Takagi and M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, *IEEE Trans. System, Man and Cybernetics*, vol. 15, no. 1, pp. 116–132, 1985.
- [47] A. D Kulkarni, Fuzzy Logic Fundamentals, *Computer Vision and Fuzzy-Neural Systems*, pp. 61–101, 2001.
- [48] Y. Tsukamoto, An approach to fuzzy reasoning method, in *Advances in fuzzy set theory and applications*, M. M. Gupta, R. K. Ragade, and R. R. Yager, Eds., North Holland, 1979, pp. 137–149, Amsterdam.
- [49] E. H. Mamdani and S. Assilian, An experiment in linguistic synthesis with a fuzzy logic controller, *Int. J. Man-Machine Studies*, vol. 7, no. 1, pp. 1–13, 1975.
- [50] W. Pedrycz, Fuzzy modelling fundamentals, construction and evaluation, *Int. J. Fuzzy Sets and Systems*, vol. 41, no. 1, pp. 1–15, 1991.
- [51] E. Kim, S. Ji, and M. Park, A new approach to fuzzy modeling, *IEEE Trans. Fuzzy Systems*, vol. 5, no. 3, pp. 328–337, 1997.
- [52] G. C. Mouzouris and M. Mendel, Dynamic non-singleton fuzzy logic systems for nonlinear modeling, *IEEE Trans. Fuzzy Systems*, vol. 5, no. 2, pp. 199–208, 1997.

- [53] J. Lygeros, A formal approach to fuzzy modeling, *IEEE Trans. Fuzzy Systems*, vol. 5, no. 3, pp. 317–327, 1997.
- [54] M. Braae and D. A. Rutherford, Fuzzy relations in a control setting, *Kybernetes*, vol. 7, no. 3, pp. 185–188, 1978.
- [55] L. Larkin, A fuzzy logic controller for aircraft flight control, in *Industrial applications of fuzzy Control*, M. Sugeno, Ed., North Holland, 1985, pp. 87–104, Amsterdam.
- [56] The fuzzy toolbox for MATLAB, The Math Work Inc.
- [57] J. M. Zurada, *Introduction to artificial neural systems*, West Publishing Co., St. Paul, MN, USA, 1992.
- [58] D. W. Patterson, *Artificial neural networks: Theory and Applications*, Prentice Hall, Singapore, 1996.
- [59] J. J. Buckley and Y. Hayashi, Fuzzy neural networks: a survey, *Fuzzy Sets and Systems*, vol. 66, pp. 1–13, 1994.
- [60] J. J. Buckley and Y. Hayashi, Neural nets for fuzzy systems, *Fuzzy Sets and Systems*, vol. 71, pp. 265–276, 1995.
- [61] P. Liu and H. Li, *Fuzzy neural network theory and application*, World Scientific, Singapore, 2004.
- [62] S. Ahmand and R. Mahapatra, M-trie: an efficient approach to on-chip logic-minimization, in *Proc. IEEE/ACM Conf. on Computer Aided Design*, 2004, pp. 428–435.
- [63] S. Aunet, Y. Berg, and T. Saether, Real-time reconfigurable linear threshold elements implemented in floating-gate cmos, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1244–1256, 2003.
- [64] A. Gopalan and A.H. Titus, A new wide range euclidean distance circuit for neural network hardware implementations, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1176–1186, 2003.

- [65] H. Hikawa, A digital pulse-mode neuron with piecewise linear activation function, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1028–1037, 2003.
- [66] J. Kowalski, 0.8/spl mu/m CMOS implementation of weighted-order statistic image filter based on cellular neural network architecture, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1366–1374, 2003.
- [67] M. Milev and M. Hristov, Analog implementation of ANN with inherent quadratic nonlinearity of the synapses, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1187–1200, 2003.
- [68] R. J. N. Gonzalez, F. V. Verdu, and A. R. Vazquez, Neuro-fuzzy chip to handle complex tasks with analog performance, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1375–1392, 2003.
- [69] S. Sato, K. Nemoto, S. Akimoto, M. Kinjo, and K. Nakajima, Implementation of a new neurochip using stochastic logic, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1122–1127, 2003.
- [70] W. Pedrycz, C.H. Poskar, and P. Czezowski, A reconfigurable fuzzy neural network with in-situ learning, *IEEE Micro Mag*, vol. 15, no. 4, pp. 19–30, 1995.
- [71] F. Yang and M. Paindavoine, Implementation of an RBF neural network on embedded systems: real-time face tracking and identity verification, *IEEE Trans. Neural Networks*, vol. 14, no. 5, pp. 1162–1175, 2003.
- [72] S. B. Yun, Y. J. Kim, S. S. Dong, and C. H. Lee, Hardware implementation of neural network with expansible and reconfigurable architecture, in *Proc. IEEE. Int. Conf. on Neural Information Processing*, 2002, vol. 2, pp. 970–975.
- [73] A. F. Gobi and W. Pedrycz, The potential of fuzzy neural networks in the realization of approximate reasoning engines, *Fuzzy Sets and Systems*, vol. 15, pp. 2954–2973, 2006.

- [74] M. J. D. Powell, Radial basis functions for multivariable interpolation: a review, *Algorithms for approximation*, pp. 143–167, 1987.
- [75] J. Moody and C. Darken, Fast learning in networks of locally tuned processing units, *Neural Computation*, vol. 1, no. 2, pp. 281–294, 1989.
- [76] Y. Tan, J. Wang, and J. M. Zurada, Nonlinear blind source separation using a radial basis function network, *IEEE Trans. Neural Networks*, vol. 12, no. 1, 2001.
- [77] K. B. Cho and B. H. Wang, Radial basis function based adaptive fuzzy systems and their applications to system identification and prediction, *Fuzzy Sets and Systems*, vol. 83, no. 3, pp. 325–339, 1996.
- [78] D. S. Broomhead and D. Lowe, Multivariable function interpolation and adaptive networks, *Complex Systems*, vol. 2, pp. 321–355, 1988.
- [79] S. Chen, C. F. Cowan, and P. M. Grant, Orthogonal least squares learning algorithm for radial basis function networks, *IEEE Trans. Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.
- [80] J. Nie and D. A. Linkens, Learning control using fuzzified self-organizing radial basis function network, *IEEE Trans. Fuzzy Systems*, vol. 1, no. 4, pp. 280–287, 1993.
- [81] J. S. R. Jang and C. T. Sun, Functional equivalence between radial basis function networks and fuzzy inference systems, *IEEE Trans. Neural Networks*, vol. 4, no. 1, pp. 156–159, 1993.
- [82] H. O. Nyongesa D. A. Linkens, Learning systems in intelligent control: an appraisal of fuzzy, neural and genetic algorithm control applications, in *IEE Proceedings on Control Theory and Applications*, 1996, vol. 143, pp. 367–386.
- [83] H. Takagi and I. Hayashi, Nn-driven fuzzy reasoning, *Int. J. Approx. Reasoning*, vol. 5, no. 3, pp. 191–212, 1991.

- [84] C. T. Lin, *Neural fuzzy control systems with structure and parameter learning*, World scientific, Singapore, 1994.
- [85] C. T. Sun, Rule-base structure identification in an adaptive-network-based fuzzy inference system, *IEEE Trans. Fuzzy Systems*, vol. 2, no. 1, pp. 64–73, 1994.
- [86] B. Kosko, Fuzzy systems as universal approximators, in *Proc. IEEE Int. Conf. on Fuzzy Systems*, San Diego, CA, USA, 1992, vol. 43, pp. 1329–1333.
- [87] H. R. Berenji and P. Khedkar, learning and tuning fuzzy logic controllers through reinforcements, *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 724–740, 1992.
- [88] J. Nie and D. A. Linkens, *Fuzzy-neural control: principles, algorithms and applications*, Prentice Hall International (UK) Ltd., Hertfordshire, UK, 1995.
- [89] A. E. Gaweda and J. M. Zurada, Data-driven linguistic modeling using relational fuzzy rules, *IEEE Trans. Fuzzy Systems*, vol. 11, no. 1, pp. 121–134, 2003.
- [90] W. Duch, R. Setiono, and J. M. Zurada, Computational intelligence methods for rule-based data understanding, *Proceedings of the IEEE*, vol. 92, no. 5, pp. 771–805, 2004.
- [91] M. M. Gupta and D. H. Rao, On the principles of fuzzy neural networks, *Fuzzy Sets and Systems*, vol. 61, no. 1, pp. 1–18, 1994.
- [92] L. X. Wang and J. M. Mendel, Fuzzy basis function, universal approximation, and orthogonal least squares learning, *IEEE Trans. Neural Networks*, vol. 3, no. 5, pp. 807–814, 1992.
- [93] C. F. Juang and C. T. Lin, An online self-constructing neural fuzzy inference network and its applications, *IEEE Trans. Fuzzy Systems*, vol. 6, no. 1, pp. 12–32, 1998.

- [94] C. K. Chak, G. Feng, and J. Ma, An adaptive fuzzy neural network for mimo system modelapproximation in high-dimensional spaces, *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 28, no. 3, pp. 436–446, 1998.
- [95] R. P. Li and M. Mukaidono, A new approach to rule learning based on fusion of fuzzy logic and neural networks, *IEICE Trans. Inf. Syst*, vol. E78-d, no. 11, pp. 1509–1514, 1995.
- [96] G. V. S. Raju and J. Zhou, Adaptive hierarchical fuzzy controller, *IEEE Trans. System Man and Cybernetics*, vol. 23, no. 4, pp. 937–980, 1993.
- [97] M. Sugeno and G. T. Kang, Structure identification of fuzzy model, *Fuzzy sets and systems*, vol. 28, no. 1, pp. 151–33, 1988.
- [98] M. Sugeno and K. Tanaka, Successive identification of a fuzzy model and its applications to prediction of a complex system, *Fuzzy Sets and Systems*, vol. 42, no. 3, pp. 315–334, 1991.
- [99] T. Kavli, ASMOD: an algorithm for adaptive spline modeling of observation data, *Int. J. of Control*, vol. 58, no. 4, pp. 947–968, 1993.
- [100] T. A. Johansen and B. A. Foss, Identification of non-linear system structure and parameters using regime decomposition, *Automatica*, vol. 31, no. 2, pp. 321–326, 1995.
- [101] R. Rovatti and R. Guerrieri, Fuzzy sets of rules for system identification, *IEEE Trans. Fuzzy Systems*, vol. 4, no. 2, pp. 89–102, 1996.
- [102] L. T. Koczy and K. Hirota, Size reduction by interpolation in fuzzy rule bases, *IEEE Trans. Systems, Man and Cybernetics*, vol. 27, no. 1, pp. 14–25, 1997.
- [103] O. Nelles, O. Hecker, and R. Isermann, Automatic model selection in local linear model trees (LOLIMOT) for nonlinear system identification of a transport delay process, in *11th IFAC Symposium on System Identification*, Japan, 1997, pp. 727–732.

- [104] L. X. Wang, Universal approximation by hierarchical fuzzy systems, *Fuzzy sets and systems*, vol. 93, no. 22, pp. 223–230, 1998.
- [105] C. Deng, *Fuzzy reinforcement learning and its applications to mobile robot navigation*, Ph.d thesis, Nanyang Technological University, Singapore, 2005.
- [106] R. Reed, Pruning algorithms - a survey, *IEEE Trans. Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.
- [107] C. T. Lin and C. S. G. Lee, Neural-network-based fuzzy logic control and decision system, *IEEE Trans. Comput.*, vol. 40, no. 12, pp. 1320–1336, 1991.
- [108] P. Langley, *Elements of machine learning*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.
- [109] T. M. Mitchell, *Machine learning*, McGraw-Hill Higher Education, 1997.
- [110] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, Eds., *Machine learning, neural and statistical classification*, Ellis Horwood, Upper Saddle River, NJ, USA, 1994.
- [111] Simon Haykin, *Neural networks: a comprehensive foundation*, Prentice Hall, Upper Saddle River, NJ, USA, 1998.
- [112] E. Oja, A simplified neuron model as a principal components analyzer, *J. Math. Biol.*, vol. 15, pp. 267–273, 1982.
- [113] P. Foldiak, Adaptive network for optimal linear feature extraction, in *Proc. Int. Joint. Conf. on Neural Networks*, Washington, DC, USA, 1989, pp. 401–406.
- [114] P. Tavan J. Rubner, A self-organizing network for principal component analysis, *Europhys Letters*, vol. 10, no. 7, pp. 693–698, 1989.
- [115] L. Wang and J. Karhunen, A unified neural bigradient algorithm for robust pca and mca, *Int. J. Neural Systems*, vol. 7, no. 1, pp. 53–67, 1996.

- [116] H. H. Harman, *Modern factor analysis*, University of Chicago Press, Chicago, USA, 1967.
- [117] M. Kendall and A. Stuart, *The advanced theory of statistics*, vol. 1-3, MacMillan, NY, USA, 1976 -1979.
- [118] A. Webb, *Statistical pattern recognition*, Arnold, Paris, 1999.
- [119] G. E. Hinton and T. J. Sejnowski, Learning and relearning in boltzmann machines, *Parallel distributed processing: explorations in the microstructure of cognition, vol. 1: foundations*, pp. 282–317, 1986.
- [120] T. Kohonen, Self-organized formation of topologically correct feature maps, *Biological Cybernetics*, vol. 43, pp. 59–69, 1982.
- [121] P. R. Krishnaiah and L. N. Kanal, Eds., *Classification, pattern recognition and reduction of dimensionality. Handbook of statistics*, vol. 2, Amsterdam: North Holland/Elsevier, 1982.
- [122] C. V. Malsburg, Self-organization of orientation sensitive cells in the striate cortex, *Kybernetik*, vol. 14, pp. 85100, 1973.
- [123] H. Ritter, T. Martinetz, and K. Schulten, *Neural computation and self-organizing maps: an introduction*, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1992.
- [124] S. Kaski and E. Oja, *Kohonen maps*, Elsevier Science Inc., NY, USA, 1999.
- [125] M. V. Hulle, *Faithful representations and topographic maps*, Wiley, NY, USA, 2000.
- [126] T. Villmann, *Topology preservation in self-organizing maps*, chapter Kohonen maps, pp. 267–292, Elsevier, Amsterdam, 1999.
- [127] R. Der and M. Herrmann, *Second-order learning in self-organizing maps*, chapter Kohonen maps, pp. 293–302, Elsevier, Amsterdam, 1999.

- [128] T. Heskes, *Energy functions for self-organizing maps*, chapter Kohonen maps, pp. 303–316, Elsevier, Amsterdam, 1999.
- [129] The SOM toolbox for MATLAB, Available at www.cis.hut.fi/projects/somtoolbox/.
- [130] Bibliography of SOM papers: a reference list of over 5000 studies on the self-organizing map, Available at www.cis.hut.fi/research/SOM-bibl/, 2000.
- [131] C. Zhou and Q. Meng, Dynamic balance of a biped robot using fuzzy reinforcement learning agents, *Fuzzy Sets and Systems*, vol. 134, no. 1, pp. 169–187, 2003.
- [132] M. J. Er and Y. Zhou, Intelligent Fuzzy Q-Learning Control of Humanoid Robots, *Lecture notes in computer science*, vol. 3498/2005, 2005.
- [133] R. S. Sutton, A. G. Barto, and R. J. Williams, Reinforcement learning is direct adaptive optimal control, in *Proceedings of the 1991 American Control Conference*, 1991.
- [134] R. S. Sutton, Reinforcement learning architectures for animats, in *Proceedings of the first international conference on simulation of adaptive behavior on from animals to animats*, Cambridge, MA, USA, 1990, pp. 288–296, MIT Press.
- [135] D. Michie and RA Chambers, BOXES: An experiment in adaptive control, *Machine Intelligence*, vol. 2, no. 2, pp. 137–152, 1968.
- [136] A. Barto and M. Duff, Monte Carlo matrix inversion and reinforcement learning, *Advances in Neural Information Processing Systems*, vol. 6, pp. 687–694, 1994.
- [137] S. P. Singh and R. S. Sutton, Reinforcement learning with replacing eligibility traces, *Machine Learning*, vol. 22, no. 1, pp. 123–158, 1996.
- [138] R. E. Bellman and R. E. Kalaba, *Dynamic programming and modern control theory*, Academic Press New York, 1965.

- [139] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, Athena Scientific, 1996.
- [140] A. G. Barto, R. S. Sutton, and C. W. Anderson, Neuronlike adaptive elements that can solve difficult learning control problems, *Artificial neural networks: concept learning*, pp. 81–93, 1990.
- [141] C. J. C. H. Watkins and P. Dayan, Technical note Q-learning, *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992.
- [142] C. T. Lin and C. S. G. Lee, Reinforcement structure/parameter learning for neural-network-based fuzzy logic control systems, *IEEE Trans on Fuzzy Systems*, vol. 2, no. 1, pp. 46–63, 1994.
- [143] C. T. Lin and C. P. Jou, Ga-based fuzzy reinforcement learning for control of a magnetic bearing system, *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 30, no. 2, pp. 276–289, 2000.
- [144] I. Ahrns J. Bruske and G. Sommer, An integrated architecture for learning of reactive behaviors based on dynamic cell structures, *Robotics and Autonomous Systems*, vol. 22, pp. 87–101, 1997.
- [145] H. Rak and H. S. Cho, A sensor-based navigation for a mobile robot using fuzzy logic and reinforcement learning, *IEEE Trans. Systems, Man, and Cybernetics*, vol. 25, no. 3, pp. 464–477, 1995.
- [146] R. M. Kandadai and J. M. Tien, A knowledge-base generating hierarchical fuzzy-neural controller, *IEEE Trans. Neural Networks*, vol. 8, no. 6, pp. 1531–1541, 1997.
- [147] H. Tong and T. X. Brown, Reinforcement learning for call admission control and routing under quality of service sonstraints in multimedia networks, *Machine Learning*, vol. 49, no. 2-3, pp. 111–139, 2002.
- [148] R. J. Williams, Simple statistical gradient-following algorithms for connectionist reinforcement learning, *Machine Learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

- [149] M. S. Kim, S. G. Hong, and J. J. Lee, Self-organizing fuzzy inference system by Q-learning, in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1999, vol. 1.
- [150] Y. Takahashi, M. Takeda, and M. Asada, Improvement continuous valued Q-learning and its application to vision guided behavior acquisition, in *Proc. RoboCup 2000*, 2000, pp. 385–390.
- [151] Y. Maeda, Modified Q-learning method with fuzzy state division and adaptive rewards, in *Proc. the 2002 IEEE world congress on computational intelligence*, 2002, vol. 2, pp. 1556–1561.
- [152] T. Horiuchi, A. Fujino, O. Katai, and T. Sawaragi, Fuzzy interpolation-based Q-learning with profit sharing plan scheme, in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1997, vol. 3, pp. 1707–1712.
- [153] A. G. Barto and M. I. Jordan, Gradient following without backpropagation in layered networks, in *IEEE First Int. Conf. Neural Networks*, 1987, vol. 2, pp. 629–636.
- [154] G. A. Rummery and M. Niranjan, Online Q-learning using connectionist systems, Technical report, Engineering Department, Cambridge University, 1994.
- [155] P. Ritthiprava, T. Maneewarn, D. Laowattana, and J. Wyatt, A modified approach to fuzzy-q learning for mobile robots, in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 2004, vol. 3, pp. 2350–2356.
- [156] G. Hailu and G. Sommer, Learning by biasing, in *IEEE Inter. Conf. on robotics and automation*, Belgium, Leuven, 1998, pp. 2168–2173.
- [157] S. A. K-Team, Switzerland, *Khepera 2 user manual*, 2002.
- [158] T. Nilsson, [online]. available: <http://www.kiks.f2s.com>.
- [159] M. J. Er and C. Deng, Obstacle avoidance of a mobile robot using hybrid learning approach, *IEEE Trans. on Industrial Electronics*, vol. 52, no. 3, pp. 898–905, 2005.

- [160] M. J. Er, Z. Li, H. Cai, and Q. Chen, Adaptive noise cancellation using enhanced dynamic fuzzy neural networks., *IEEE Trans. Fuzzy Systems*, vol. 13, no. 3, pp. 331–342, 2005.
- [161] T. H. Stephan and K. Ben, Q-learning for systems with continuous state and action spaces., in *10th Belgian-Dutch Conference on Machine Learning*, 2000.
- [162] T. Horiuchi, A. Fujino, O. Katai, and T. Sawaragi, Fuzzy interpolation-based Q-learning with continuous states and actions, in *the Fifth IEEE Int. Conference on Fuzzy System*, New Orleans, LA, USA, 1996, pp. 594–600.
- [163] J. N. Tsitsiklis and B. V. Roy, Feature-based methods for large scale dynamic programming, *Maching Learning*, vol. 22, no. 1-3, pp. 59–94, 1996.
- [164] C. Karr, Applying genetics to fuzzy logic, *AI Expert*, vol. 6, no. 3, pp. 38–43, 1991.
- [165] D. A. Linkens and H. O. Nyongesa, Genetic algorithms for fuzzy control: offline/online system development and application, in *Control Theory and Applications*, 1995, vol. 142, pp. 161–185.
- [166] O. Cordon and F. Herrera, *A hybrid genetic algorithms evolution strategy process for learning fuzzy logic controller knowledge bases*, chapter Genetic algorithms and soft computing, pp. 251–278, Physica Verlag, Berlin, Germany, 1996.
- [167] O. Cordon, F. Herrera, F. Gomide, F. Hoffmann, and L. Magdalena, Ten years of genetic fuzzy systems: current framework and new trends, *Fuzzy sets and systems*, vol. 141, pp. 5–31, 2004.
- [168] Z. Michalewicz, *Genetic algorithms + data structures = evolution programs (3rd ed.)*, Springer-Verlag, London, UK, 1996.
- [169] R. A. McCallum, Hidden state and reinforcement learning with instance-based state identification, *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 26, no. 3, pp. 464–473, 1996.

-
- [170] R. Sun and C. Sessions, Self-segmentation of sequences: automatic formation of hierarchies of sequential behaviors, *IEEE Trans. Systems, Man and Cybernetics, Part B*, vol. 30, no. 3, pp. 403–418, 2000.
- [171] S. D. Whitehead and L. J. Lin, Reinforcement learning of non-markov decision processes, *Artificial Intelligence*, vol. 73, no. 1-2, pp. 271–306, 1995.